



BA IT-Security

Chapter 1: Introduction

Prof. Dr.-Ing. Ulrike Meyer



Overview on Chapter 1

Security goals, attacks, mechanisms and services

- ▶ Definition of security goals
- ▶ Types of attacks that threaten them
- ▶ Examples for these attacks
- ▶ Definition of security mechanisms and services
- ▶ Examples

Vulnerabilities exploited by attacks

- ▶ Levels of a system on which vulnerabilities occur
- ▶ Examples of typical vulnerabilities on each level

Attackers

- ▶ Types of attackers
- ▶ Motivation of attackers

Overview on the rest of the lecture

- ▶ Overall structure
- ▶ Connections
- ▶ Further lectures and other teaching activities

Overview

Security goals, attacks, mechanisms and services

- ▶ Definition of security goals
- ▶ Types of attacks that threaten them
- ▶ Examples for these attacks
- ▶ Definition of security mechanisms and services
- ▶ Examples

Vulnerabilities exploited by attacks

- ▶ Levels of a system on which vulnerabilities occur
- ▶ Examples of typical vulnerabilities on each level

Attackers

- ▶ Types of attackers
- ▶ Motivation of attackers

Overview on the rest of the lecture

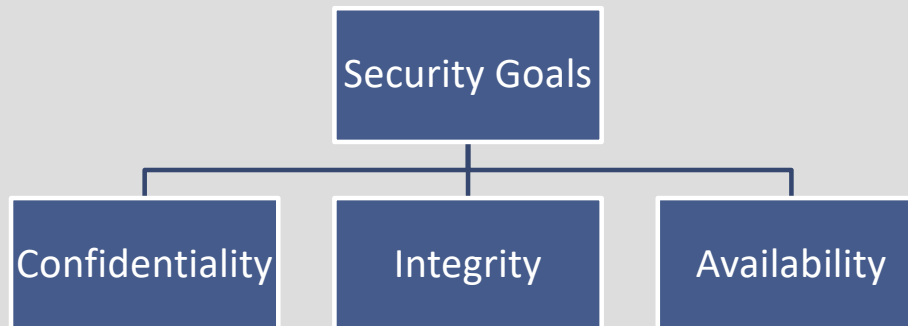
- ▶ Overall structure
- ▶ Connections
- ▶ Further lectures and other teaching activities

Definition of IT-Security and Security Goals

IT-Security comprises all measures to

prevent, detect, mitigate, or deter **attacks** against **confidentiality**, **integrity**, or **availability** of an asset in a system, including data, software, hardware, and networks.

An **attack** is thus any action that compromises one of the three main security goals



Definition of IT-Security and Security Goals

Confidentiality

Only authorized entities can access assets in a system

Integrity

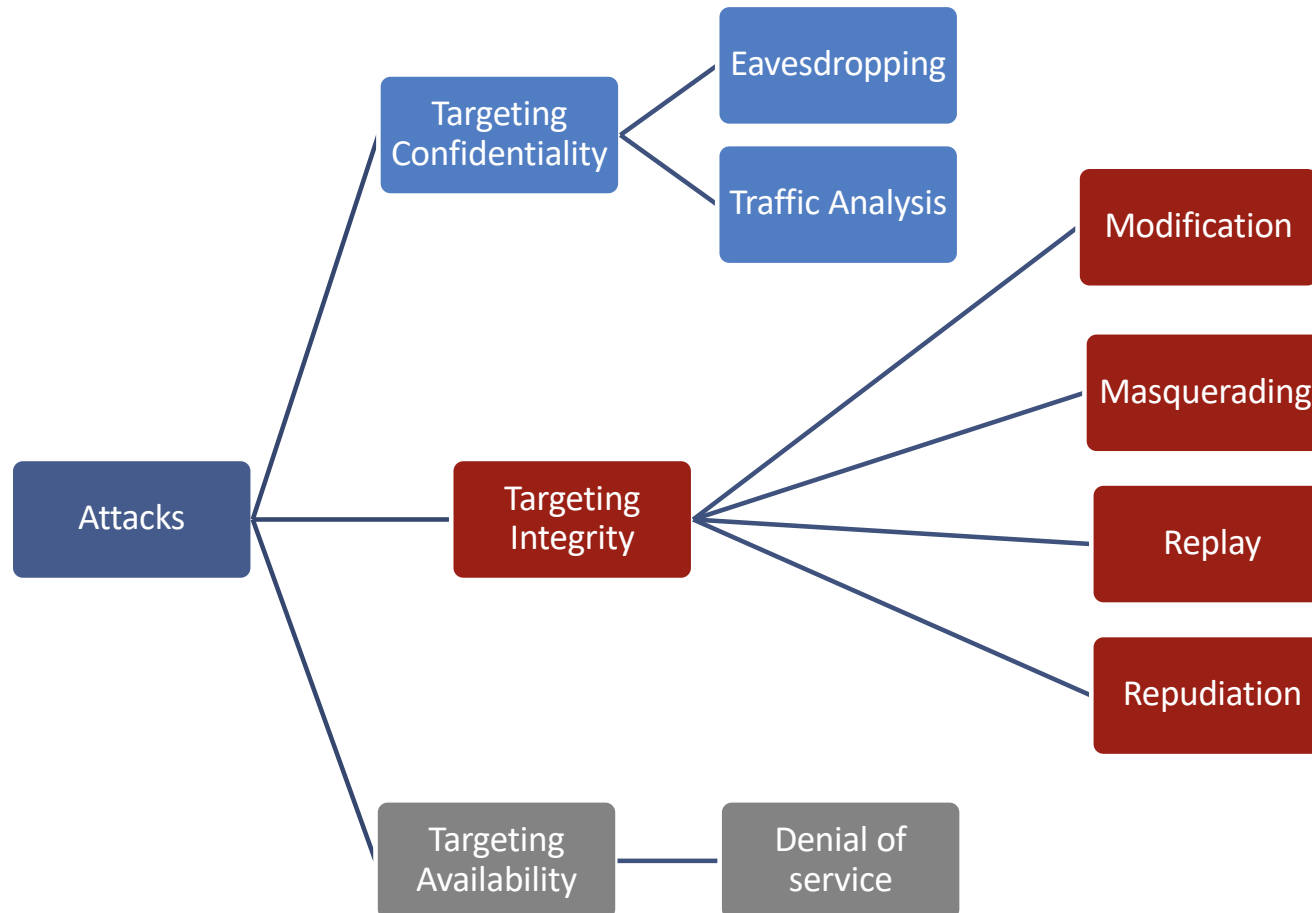
Only authorized entities can make changes assets in a system

Availability

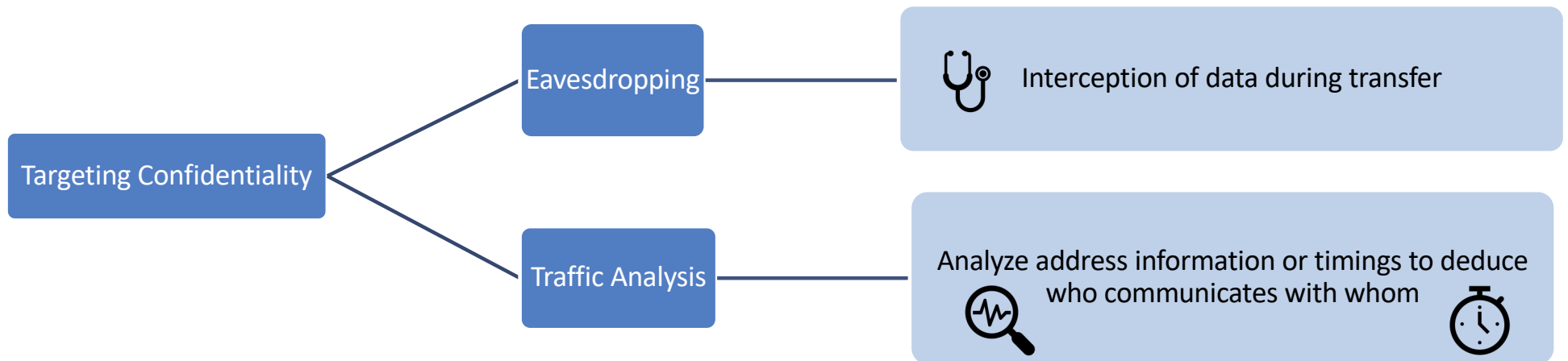
Authorized entities can access assets in a system as intended

Collectively referred to as **CIA**

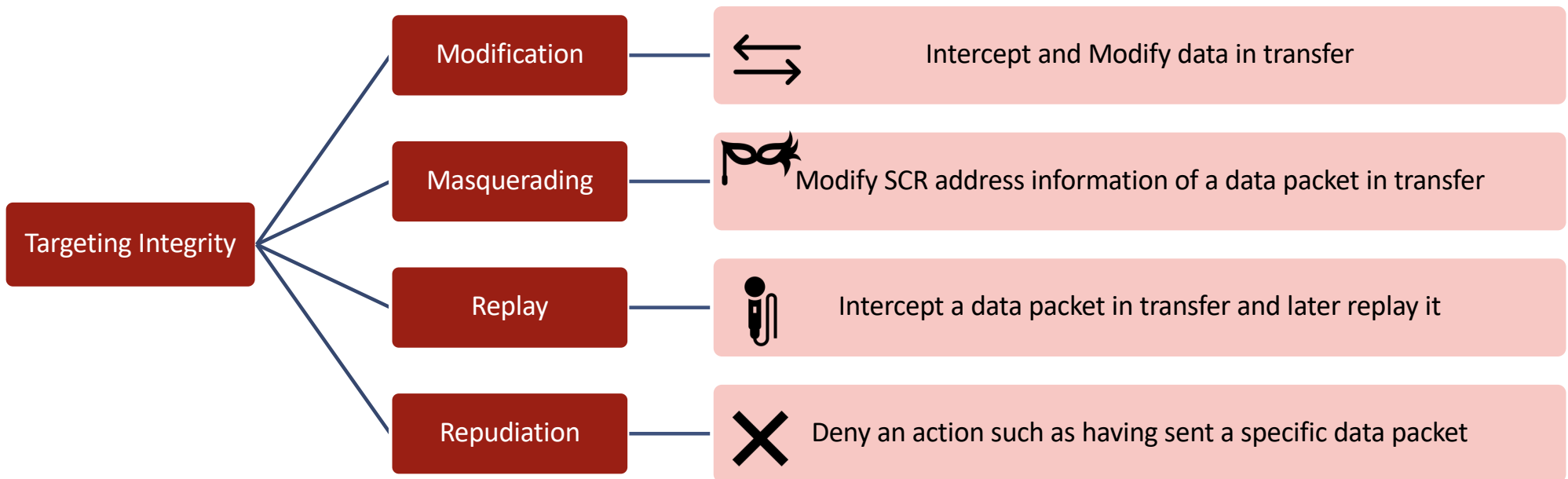
Example Types of Attacks per Goal



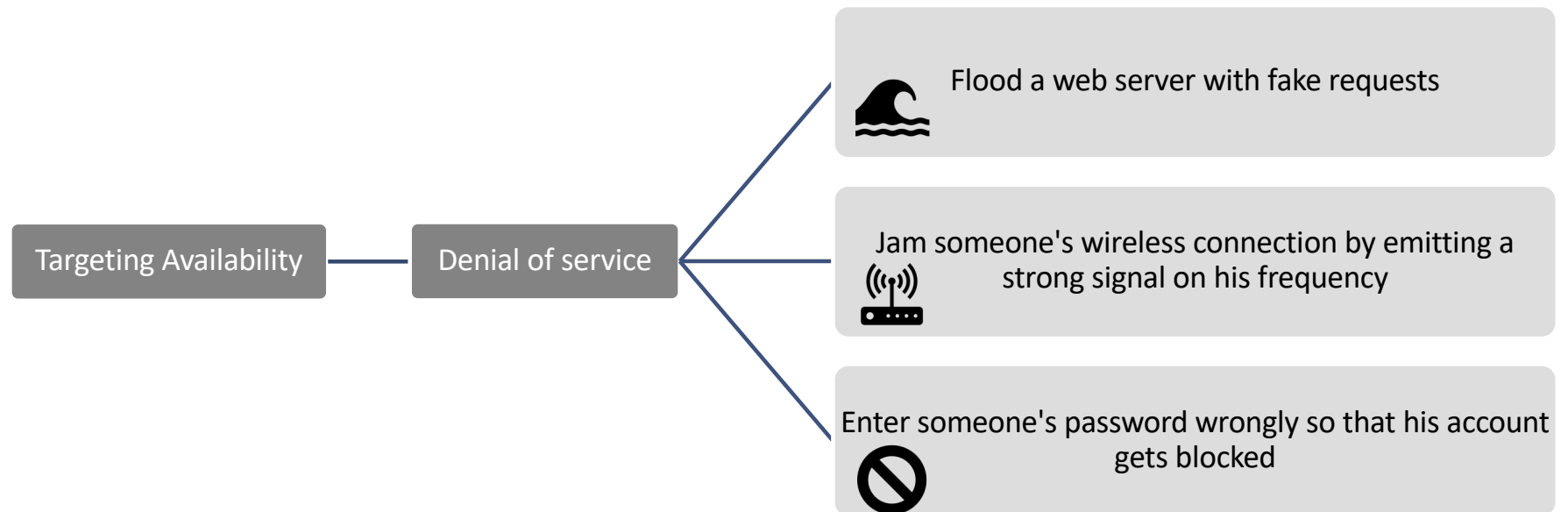
Example Attacks Against Confidentiality



Example Attacks Against Integrity



Example Attacks Against Availability



Attack Examples



Attack against Availability

Hackers Flood NPM with Bogus Packages Causing a DoS Attack

📅 Apr 10, 2023 [Software Security / JavaScript](#)

Threat actors flooded the npm open source package repository for Node.js with bogus packages that briefly even resulted in a...



Attack against Availability

LockBit Ransomware Now Targeting Apple macOS Devices

📅 Apr 18, 2023 [Encryption / Malware](#)

Threat actors behind the LockBit ransomware operation have developed new artifacts that can encrypt files on devices...

Examples taken from <https://thehackernews.com/>

Attack Examples



Attack against Integrity, Confidentiality

Goldoson Android Malware Infects Over 100 Million Google Play Store Downloads

📅 Apr 18, 2023 [Mobile Security / Hacking](#)

The rogue component is part of a third-party software library used by the apps in question and is capable of gathering information about installed apps, Wi-Fi and Bluetooth-connected devices, and GPS locations.



Attack against Confidentiality, Integrity, Availability

New Atomic macOS Malware Steals Keychain Passwords and Crypto Wallets

📅 Apr 28, 2023 [Endpoint Security / Cryptocurrency](#)

Threat actors are advertising a new information stealer for the Apple macOS operating system called Atomic macOS Stealer...

Examples taken from <https://thehackernews.com/>

Security Mechanisms and Services

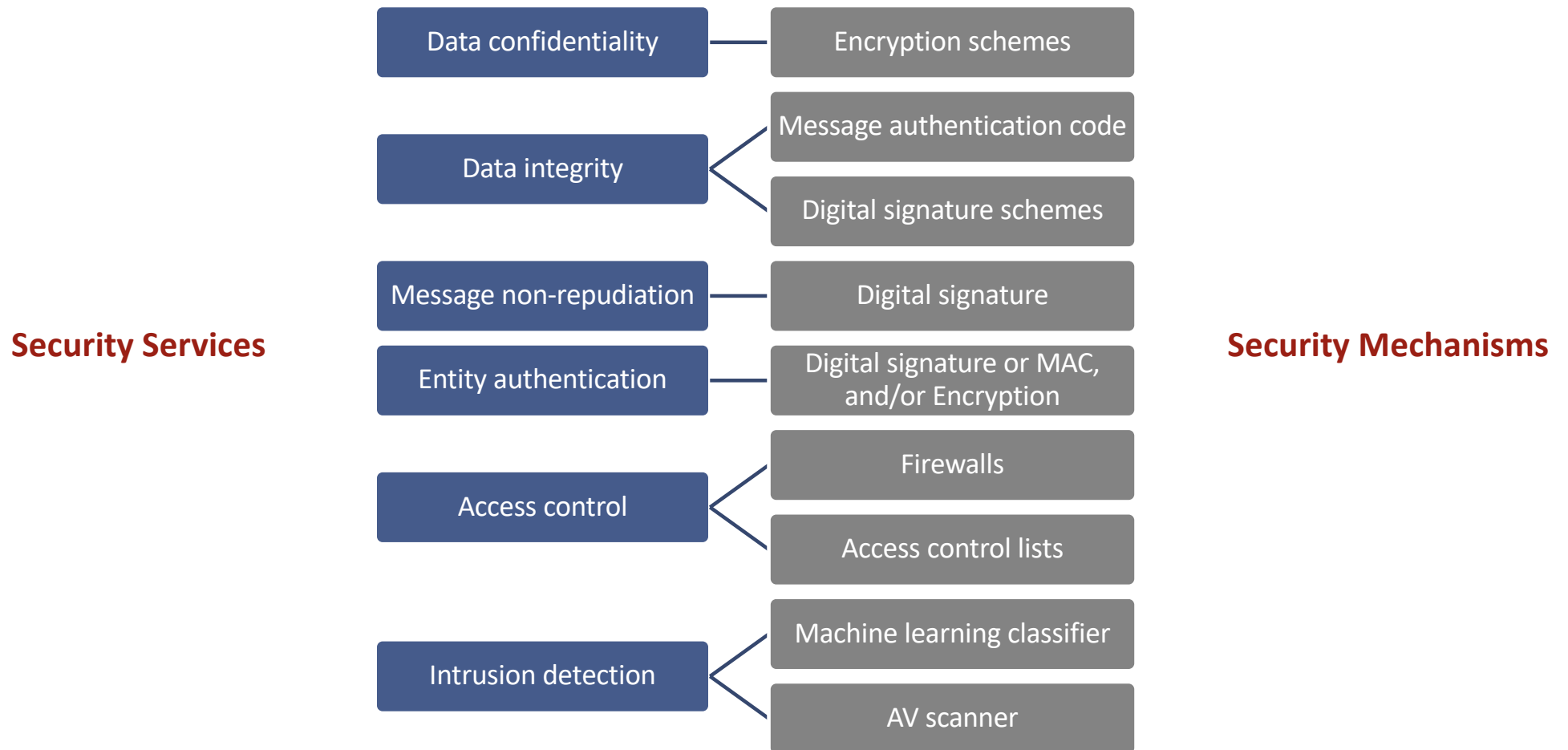
Security Mechanism

- ▶ A mechanism that is designed to detect, prevent, recover from, or deter an attack against an asset in a system

Security Service

- ▶ A service that protects the security goals of assets in a system
- ▶ A security service makes use of one or more security mechanisms

Examples for Security Services and Example Security Mechanisms



Overview

Security goals, attacks, mechanisms and services

- ▶ Definition of security goals
- ▶ Types of attacks that threaten them
- ▶ Examples for these attacks
- ▶ Definition of security mechanisms and services
- ▶ Examples

Vulnerabilities exploited by attacks

- ▶ Levels of a system on which vulnerabilities occur
- ▶ Examples of typical vulnerabilities on each level

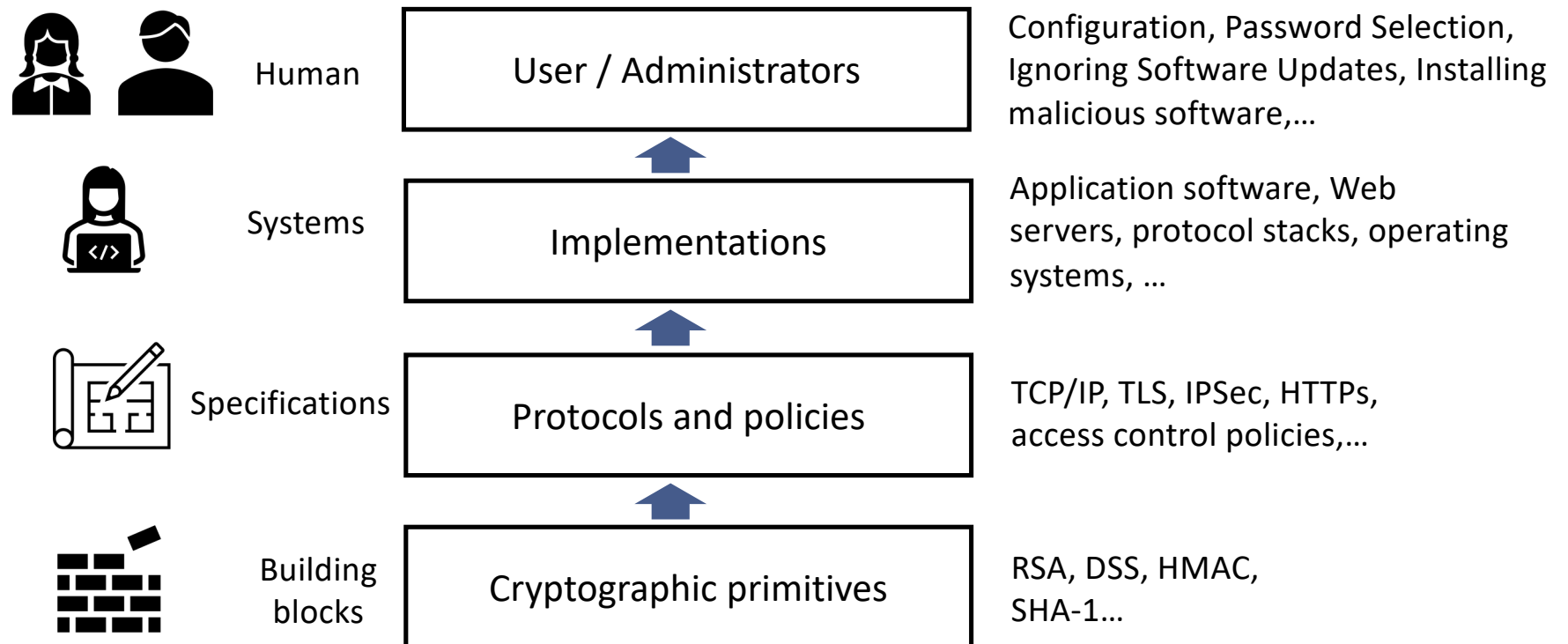
Attackers

- ▶ Types of attackers
- ▶ Motivation of attackers

Overview on the rest of the lecture

- ▶ Overall structure
- ▶ Connections
- ▶ Further lectures and other teaching activities

Attacks make use of Vulnerabilities on all Levels of a System



All defense mechanisms on all layers can be targeted and must interact properly

Broken Building Blocks

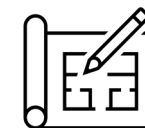
Classical Examples

- ▶ Encryption algorithms used in 2G mobile networks (A5/2, A5/1...)
- ▶ RC4 Encryption algorithms used in WLAN, TLS, IPsec,....
- ▶ Cryptographic hash functions MD5, SHA1
 - Used, e.g., in TLS



• Typical Solution: Integrate multiple algorithms to choose

- ▶ New attacks on ciphers cannot be prevented
- ▶ Include multiple algorithms as “mandatory” to support in protocol specifications
- ▶ Allow for an easy integration of additional algorithms
- ▶ Configure your system to use secure algorithm if an algorithm is broken



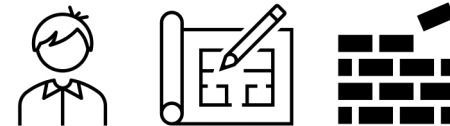
New Problem: Secure Algorithm Selection

Need to agree on the algorithm to be used on specific connection

- ▶ Algorithm negotiation must be protected

Typical Approach

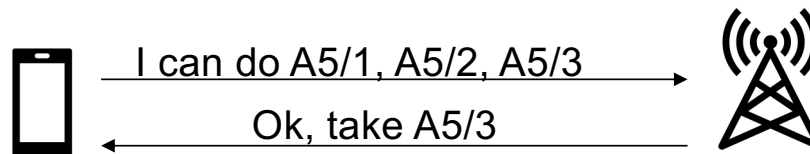
- ▶ Parties exchange information on which algorithms they support
- ▶ One of the algorithms both support is selected
- ▶ Information exchanged needs to be protected against manipulation
 - Problem: algorithms, e.g., for integrity protection have not been selected yet



Example for Insecure Algorithm Negotiation

Insecure negotiation leads to downgrading attacks

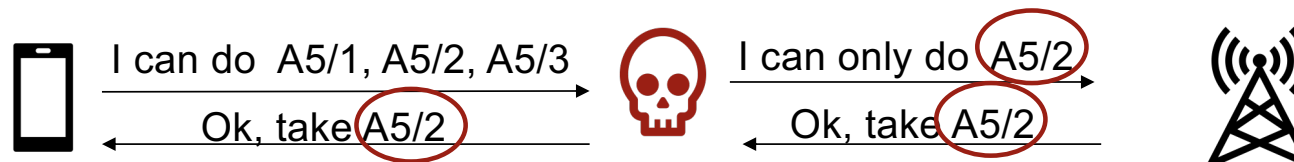
- ▶ Attacker can downgrade the negotiation to a broken algorithm



A5/1, A5/2, A5/3

- ▶ encryption algorithms supported in 2G mobile networks
- ▶ A5/2 totally broken since 2001

Broken!

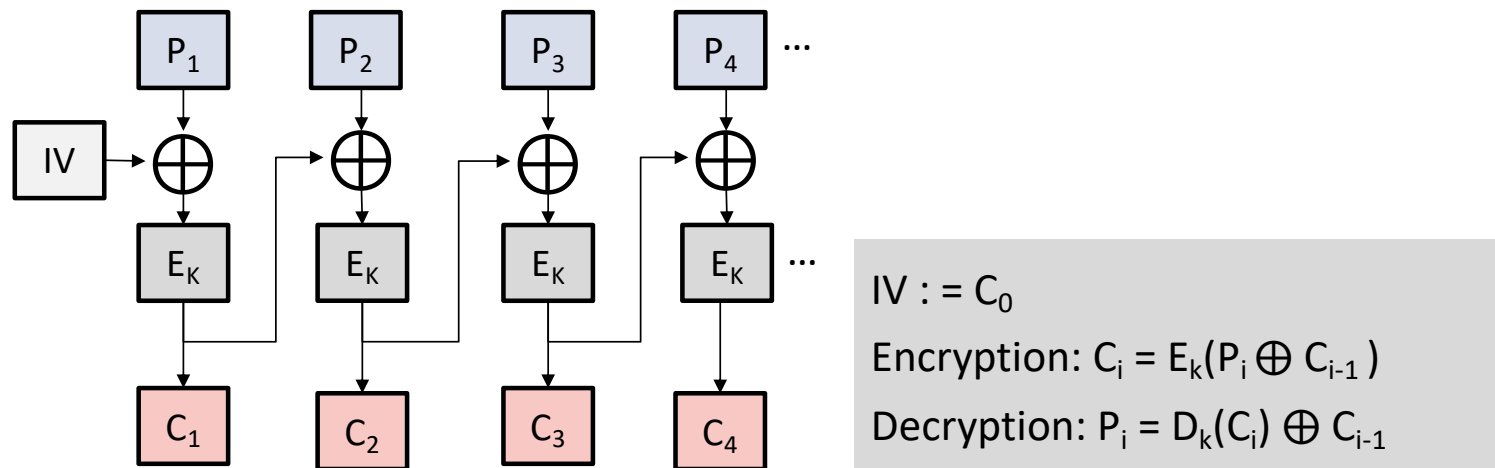


- ▶ Works because integrity of messages indicating the supported algorithms is not protected
- ▶ Broken algorithm often still supported to service old devices that only support old mechanisms
 - Backward Compatibility

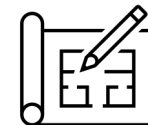


Problem: **ALL** Building Blocks Need to be Negotiable

Broken CBC Mode of Encryption for Symmetric Ciphers



- ▶ If CBC Mode is used, then in some application settings it is possible to decrypt messages even if the underlying encryption algorithm E_k is secure
- ▶ **All mandatory TLS 1.2** ciphers used CBC-Mode



Typical Sources for Vulnerabilities in Protocols and Specifications

• Design Flaws

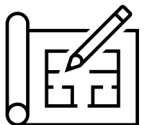
- ▶ E.g. WEP: wired equivalent privacy problem in Wireless LAN (2001)
 - Authentication breaks after simple eavesdropping on one authentication protocol run
 - Weak encryption, no integrity protection
 - ...

• Backward Compatibility Problems

- ▶ If devices support different versions of a protocol, downgrading to an older version is often possible
 - Attack pretends that one of the communicating endpoints does not support newer version

• Incomplete Specifications

- ▶ Krack-attack against WLAN (2017)
 - Problem in the protocol design: unspecified how to handle unexpected messages



Typical Sources for Vulnerabilities in Implementations

- **Software vulnerabilities**

- ▶ Buffer overflows, Format string vulnerabilities, XSScripting,...
- ▶ Bugs like the OpenSSL bug: implementation problem on Debian-based systems (2006)
 - Lead to only 32,767 ($= 2^{15} - 1$) different SSH-keys
 - Not a vulnerability in the protocol design
 - “Just” a problem in the implementation of the pseudo-random function
- ▶ Using malicious libraries or insecure code fragments of others

- **Update life cycles**

- ▶ Software vulnerabilities can typically not be entirely avoided
- ▶ Updates that patch vulnerabilities need to be published and deployed

- **Insecure default settings**

- ▶ E.g., if IoT device ships with a default admin password and does not require changing it



Examples for Users and Administrators as Vulnerabilities

- **Failing to update available software patches**

- ▶ More and more automated but still many software vulnerabilities exploited although patches are available

- **Deliberately installing malicious software**

- ▶ Typically, unintended
 - Trojans: malicious software masquerading as benign application
 - Clicking on a malicious attachment
 - Installing free software from dubious sources

- **Social Engineering**


- ▶ Talking someone into revealing their password
- ▶ Luring someone on a fake website and making them enter their login data



Example Social Engineering




Called to an urgent Zoom meeting with HR? It might be a phishing attack

 Graham Cluley • [@gcluley](#)
10:15 am, April 26, 2020



Coronavirus phishing attack disguises as a message from the Center for Disease Control

 Graham Cluley • [@gcluley](#)
12:36 pm, February 10, 2020



Example: Malware delivered with Social Engineering

Hackers disguise malware attack as new details on Donald Trump's COVID-19 illness



GRAHAM CLULEY

[Follow @gcluley](#)

OCT 8, 2020

IT SECURITY AND DATA PROTECTION



Overview

Security goals, attacks, mechanisms and services

- ▶ Definition of security goals
- ▶ Types of attacks that threaten them
- ▶ Examples for these attacks
- ▶ Definition of security mechanisms and services
- ▶ Examples

Vulnerabilities exploited by attacks

- ▶ Levels of a system on which vulnerabilities occur
- ▶ Examples of typical vulnerabilities on each level

Attackers

- ▶ Types of attackers
- ▶ Motivation of attackers

Overview on the rest of the lecture

- ▶ Overall structure
- ▶ Connections
- ▶ Further lectures and other teaching activities

Types and motivation of attackers

All of them can be insiders or outsider attacker

• Criminals and Hackers-for-hire



- ▶ Making money as main motivation
 - Stealing and selling login credentials, trade secrets, personal data, ...
 - Extortion, e.g., by threatening to publish stolen data or to stage a denial-of-service attack,...
 - Spreading spam
 - Get paid for exploits, malware, bots,...

• Crackers and Hacktivists



- ▶ Achieve Fame and glory in the blackhat community
- ▶ Claim to crack for the greater good



• Secret service and military personal

- ▶ Cyber attacks and defenses



• End users

- ▶ That do not adequately protect their computers

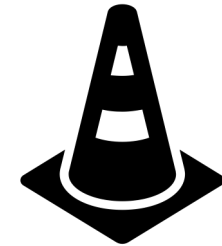


• Pentesters

- ▶ Try to break into systems on demand
 - with explicit consent of system operator
- ▶ Reveal and help to fix exposed vulnerabilities

Summary

- **Attacks typically threaten one or more of the CIA security goals**
 - ▶ Confidentiality, Integrity, Availability
- **Attacks can exploit vulnerabilities on all levels of a system**
 - ▶ Security mechanisms required on each level
 - ▶ Mechanisms on different levels must interact properly
- **Security mechanisms and services aim at protecting against attacks by**
 - ▶ prevention, detection, mitigation, or deterrence
- **Attackers vary greatly with respect to**
 - ▶ Their motivation
 - ▶ Their power w.r.t. their skills, knowledge on / access to the target, and their computational resources,...



Overview

Security goals, attacks, mechanisms and services

- ▶ Definition of security goals
- ▶ Types of attacks that threaten them
- ▶ Examples for these attacks
- ▶ Definition of security mechanisms and services
- ▶ Examples

Vulnerabilities exploited by attacks

- ▶ Levels of a system on which vulnerabilities occur
- ▶ Examples of typical vulnerabilities on each level

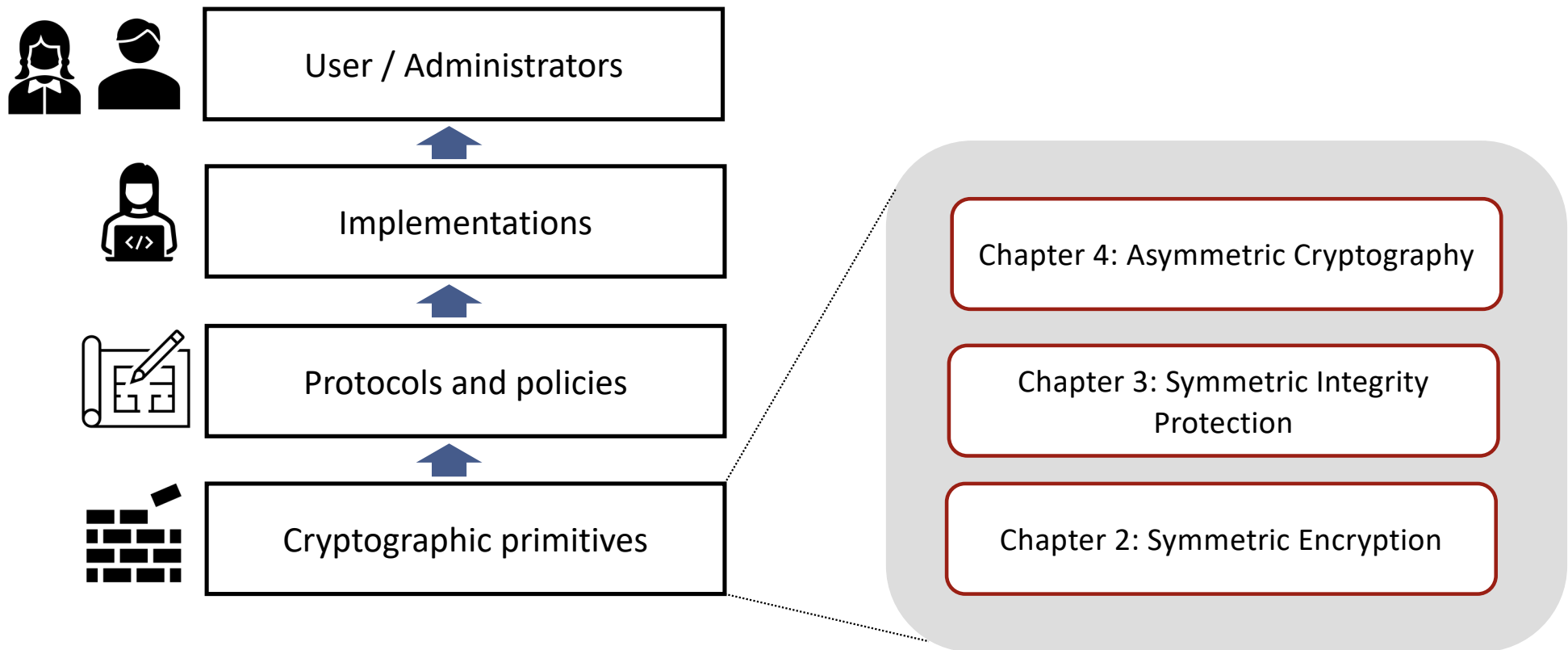
Attackers

- ▶ Types of attackers
- ▶ Motivation of attackers

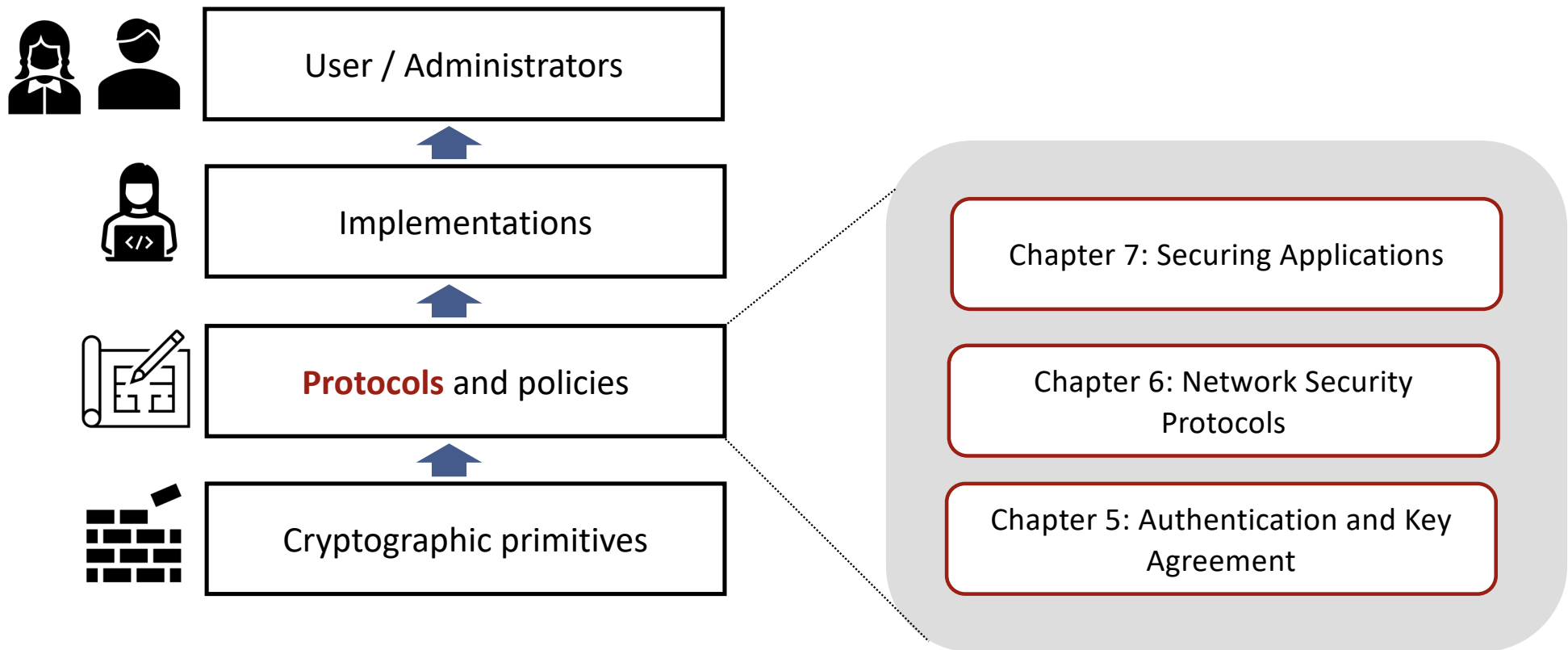
Overview on the rest of the lecture

- ▶ Overall structure
- ▶ Connections
- ▶ Further lectures and other teaching activities

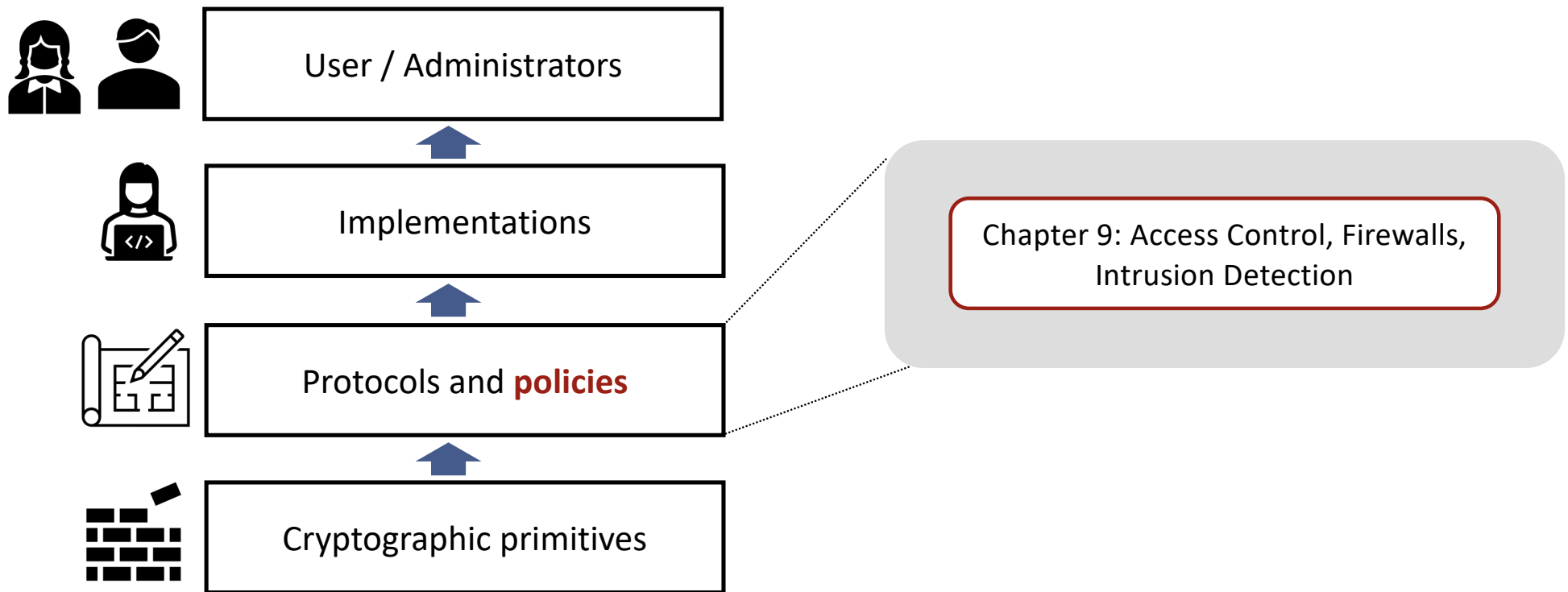
Cryptographic Primitives



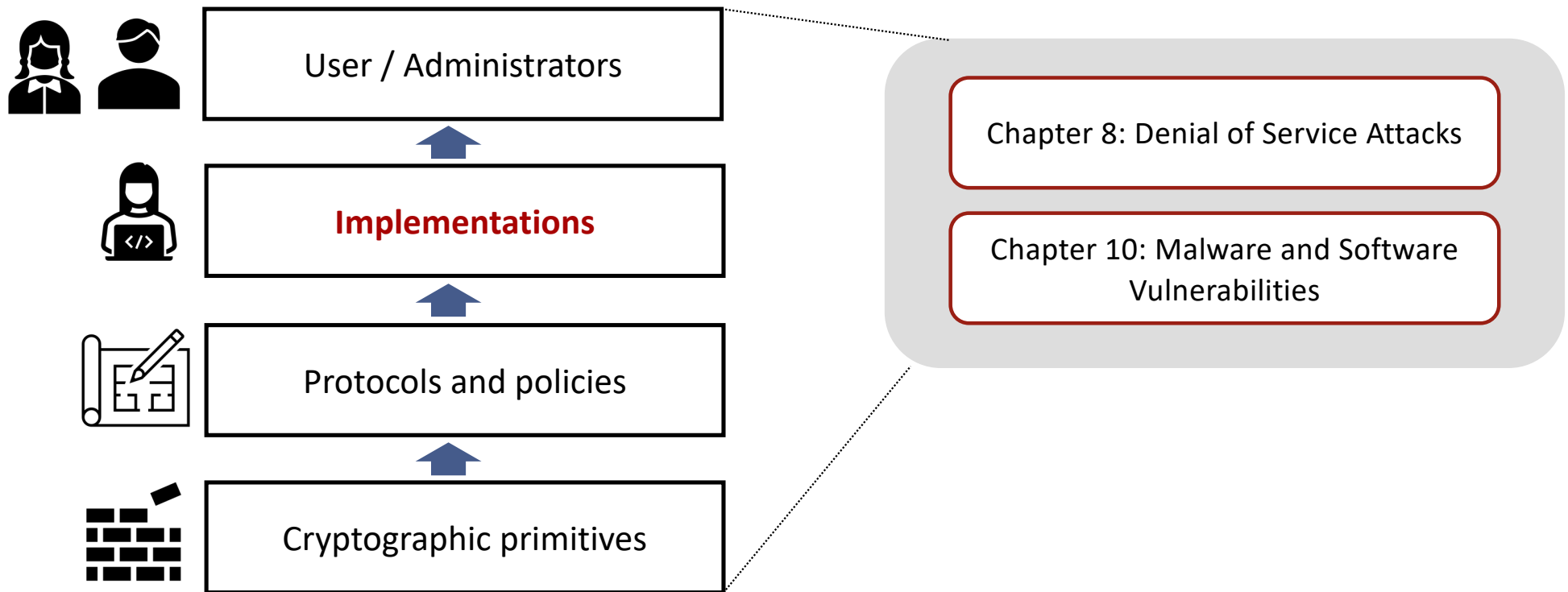
Cryptographic Primitives



Cryptographic Primitives



Cryptographic Primitives



Cryptographic vs. Non-Cryptographic Protection

Cryptographic Protection against Attacks on Confidentiality and Integrity

Chapter 4: Asymmetric Cryptography

Chapter 3: Symmetric Integrity Protection

Chapter 2: Symmetric Encryption

Chapter 5: Authentication and Key Agreement

Chapter 6: Network Security Protocols

Chapter 7: Securing Applications

Most Prominent Example Attacks that cannot be prevented / detected by cryptographic means alone

Chapter 8: Denial of Service Attacks

Chapter 10: Malware and Software Vulnerabilities

Non-Cryptographic Protection against Attacks on Confidentiality, Integrity, and Availability

Chapter 9:
Access Control,
Firewalls,
Intrusion Detection

References

- **IETF RFC 4949: Security Glossary**
- **W. Stallings, Cryptography and Network Security: Principles and Practice, 8th edition, Pearson 2022**
 - ▶ Chapter 1: Information and Network Security Concepts



IT-Security

Chapter 2: Symmetric Encryption

Prof. Dr.-Ing. Ulrike Meyer



Overview

• Introduction

- ▶ Intuition
- ▶ Formal definition
- ▶ Historic examples

• Computational Security

- ▶ Attacker models
 - Knowledge
 - Goal
 - Strategy

• Perfect Secrecy

- ▶ Definition
- ▶ Shannon's theorem
- ▶ One-time-pad

• Practical Schemes

- ▶ Stream ciphers
- ▶ Block ciphers
- ▶ Modes of encryption

What is an encryption scheme

Can a cipher be perfectly secure?

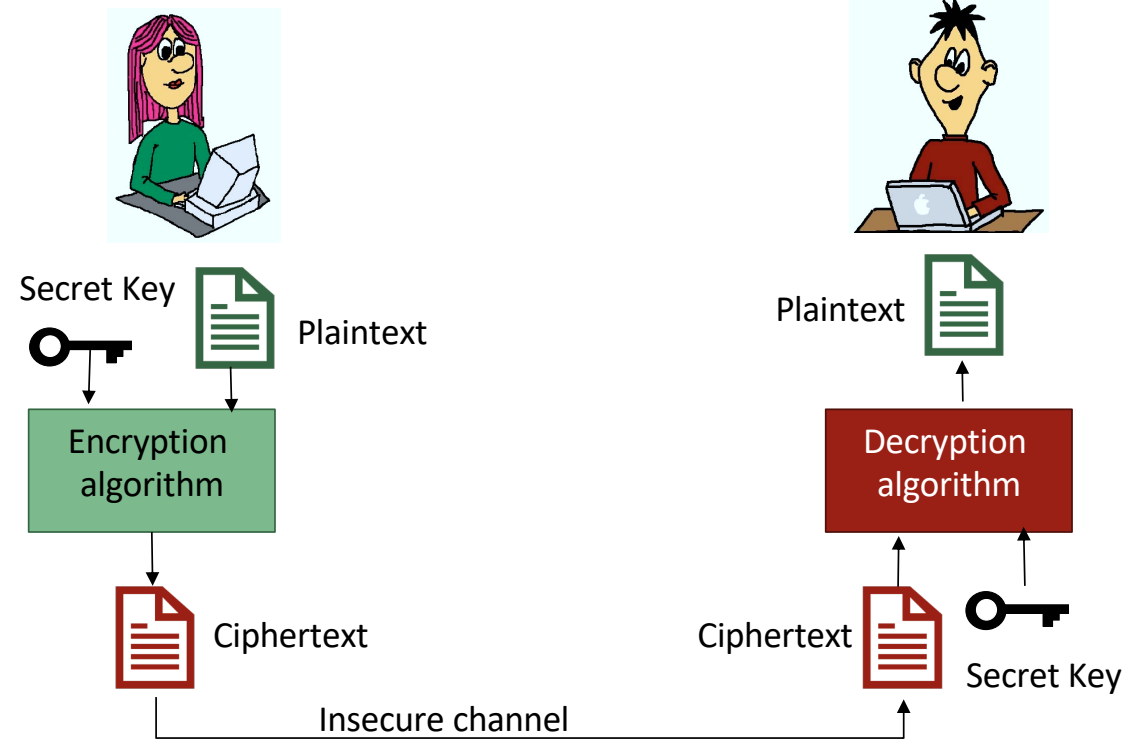


How can we model attackers?

How do modern ciphers work and how are they used?

Intuition on Symmetric Ciphers

- Alice wants to send a **confidential** plaintext to Bob
- Alice and Bob **share** a **secret key**
- Alice uses the key to **encrypt** plaintext to ciphertext
- Bob uses the key to **decrypt** ciphertext to plaintext
- Decryption is **"difficult"** without the key



Formal Definition of Encryption Scheme

- An encryption scheme is a five-tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ consisting of
 - ▶ The plaintext space \mathcal{P} of **plaintexts** (e.g., $\mathcal{P} = \{0,1\}^n$ for some $n \in \mathbb{N}$)
 - ▶ The cipher space \mathcal{C} of **ciphertexts** (e.g., $\mathcal{C} = \{0,1\}^m$ for some $m \in \mathbb{N}$)
 - ▶ A key space \mathcal{K} of **keys** (e.g., $\mathcal{K} = \{0,1\}^k$ for some $k \in \mathbb{N}$)
 - ▶ A family $\mathcal{E} = \{E_K: K \in \mathcal{K}\}$ of functions $E_K: \mathcal{P} \rightarrow \mathcal{C}$ called **encryption functions**
 - ▶ A family $\mathcal{D} = \{D_K: K \in \mathcal{K}\}$ of functions $D_K: \mathcal{C} \rightarrow \mathcal{P}$ called **decryption functions**
- Such that for any $K_1 \in \mathcal{K}$ there is a $K_2 \in \mathcal{K}$ such that
 - ▶ For all $P \in \mathcal{P}$ it holds that $D_{K_2}(E_{K_1}(P)) = P$
- In a **symmetric encryption** scheme the encryption and decryption keys are the same
- Note that this definition does not cover any notion of security yet

Kerckhoff's Principle 1883

A cryptosystem should be secure even if everything about the system, **except the key**, is public knowledge



- **In contrast:**

- ▶ Keeping the design of a cryptosystem secret is often referred to as **“security by obscurity”**

Example Caesar Cipher

- **The cipher**

- ▶ Plaintext space = ciphertext space = {A,..., Z}, Key space = {1,...,25}
- ▶ Replace each plaintext letter with the one **k** letters after it. E.g., for $k = 4$

Plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M
Ciphertext	E	F	G	H	I	J	K	L	M	N	O	P	Q

- **Security of the Caesar cipher**

Plaintext	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Ciphertext	R	S	T	U	V	W	X	Y	Z	A	B	C	D

- ▶ Assume a message has been encrypted letter by letter using the Cesar cipher
- ▶ Try out each of the 25 keys and check if the resulting plaintext makes sense
 - Requires **recognizable plaintext**
- ▶ **The key space is too small!**

 **A secure cipher requires a large key space**

Brute Force Attack on the Caesar Cipher

Plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M
Ciphertext	E	F	G	H	I	J	K	L	M	N	O	P	Q

Plaintext	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Ciphertext	R	S	T	U	V	W	X	Y	Z	A	B	C	D



k=1? WIGYVMXC
 k=2? VHFXULWB
 k=3? UGEWTKVA
 k=4? TFDVSJUY
 k=5? SECURITY
 ... RDBTQHSX

- If the message is **short, multiple keys** may lead to **sense making plaintexts**
- If the message is long enough, on **average** key found after $\frac{1}{2} |\mathcal{K}|$ tries
- Brute force attacks are also known as **exhaustive search attacks**

Monoalphabetic Substitution Cipher

- **Idea**

- ▶ Replace each plaintext letter with one specific other letter according to a substitution table
- ▶ Plaintext space = ciphertext space = {A,...Z}
- ▶ Key space = **all permutations** of the letters A,..., Z
- ▶ Size of the key space: $|\mathcal{K}| = 26! = 4.0329146 \cdot 10^{26}$

- **Example**

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

- **Trying out each possible key is quite time consuming!**

Exhaustive Search for Monoalphabetic Ciphers

- **Let's assume we**

- ▶ Can decrypt 5 characters per ms
- ▶ Need to decrypt 100 characters to be sure we found the right key

- **Then we will on average need $\frac{1}{2} \cdot \frac{100}{5} \cdot |\mathcal{K}| = \frac{1}{2} \cdot 20 \cdot |\mathcal{K}|$ ms to find the right key**

- ▶ That is $10 \cdot 4.0329146 \cdot 10^{26}$ ms = $4.0329146 \cdot 10^{27}$ ms = $4.0329146 \cdot 10^{24}$ s = $6.7215243 \cdot 10^{22}$ min
= $1.2788288 \cdot 10^{17}$ years

- **Let's assume we**

- ▶ Can decrypt 500 000 characters per ms and still need to decrypt 100 characters in order to be sure

- **Then we will on average need $\frac{1}{2} \cdot \frac{100}{500\,000} \cdot |\mathcal{K}| = \frac{1}{2} \cdot \frac{1}{5\,000} \cdot |\mathcal{K}|$ ms to find the right key**

- ▶ That is $10^{-4} \cdot 4.0329146 \cdot 10^{26}$ ms = $4.0329146 \cdot 10^{22}$ ms = $4.0329146 \cdot 10^{19}$ s = $6.7215243 \cdot 10^{17}$ min
= $1.2788288 \cdot 10^{12}$ years

Difficulty of exhaustive search depends on

- ▶ size of key space
- ▶ resources of attacker

Example Letter Frequencies

- For any given language and text basis one can determine the relative letter frequencies

Letter	ENG	GER	Letter	ENG	GER	Letter	ENG	GER
A	8.167%	6.516%	J	0.153%	0.268%	S	6.327%	7.270%
B	1.492%	1.886%	K	0.772%	1.417%	T	9.056%	6.154%
C	2.782%	2.732%	L	4.025%	3.437%	U	2.758%	4.166%
D	4.253%	5.076%	M	2.406%	2.534%	V	0.978%	0.846%
E	12.702%	16.396%	N	6.749%	9.776%	W	2.360%	1.921%
F	2.228%	1.656%	O	7.507%	2.594%	X	0.150%	0.034%
G	2.015%	3.009%	P	1.929%	0.670%	Y	1.974%	0.039%
H	6.094%	4.577%	Q	0.095%	0.018%	Z	0.074%	1.134%
I	6.966%	6.550%	R	5.987%	7.003%			


Top 5 letters in English texts

Letter	ENG
E	12.702%
T	9.056%
A	8.167%
O	7.507%
I	6.966%

- Other useful frequencies include, Bigrams, double letters, etc.

Frequency Analysis

- Can be used to
 - ▶ Break any cipher that **preserves frequencies**
 - As long as enough ciphertext is available that has been produced by the same key
- E.g., Monoalphabetic Substitution Ciphers can be broken this way

 A large key space is necessary but does not guarantee a secure cipher



So, how can we get a secure cipher and what does secure mean anyway

Frequency Analysis

- ▶ Given a (long) ciphertext in a known language
- ▶ Count the frequency of each letter occurring in the ciphertext
- ▶ Replace them according to their frequency in the natural language
- ▶ Check if the resulting plaintext makes sense

Example Frequency Analysis on Monoalphabetic Substitution Cipher

- Ciphertext C

- ▶ JW XAR DGZ FDGDPAJE XAR HZOJZTZ BSDB D TZGX ZTJO DBBDCLZG JN ARB BA VZB XAR
- ▶ JW XAR DGZ FDGDPAJE XAR HZOJZTZ BSDB D TZGX ZTJO DBBDCLZG JN ARB BA VZB XAR
- ▶ I? ?O? A?E ?A?A?OI? ?O? ?E?IE?E T?AT A ?E?? E?I? ATTA??E? I? O?T TO ?ET ?O?

Top 5
E
T
A
O
I

Letter in C	Z	B	D	A	J	G	C	L	X	R	W	F	P	E	D	T	O	N	V	H
Frequency	8	7	7	6	5															
Replace with	E	T	A	O	I	R	C	K	Y	U	F	P	N	D	H	V	L	S	G	B

- ▶ I? YOU ARE ?ARA?OI? YOU ?E?IE?E T?AT A ?ERY E?I? ATTACKER I? OUT TO ?ET YOU
- ▶ IF YOU ARE PARANOID YOU BELIEVE THAT A VERY EVIL ATTACKER IS OUT TO GET YOU

- Gives us 20 letters for which the mapping is known, i.e. 76,9% of the key

Overview

• Introduction

- ▶ Intuition
- ▶ Formal definition
- ▶ Historic examples

• Computational Security

- ▶ Attacker models
 - Knowledge
 - Goal
 - Strategy

• Perfect Secrecy

- ▶ Definition
- ▶ Shannon's theorem
- ▶ One-time-pad

• Practical Schemes

- ▶ Stream ciphers
- ▶ Block ciphers
- ▶ Modes of encryption

What is an encryption scheme

Can a cipher be perfectly secure?



How can we model attackers?

How do modern ciphers work and how are they used?

Perfect Secrecy

- **Idea of Shannon**

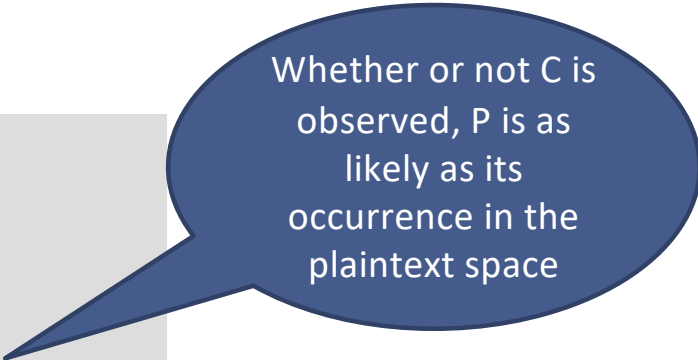
- ▶ A ciphertext should not reveal any new information on the plaintext

Definition:

An encryption scheme is said to provide **perfect secrecy** if

Given a probability distribution \Pr on \mathcal{P} , and $\Pr(P) > 0$ for all plaintexts P

For each $P \in \mathcal{P}, C \in \mathcal{C}$ and $K \in \mathcal{K}$ chosen uniformly at random **$\Pr(P|C) = \Pr(P)$**



Whether or not C is observed, P is as likely as its occurrence in the plaintext space

- **This implies: $|\mathcal{K}| \geq |\mathcal{C}| \geq |\mathcal{P}|$ for a perfectly secure encryption scheme**

- ▶ $|\mathcal{C}| \geq |\mathcal{P}|$ holds for any encryption scheme as the encryption functions need to be injective
- ▶ If $|\mathcal{K}| < |\mathcal{C}|$ would hold, then for any $P \in \mathcal{P}, \{E_k(P) \mid k \in \mathcal{K}\} \neq \mathcal{C}$, i.e., there is a $C \in \mathcal{C}$ that does not occur as ciphertext of P such that $\Pr(P|C) = 0$ for this C
- ▶ As we assume $\Pr(P) > 0$, this contradicts the perfect forward secrecy

Equivalent Formulations for Perfect secrecy

Definition:

Given a probability distribution \Pr on \mathcal{P} , and $\Pr(P) > 0$ for all plaintexts P

An encryption scheme is said to provide **perfect secrecy** if

For each $P \in \mathcal{P}, C \in \mathcal{C}$ and $K \in \mathcal{K}$ chosen uniformly at random

$$\Pr(P|C) = \Pr(P)$$

Equivalent

1. $\Pr(C|P) = \Pr(C)$
2. $\Pr(C|P_1) = \Pr(C|P_2)$

Proof of 1.:

“ \Leftarrow ”: Assume $\Pr(C|P) = \Pr(C)$, then $\frac{\Pr(C|P)\Pr(P)}{\Pr(C)} = \Pr(P)$

as $\Pr(C|P)\Pr(P) = \Pr(P|C)\Pr(C)$ it follows that $\Pr(P)$

$$= \Pr(P|C)$$

“ \Rightarrow ”: Symmetrical argument

Equivalent Formulations for Perfect secrecy

Definition:

Given a probability distribution \Pr on \mathcal{P} , and $\Pr(P) > 0$ for all plaintexts P

An encryption scheme is said to provide **perfect secrecy** if

For each $P \in \mathcal{P}, C \in \mathcal{C}$ and $K \in \mathcal{K}$ chosen uniformly at random

$$\Pr(P|C) = \Pr(P)$$

Equivalent

1. $\Pr(C|P) = \Pr(C)$
2. $\Pr(C|P_1) = \Pr(C|P_2)$

Proof of 2.:

“ \Rightarrow ”: Follows directly from 1.

If $\Pr(C|P) = \Pr(C)$ for any $P \in \mathcal{P}, C \in \mathcal{C}$

then $\Pr(C|P_1) = \Pr(C|P_2)$ for any $P_1, P_2 \in \mathcal{P}, C \in \mathcal{C}$

Proof of 2.:

“ \Leftarrow ”: If $\Pr(C|P_1) = \Pr(C|P_2) = x$ for any $P_1, P_2 \in \mathcal{P}, C \in \mathcal{C}$, then

$$\Pr(C) = \sum_P \Pr(C|P) \Pr(P) = x \sum_P \Pr(P) = x = \Pr(C|P)$$

Shannon's Theorem 1949

Shannon's Theorem:

Let $|\mathcal{P}| = |\mathcal{C}| = |\mathcal{K}|$, and $\Pr(P) > 0$ for all plaintexts P .

Then an encryption scheme provides **perfect secrecy** \Leftrightarrow

1. **K chosen uniformly at random** for each plaintext to encrypt and
2. **for each $P \in \mathcal{P}$ and $C \in \mathcal{C}$ there is exactly one $K \in \mathcal{K}$ with $E_K(P) = C$**

**A cipher providing perfect secrecy cannot be broken by an attacker.
Not even by one with infinite computational resources and infinite time**

Proof Sketch for Shannon's Theorem

Proof

" \Rightarrow " Assume encryption scheme is perfectly secure

- ▶ Let $P \in \mathcal{P}$ and assume there is a $C \in \mathcal{C}$ such that there is no K with $E_K(P) = C$,
- ▶ then $\Pr(P|C) = 0$ and thus $\Pr(P) \neq \Pr(P|C)$ which contradicts the perfect secrecy.
- ▶ Consequently, there must be at least one K such that $E_K(P) = C$. As there are as many keys as ciphertexts, there must be exactly one such K for each P and C .
- ▶ If K was not chosen uniformly, then given C , there would be some plaintexts that is more likely, than others. This again contradicts the perfect secrecy.

" \Leftarrow " Assume each key is equally likely and for each P , C and there is exactly one K such that $E_K(P) = C$.

- ▶ Then, $\Pr(C|P) = \frac{1}{|\mathcal{K}|}$ such that for any C and P_1, P_2 it holds that $\Pr(C|P_1) = \Pr(C|P_2) = \frac{1}{|\mathcal{K}|}$, such that the second equivalent definition of perfect secrecy holds

The One-Time-Pad (OTP)

- Plaintext space, ciphertext space, key space

 - ▶ $\mathcal{P} = \mathcal{C} = \mathcal{K} = \{0,1\}^n$ for some $n \in \mathbb{N}$,

- Key Generation:

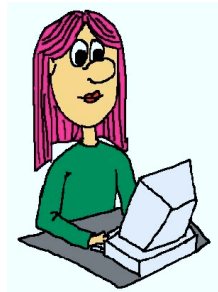
 - ▶ Pick $K \in \mathcal{K}$ **uniformly at random** for each $P \in \mathcal{P}$ to encrypt

- Encryption:

$$C = P \oplus K$$

- Decryption

$$C \oplus K = P \oplus K \oplus K = P$$



$$\begin{array}{r} P = 10111101 \\ \oplus \\ K = 00110010 \\ \parallel \\ C = 10001111 \end{array}$$



$$\begin{array}{r} C = 10001111 \\ \oplus \\ K = 00110010 \\ \parallel \\ P = 10111101 \end{array}$$



Also Known as
Vernam Cipher or
Vernam's one-time-pad

Perfect Secrecy of the One-Time-Pad

Theorem:

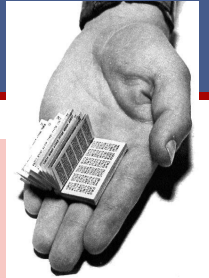
The One-Time-Pad provides perfect secrecy

Proof:

- ▶ Follows directly from Shannon's Theorem:
 - As $|\mathcal{P}| = |\mathcal{C}| = |\mathcal{K}|$ per definition of the OTP, we can apply Shannon's Theorem
 - Key is selected uniformly at random in one-time pad \Rightarrow each key is equally likely
 - Given any pair C, P of ciphertext and plaintext there is a key K that encrypts P to C , namely $K = P \oplus C$:

$$E_K(P) = P \oplus K = P \oplus (P \oplus C) = C$$

Properties of the One-Time-Pad



Advantages

- **Easy to compute**
 - ▶ Encryption and decryption are the same operation
 - ▶ Bitwise XOR is very cheap to compute
- **As secure as theoretically possible**
 - ▶ Given a ciphertext, all plaintexts are equally likely
 - ▶ Security independent on the attacker's computational resources

Disadvantages

- **Key must be as long as plaintext**
 - ▶ Impractical in most realistic scenarios
 - ▶ Still used for diplomatic and intelligence traffic
- **Does not guarantee integrity**
 - ▶ One-time pad only guarantees confidentiality
 - ▶ Attacker cannot recover plaintext, but can easily change it to something else without being detected
- **Insecure if keys are reused**
 - ▶ Attacker can obtain XOR of plaintexts
- **Obviously not practical for all applications**

Overview

• Introduction

- ▶ Intuition
- ▶ Formal definition
- ▶ Historic examples

• Computational Security

- ▶ Attacker models
 - Knowledge
 - Goal
 - Strategy

• Perfect Secrecy

- ▶ Definition
- ▶ Shannon's theorem
- ▶ One-time-pad

• Practical Schemes

- ▶ Stream ciphers
- ▶ Block ciphers
- ▶ Modes of encryption

What is an encryption scheme

Can a cipher be perfectly secure?



How can we model attackers?

How do modern ciphers work and how are they used?

Practical Modern Encryption Schemes

- **Most encryption schemes used in practice do not provide perfect secrecy**
 - ▶ **Stream ciphers** try to simulate the OTP based on a small random seed
 - ▶ **Block cipher** encrypt complete blocks of plaintexts instead of single bits
- **When do we call such encryption schemes secure?**

Computational Security

An encryption scheme is called **computationally secure** if

- ▶ **All known attacks against the cipher are computationally infeasible**
- ▶ I.e., theoretically possible but would take too much time to be practical for any (reasonable) amount of resources

Attacker Models

General assumption in any attack

- ▶ Attacker knows which cipher is used
- ▶ In line with Kerckhoff's principle

Attack result

- ▶ (Partial) key recovery
 - Attacker tries to retrieve (part of) the key
- ▶ (Partial) plaintext recovery
 - Attacker tries to retrieve (part of) the plaintext

Key recovery implies plaintext recovery but not the other way round

Power of attacker



- Strength of attacker increases ↓
- ▶ **Cipher-text-only attack**
 - Attacker knows only ciphertext
 - ▶ **Known-plaintext attack**
 - Knows some pairs of plaintext and ciphertext
 - ▶ **Chosen-plaintext attack** >  **Chapter 4**
 - Can obtain ciphertext for plaintexts of his choice
 - ▶ **Chosen-ciphertext attack** >  **Chapter 4**
 - Can obtain plaintext for ciphertexts of his choice before target ciphertext is known

Illustration of Ciphertext-only Attack

- A classical eavesdropper has access to ciphertext
- Thus, he can collect ciphertext(s) and try to
 - ▶ Recover the key and/or
 - ▶ Recover the plaintext

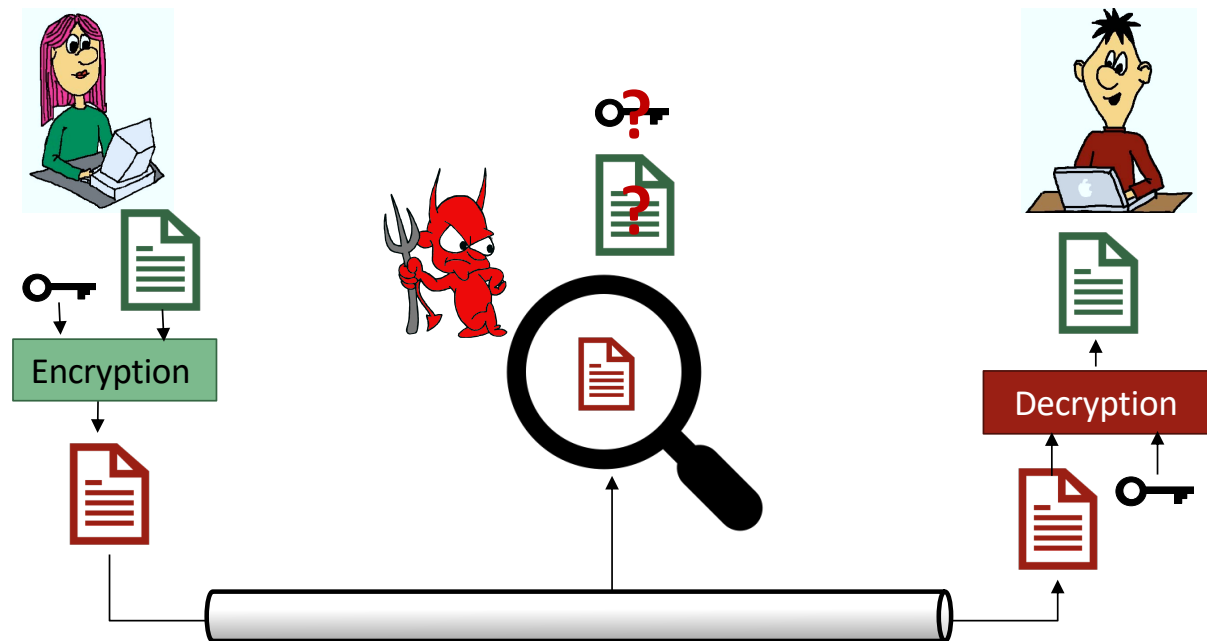
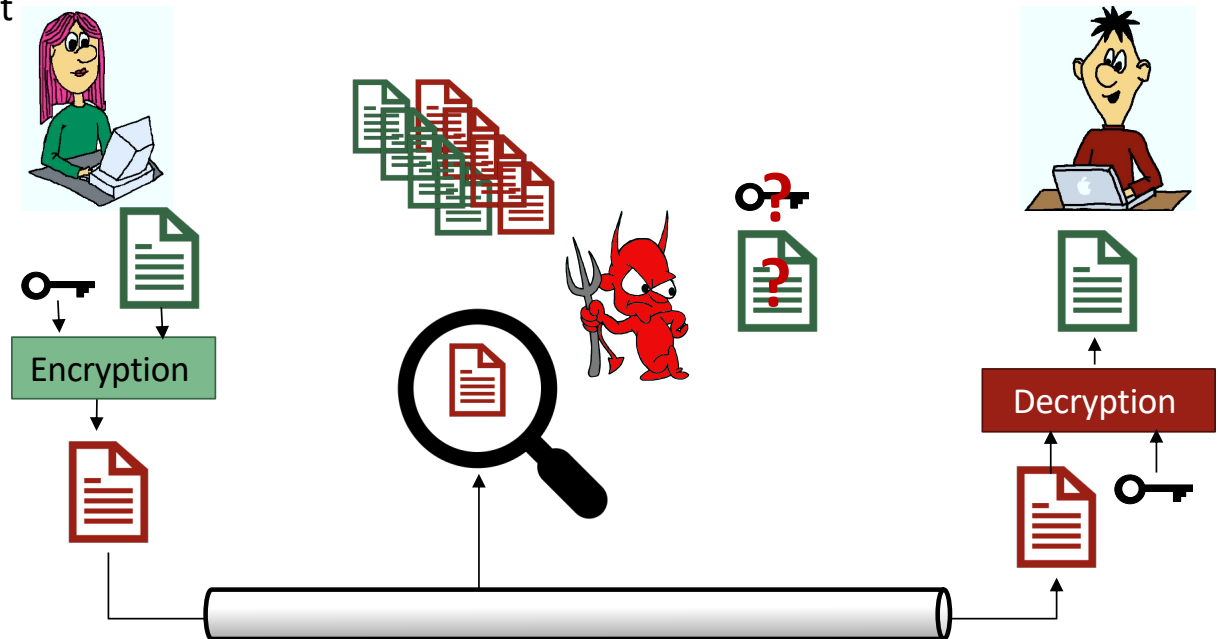


Illustration of Known-Plaintext Attack

- **Attacker observes ciphertext and has access to one or more pair of plaintext and ciphertext**
 - ▶ E.g., as he is able to guess plaintext for some ciphertexts
 - E.g., due to Bob's reaction on receiving the ciphertext
 - ▶ Tries to recover key and/or plaintext

Example:

- ▶ Substitution cipher vulnerable to a known plaintext attack
- ▶ One pair of plaintext / ciphertext sufficient to break (part of) the key



Example: Exhaustive Key Search

- **Try out all possible keys from the key space**
 - ▶ **Ciphertext-only setting**
 - Try out each key to decrypt the ciphertext and check if resulting plaintext **“makes sense”**
 - Only works if valid plaintexts are recognizable for the attacker
 - ▶ **Known-plaintext setting**
 - Try out each key to decrypt the ciphertext
 - Check if it decrypts to the known plaintext
- **Ciphertext-only setting is more difficult for the attacker**
 - ▶ Consequently: being secure against a ciphertext-only attack is easier to achieve
- **Security in a chosen-ciphertext setting is hardest to achieve**

Difficulty of Known-Plaintext Brute Force Attack

- **Difficulty of exhaustive key search is proportional to the key size**

▶ On average attacker will have to try out $\frac{|\mathcal{K}|}{2}$ keys

- **And proportional to the resources of the attacker**

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

Other Attack Strategies besides Brute Force and Frequency Analysis

- **Time-memory trade-off**

- ▶ Can be used to accelerate known-plaintext attacks
- ▶ Exploits a trade-off between time, memory and key space size

- **Differential cryptanalysis**

- ▶ Chosen-plaintext attack
- ▶ Attacker tries to recover key using known differences between plaintexts and comparing them to the differences in the ciphertexts

- **Algebraic attacks**

- ▶ Reduces breaking a cipher to solving a system of linear equations with the key bits as unknowns
- ▶ Can work very well in a known-plaintext setting

- **Related key attacks**

- ▶ Chosen-plaintext attack
- ▶ Assumes attacker has access to chosen plaintext encrypted with keys
- ▶ Attacker knows relations between keys

Overview

• Introduction

- ▶ Intuition
- ▶ Formal definition
- ▶ Historic examples

• Computational Security

- ▶ Attacker models
 - Knowledge
 - Goal
 - Strategy

• Perfect Secrecy

- ▶ Definition
- ▶ Shannon's theorem
- ▶ One-time-pad

• Practical Schemes

- ▶ Stream ciphers
- ▶ Block ciphers
- ▶ Modes of encryption

What is an encryption scheme

Can a cipher be perfectly secure?



How can we model attackers?

How do modern ciphers work and how are they used?

Stream Ciphers

- **The one-time pad $C = P \oplus K$ is perfectly secure**
 - ▶ If the key is chosen uniformly at random for each P
- **Idea of stream cipher**
 - ▶ Replace K with pseudo-random bit-generator PRBG
 - Seed PRBG with "truly random" key K
 - Include a fresh initialization vector IV for each P
 - ▶ Encryption/Decryption very fast
 - Key stream can be pre-generated
- **The PRNG should be cryptographically secure**
 - ▶ We typically **cannot proof** that a PRBG is cryptographically secure, we assume it is if no attack **is known**

Stream cipher

For each plaintext P select a fresh IV and set $C = E_K(P) = IV \parallel P \oplus \text{PRBG}(IV, K)$. PRBG(IV, K) is also referred to as **key stream**. The same key K is used for multiple plaintexts

A PRBG is said to **be cryptographically secure** iff

There is no polynomial-time algorithm which on input of the first k bits of the output of PRBG can predict the next bit with probability $> \frac{1}{2}$. I.e., it **passes the next bit test**.

General Stream Cipher Weakness

- **If the IV is ever reused with the same key**
 - ▶ Stream ciphers are vulnerable to a known-plaintext attack
- **Why?**
 - ▶ Assume attacker known P_1, C_1
 - As $C_1 = E_K(P_1) = IV \parallel P_1 \oplus \text{PRBG}(IV, K)$ attacker knows IV and $\text{PRBG}(IV, K)$
 - Thus, if IV and K are reused to encrypt P_2 , and attacker observes C_2
 - Then he can decrypt P_2 by $C_2 \oplus IV \parallel \text{PRBG}(IV, K) = 0 \parallel P_2$
- **As, e.g., been used to attack the security architecture WPA2 for WLAN**
 - ▶ Known as **KRACK attack**

Examples for Stream Ciphers

• Well-known insecure stream ciphers

- ▶ RC4
 - Before its break used in WLAN, TLS, ...
- ▶ A5/1, A5/2
 - Supported by GSM (2G mobile networks)
- ▶ E0
 - Supported by old Bluetooth versions
- ▶ ...

• Well-known (yet) unbroken stream ciphers

- ▶ SNOW 3G
 - Supported by 3G/LTE/5G networks
- ▶ CHACHA20
 - Supported by TLS, IPsec,...
- ▶ Unbroken Block ciphers in CTR Mode
 - Supported by LTE/5G networks
 - Supported by TLS, IPsec,...
- ▶ ...

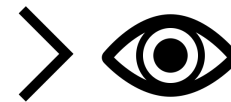
• Any cipher that only provides computational security can break at any point in time

- ▶ We need to be prepared and always ensure that we can easily switch from one cipher to another



Block Ciphers

- **Operate on plaintext blocks of a specific length**
 - ▶ Called the **block length** $b \in \mathbb{N}$ of the cipher
 - ▶ Plaintext space $\mathcal{P} = \{0,1\}^b$ and ciphertext space $\mathcal{C} = \{0,1\}^b$
 - ▶ For each key K in the key space $\mathcal{K} = \{0,1\}^k$, $E_K : \mathcal{P} \rightarrow \mathcal{C}$
- **Typically need to be used in a specific mode of encryption**
 - ▶ Specifies how plaintexts of length $> b$ bits are encrypted



Later in this Chapter

Examples for Block Ciphers

- **Well-known insecure block ciphers**

- ▶ DES
 - Before its break used in IPSec, TLS, ...
- ▶ IDEA
- ▶ ...

- **Well-known (yet) unbroken block ciphers**

- ▶ KASUMI
 - Supported by 3G/LTE/5G networks
- ▶ AES
 - Supported by TLS, IPSec,...
- ▶ Camellia
 - Supported by TLS
- ▶ ...

- **Any cipher can break at any point in time**

- ▶ We need to be prepared and always ensure that we can easily switch from one cipher to another



Example Block Cipher: DES

- **Published in 1977 by the National Bureau of Standards***
 - ▶ Designed by IBM and the NSA
- **Uses a 64-bit key K and a block length of 64 bit**
 - ▶ But: 8 bits of the key are used as parity bits
- **Effective key size is 56 bits**



* called National Institute of Standards and Technology (NIST) since 1988

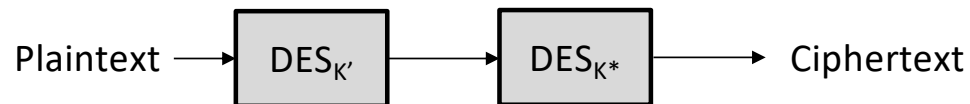
Security of DES

- **January 13th, 1999: DES key broken within 22 hours and 15 minutes**
 - ▶ In a contest sponsored by RSA Labs using
 - ▶ Brute force key search using
 - ▶ the Electronic Frontier Foundation's Deep Crack custom DES cracker ...
 - ▶ ... and the idle CPU time of around 100,000 computers
- **Since then, DES is considered insecure**
- **Biggest weakness still is the key length of 56 bits only!**

First Proposed Fix: 2DES

- **First idea to increase the key size of DES**

- ▶ Use DES twice with two independently chosen keys



- **Problem: this does not double the key size!**

Complexity of the attack:

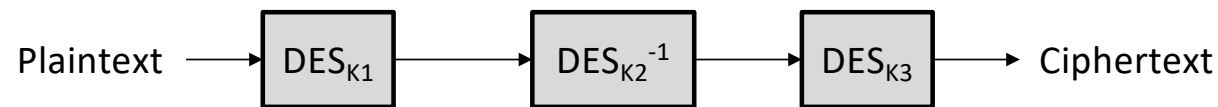
- ▶ $2 \cdot 2^{56} = 2^{57}$ DES operations
- ▶ Effective key size only increased by one!

Meet in the middle attack on 2DES

- ▶ Assume attacker has access to (P, C) , where $C = \text{DES}_{K^*}(\text{DES}_{K'}(P))$
- ▶ Attacker can encrypt P with any possible key (2^{56} DES operations)
 - And thus, create lookup table $E_K(P) = Z_K$ for $K \in \{0,1\}^{56}$ of intermediate ciphertext
- ▶ Attacker can decrypt C with all possible keys (at most 2^{56} DES operations)
 - And compute $D_K(C) = X_K$, $K \in \{0,1\}^{56}$ until $X_{K_i} = Z_{K_j}$ is found in the lookup table
- ▶ Then $K_j = K'$ and $K_i = K^*$ with high probability

3DES = "Triple DES"

- Use DES three times in a row



- Variants

- ▶ 3-key DES: K1, K2, and K3 are pairwise different
 - Provides an effective key size of 112 bit according to NIST
- ▶ 2-key DES: K1 = K3
 - Provides an effective key size of 80 bit according to NIST
- ▶ Both variants use encryption with K1, decryption with K2 and encryption with K3
 - Setting K1=K2=K3 this allows 3DES-only capable senders to communicate with DES-only capable receivers

The Advanced Encryption Standard (AES)

• Goals of the NIST Call for AES

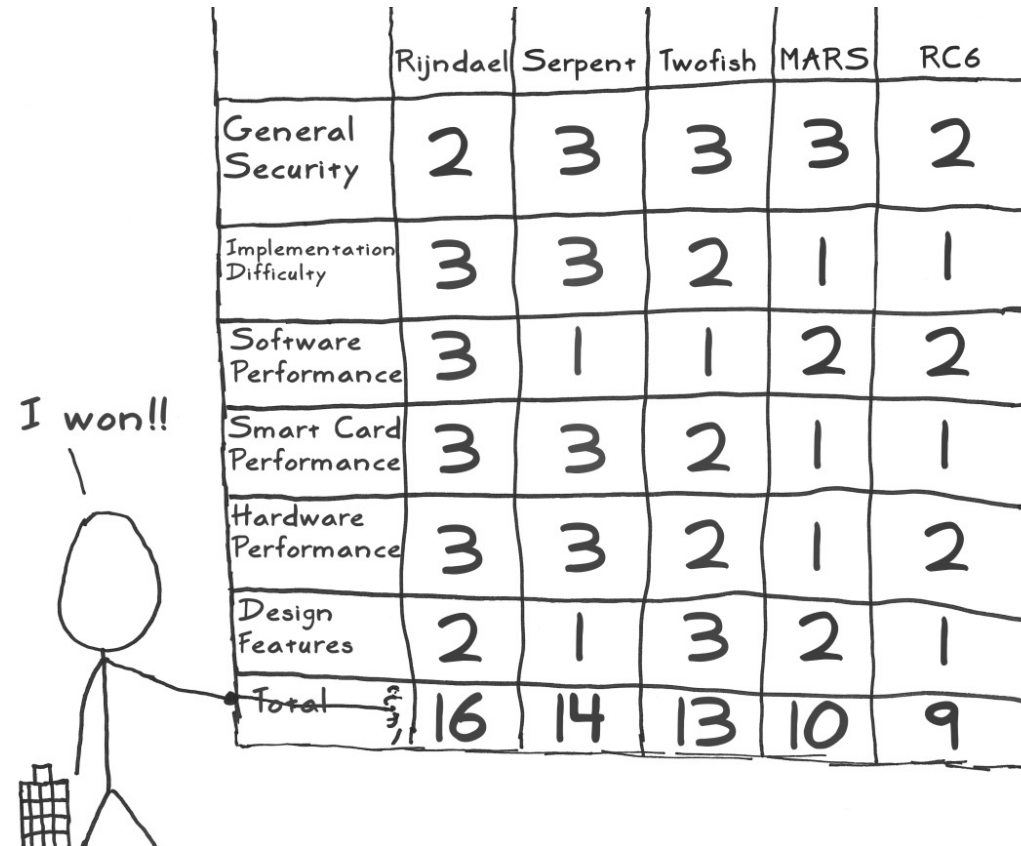
- ▶ More secure than 3DES
- ▶ More efficient than 3DES
- ▶ Support different key lengths
 - 128, 192, and 256 bit
- ▶ The block length of the cipher is 128 bit
 - Regardless of the key length



• Timeline of AES Selection

- ▶ Jan. 1997 NIST-call published
- ▶ Aug. 1998: 15 candidates presented
 - Cast-256, Crypton, DEAL, DFC, E2, Frog, HPC, Loki97, Magenta, MARS, RC6, Rijndael, SAFER+, Serpent, Twofish
 - Broken shortly afterwards (or during presentation)
 - DEAL, Frog, HPC, Loki97, Magenta
- ▶ Aug. 1999 finalists announced
 - MARS, RC6, Rijndael, Serpent, Twofish
- ▶ Oct. 2000 **Rijndael selected as AES**
- ▶ Nov. 2001 AES standardized in FIPS 197

Selection Criteria



A hand-drawn table comparing encryption algorithms across seven criteria. A stick figure on the left points to the 'Total' row, exclaiming 'I won!!'.

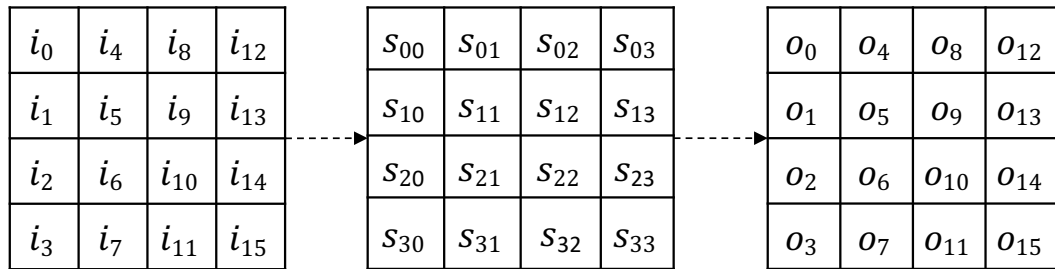
	Rijndael	Serpent	Twofish	MARS	RC6
General Security	2	3	3	3	2
Implementation Difficulty	3	3	2	1	1
Software Performance	3	1	1	2	2
Smart Card Performance	3	3	2	1	1
Hardware Performance	3	3	2	1	2
Design Features	2	1	3	2	1
Total	16	14	13	10	9

Taken from <http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html>

Structure of AES

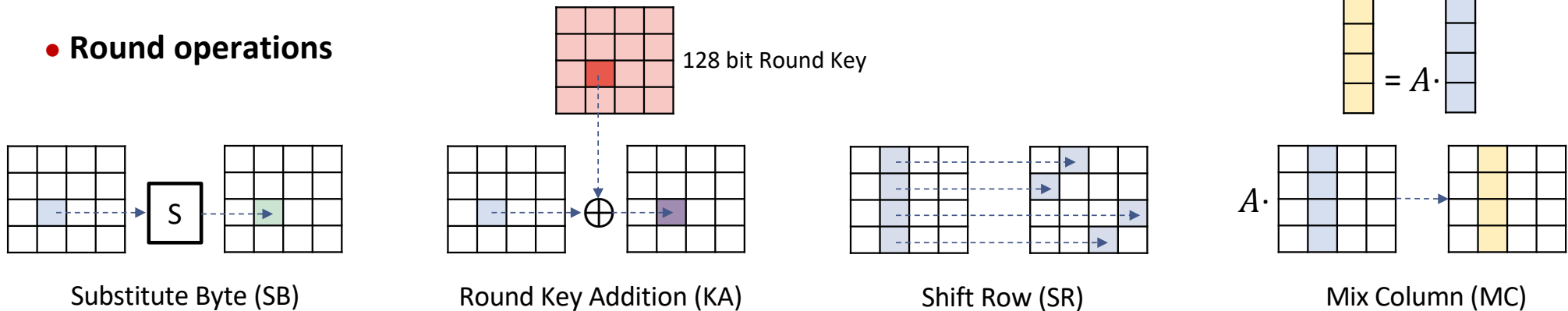
- AES operates in rounds

▶ Input and output of each round represented as 4x4 byte matrices



$$A = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

- Round operations



Reminder: Multiplication in $GF(2^8)$ with $x^8 + x^4 + x^3 + x + 1$ as irreducible Polynomial

- For example, (in hex notation) $57 \bullet 83 = c1$ in $GF(2^8)$ because

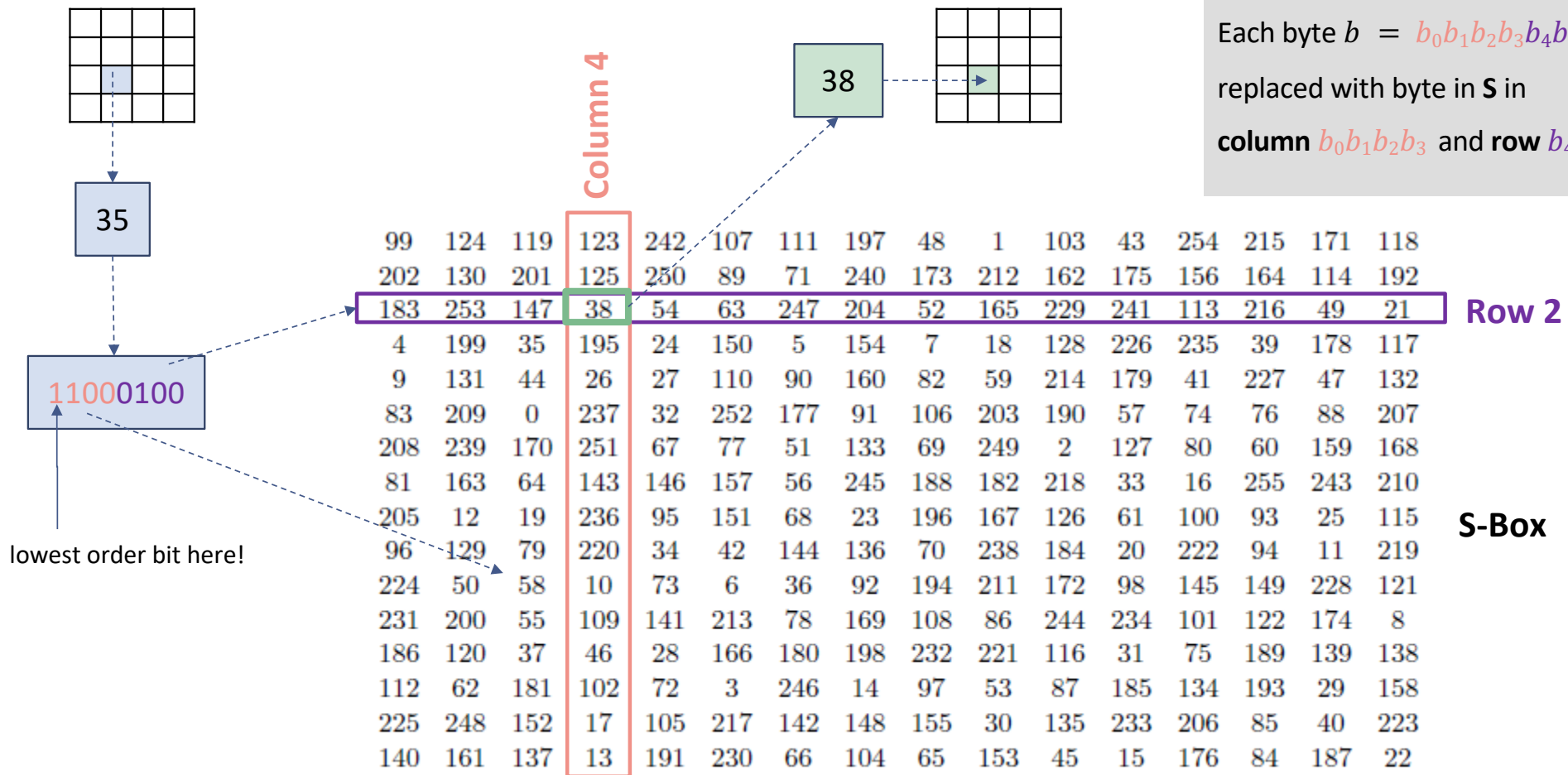
▶ $57 = 01010111 \simeq x^6 + x^4 + x^2 + x + 1$

▶ $83 = 10000011 \simeq x^7 + x + 1$

▶ $(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1 = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

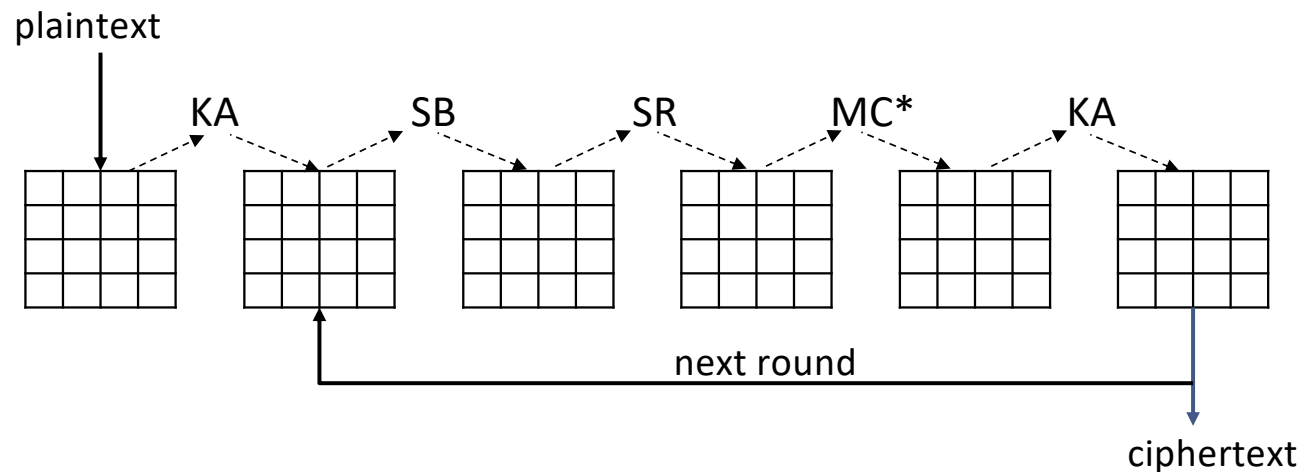
▶ $x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$ modulo $x^8 + x^4 + x^3 + x + 1 = x^7 + x^6 + 1$
 $\simeq 1100\ 0001$
 $= c1$

Substitute Byte (SB)



Each byte $b = b_0b_1b_2b_3b_4b_5b_6b_7$ is replaced with byte in S in column $b_0b_1b_2b_3$ and row $b_4b_5b_6b_7$

AES Operation Overall



- **The round key is always 128 bit key**

- ▶ Different for each round, generated from the secret key

MC*: no mix column operation in the last round

- **Number of rounds depends on the key size**

- ▶ 128 bit key: 10 rounds 192 bit key: 12 rounds 256 bit key: 14 rounds

Modes of Encryption

- **Block ciphers of block length b**

- ▶ Allow us to encrypt a plaintext P of b bit
- ▶ How can we encrypt longer plaintexts?

- **Mode of encryption**


- ▶ Let $P = P_1 \parallel P_2 \parallel P_3 \parallel P_4 \parallel \dots \parallel P_n$
with $P_i \in \{0, 1\}^b$ for $i = 1, \dots, n - 1$
and $P_n \in \{0, 1\}^l$ for some $0 < l \leq b$
- ▶ A mode of encryption specifies how to encrypt plaintext P based on a b bit block cipher $E_K(\cdot)$

- **Modes we cover here**

- ▶ Electronic Code Book (ECB) mode
- ▶ Cipher Block Chaining (CBC) mode
- ▶ Counter Mode (CTR)

- **Modes we may cover in exercises**

- ▶ Cipher Feedback Mode (CFB)
- ▶ Output Feedback Mode (OFB)

- **AEAD Modes**  **Chapter 3**

- ▶ Authenticated Encryption with Associated Data (AEAD) Modes
 - E.g., Galois Counter Mode (GCM)

Electronic Codebook Mode (ECB)

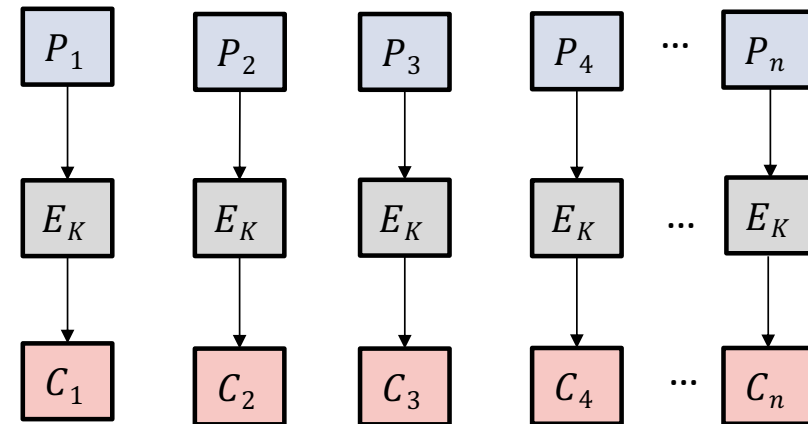
ECB Mode

Encryption: $C_i = E_k(P_i)$ for $i = 1, \dots, n$

Decryption: $P_i = D_k(C_i)$ for $i = 1, \dots, n$

Requires **padding** of P_n to b bit

Illustration of encryption in ECB Mode



• Problem

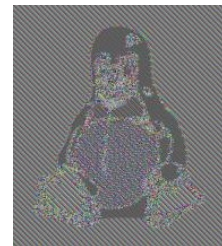
- ▶ Same P_i leads to same C_i
- ▶ Thus, patterns in plaintext lead to patterns in ciphertext
- ▶ **ECB mode should not be used!**



Plaintext



ECB-encrypted



Cipher Block Chaining Mode (CBC)

CBC Mode

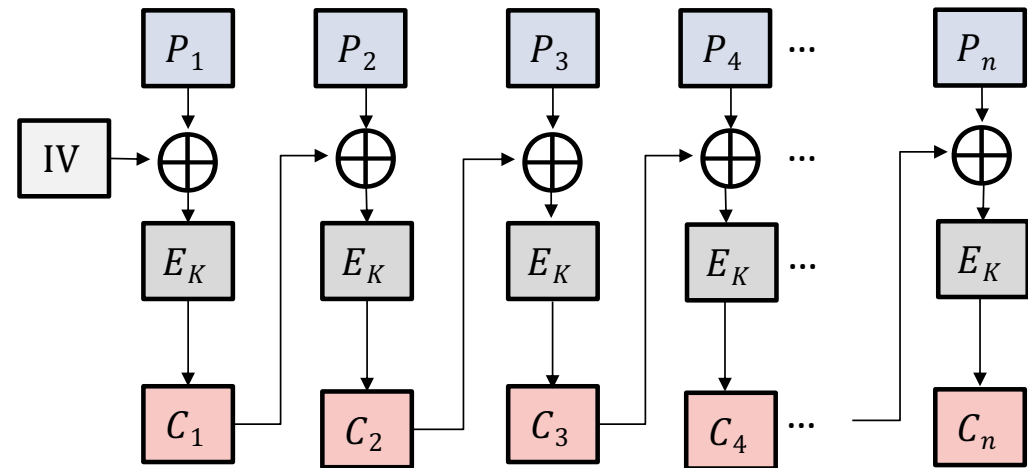
IV := C_0

Encryption: $C_i = E_K(P_i \oplus C_{i-1})$ for $i = 1, \dots, n$

Decryption: $P_i = D_K(C_i) \oplus C_{i-1}$ for $i = 1, \dots, n$

Requires **padding** of P_n to b bit

Illustration of encryption in CBC Mode



- Requires a fresh IV for each plaintext to encrypt



- ▶ If same IV is reused on P and P^*
 - then C_1 and C_1^* reveal, whether $P_1 = P_1^*$

- ▶ Is **vulnerable** to a so-called **padding-oracle attack** > Should not be used anymore

Counter Mode (CTR)

CTR Mode

IV public, fresh for each plaintext

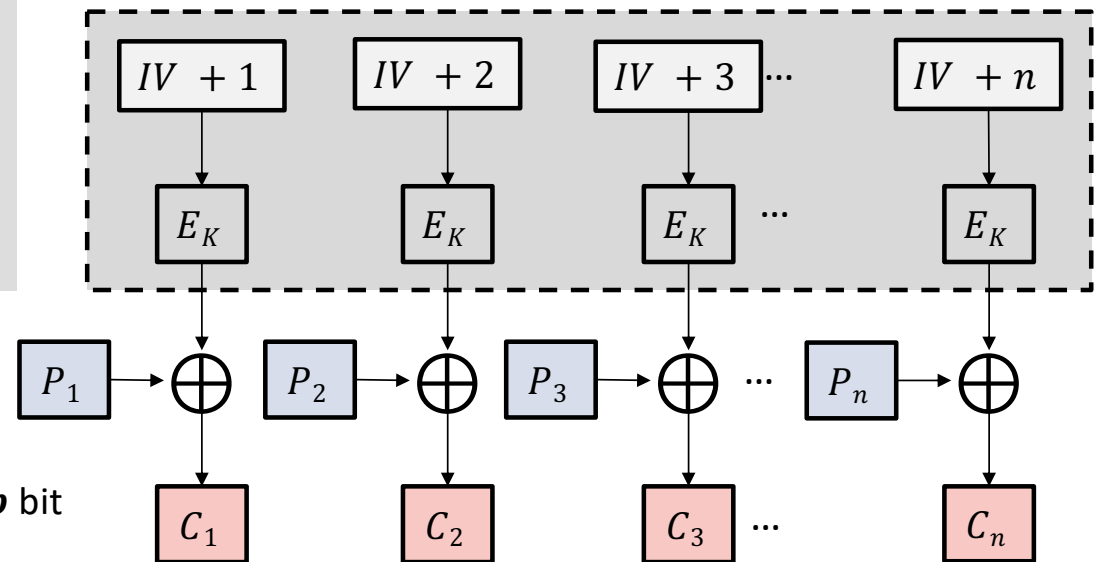
Encryption: $C_i = E_k(IV + i) \oplus P_i$ for $i = 1, \dots, n$

Decryption: $P_i = C_i \oplus E_k(IV + i)$ for $i = 1, \dots, n$

Properties of CTR Mode

- ▶ CTR Mode **does not require padding** of P_n to b bit
- ▶ Ciphertext is of the same size as plaintext
- ▶ CTR Modes **turns a block cipher** into a **stream cipher**
- ▶ CTR mode encryption and decryption can be parallelized

Illustration of encryption in CTR Mode

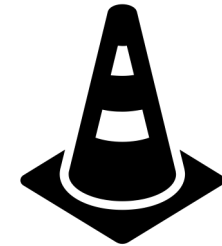


Summary

- **Symmetric Encryption Schemes provide confidentiality**
 - ▶ Require a secret key shared between the communicating entities
- **Perfect secrecy can be obtained by the one-time-pad**
 - ▶ Requires key chosen uniformly at random and as long as the plaintext for each plaintext
 - ▶ Impractical to use in many situations
- **Practical encryption schemes only provide computational security**
 - ▶ **Can in theory always be broken** with a brute force attack in a known plaintext setting
 - Require long keys to make brute force **attack practically impossible**
- **Different attacker models make different assumptions with respect to**
 - ▶ The knowledge of the attacker (ciphertext-only, known plaintext,...)
 - ▶ The goal of the attacker (plaintext recovery, key recovery)
 - ▶ The approach the attacker takes (brute force, frequency analysis, differential analysis...)

Summary

- **Practical symmetric encryption schemes can be divided into**
 - ▶ Stream ciphers, e.g., ChaCha20
 - ▶ Block ciphers, e.g., AES
- **Stream ciphers encrypt a plaintext by xoring it with a key stream**
 - ▶ Key stream is generated by
 - a (longer term) secret key that is reused for multiple plaintext
 - and fresh IV for each plaintext to encrypt
 - ▶ Should never reuse IVs with the same key
- **Block ciphers require the use of a mode of encryption**
 - ▶ Specifies how to encrypt plaintext that are longer than one block-length of the block cipher
 - ▶ These modes have a strong influence of the security of the encryption scheme
 - Used with in an insecure mode, a secure block cipher may become insecure
 - ▶ The effective key size of a block cipher cannot be doubled by applying the cipher twice



References

- **More details on symmetric encryption**

- ▶ Johannes Buchmann, Einführung in die Kryptographie, 6. Auflage, Springer Verlag 2016
 - Kapitel 3 - Kapitel 6
- ▶ W. Stallings, Cryptography and Network Security: Principles and Practice, 8th edition, Pearson 2022
 - Chapters 3, 4, 6, and 7

- **Standard Documents**

- ▶ FIPS 197: Advanced Encryption Standard
 - <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf>
- ▶ FIPS 46-3: Data Encryption Standards (DES)
 - <https://csrc.nist.gov/files/pubs/fips/46-3/final/docs/fips46-3.pdf>



IT-Security

Chapter 3: Symmetric Integrity Protection

Prof. Dr.-Ing. Ulrike Meyer



Overview

- **Definition and security of integrity protection**

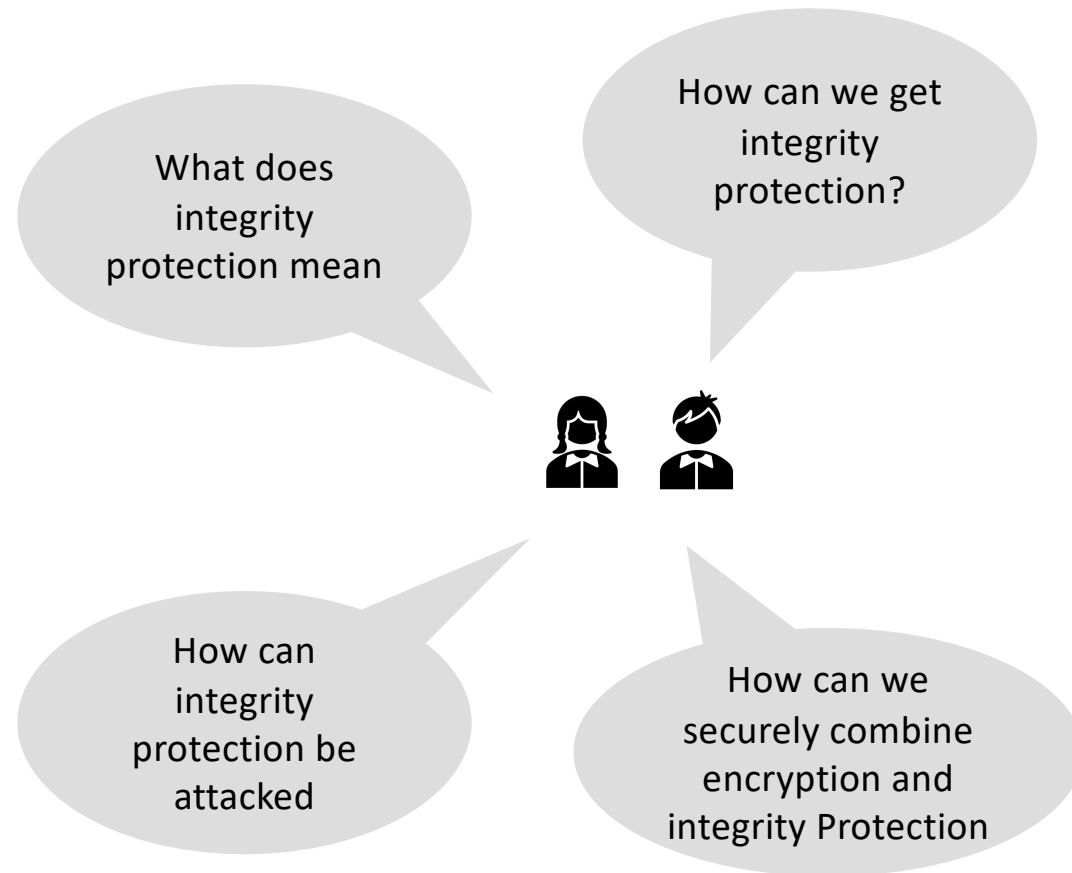
- ▶ Intuition
- ▶ More formal definition

- **Message Authentication Codes**

- ▶ Based on cryptographic hash functions
- ▶ Based on symmetric ciphers

- **Combining Encryption and Integrity Protection**

- Based on cryptographic hash functions
 - ▶ Based on symmetric ciphers



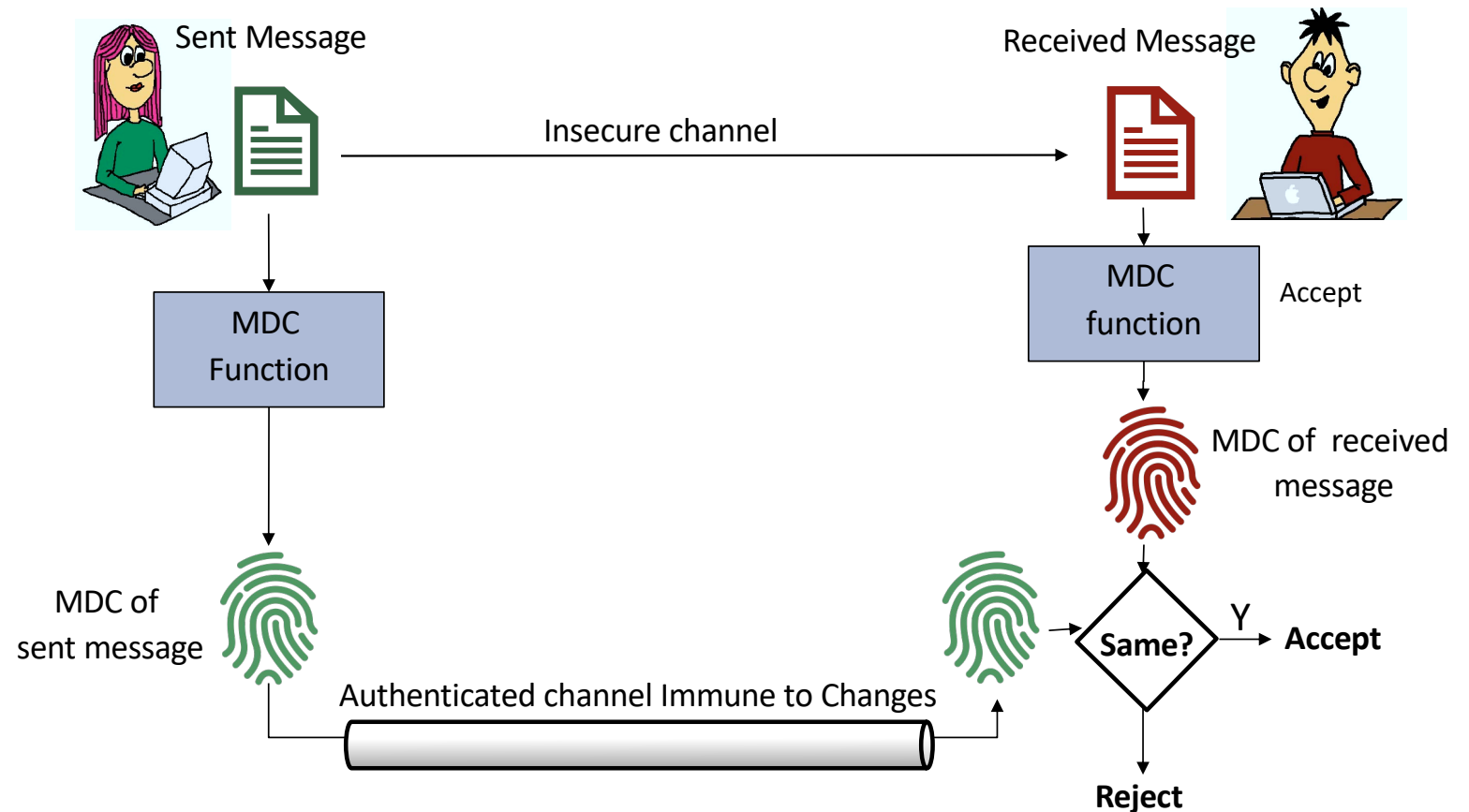
Intuition for Data Integrity protection

- **Manipulation of messages sent over an insecure network cannot be prevented**
 - ▶ Anyone between the communicating entities can change the message
 - Flip bits, delete bits, replace messages with other ones
- **Encryption schemes do typically NOT enable detection of such manipulations**
 - ▶ See the many examples in the exercises
- **Data integrity protection mechanisms aim at **detecting any message manipulation** by unauthorized entities**
 - ▶ Can be realized in form of Modification Detection Codes (MDCs)
 - ▶ Can be realized in form of Message Authentication Codes (MACs)

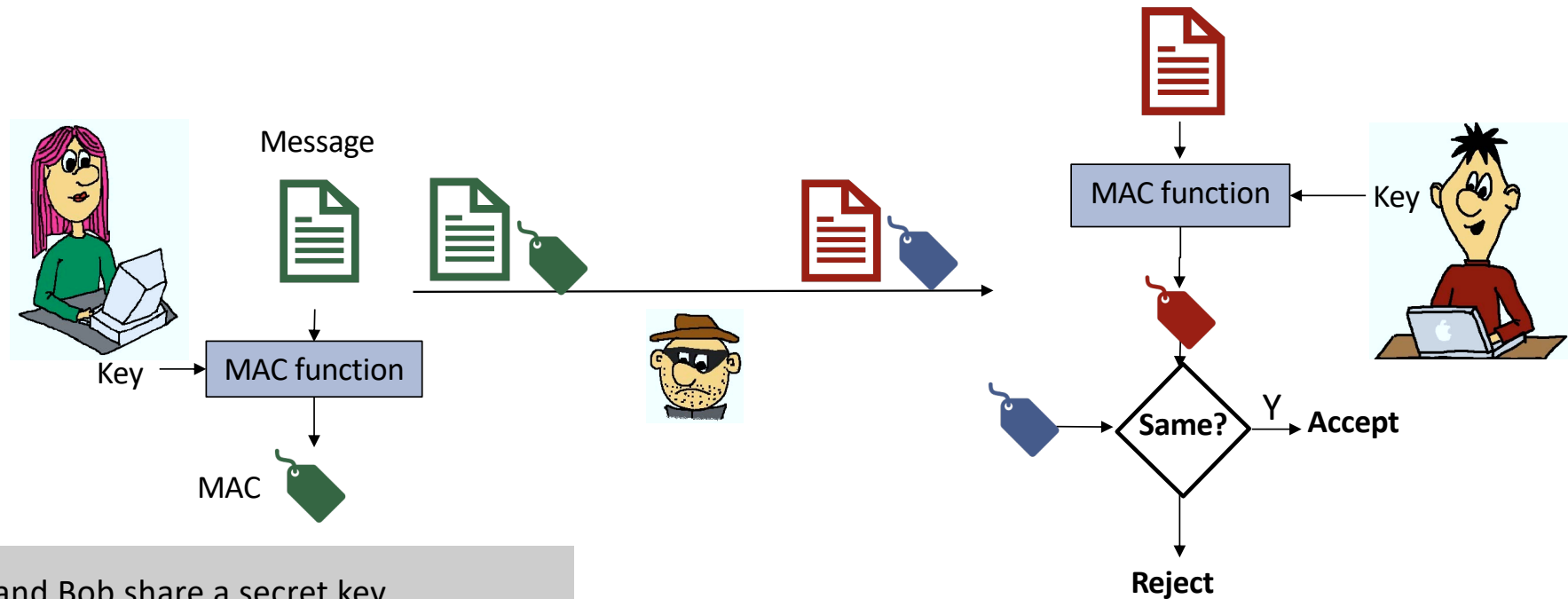
Idea of Modification Detection Codes

Problem with MDCs

- ▶ Require a second channel immune to change
- ▶ Bob needs to be sure that the MDC he receives really comes from Alice



Idea of Message Authentication Codes



- Alice and Bob share a secret key
- Alice computes MAC of message using key
- Alice sends message and MAC to Bob
- Attacker may change message and/or MAC

- Bob computes MAC of received message using key
- Compares computed MAC to received MAC
- Decides that message was received as sent if both are the same

Hash Function

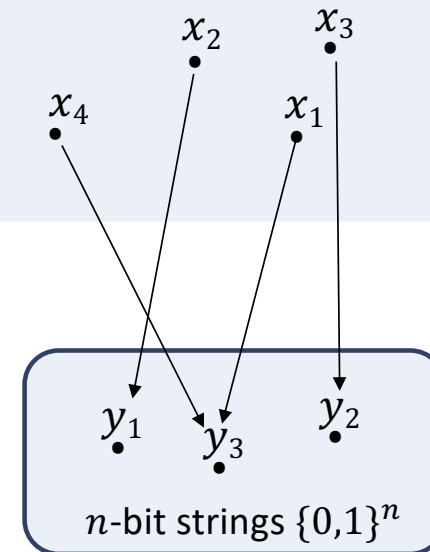
- A **hash function** is a function h with the properties

- ▶ **compression:** h maps an input x of arbitrary bit-length to an output $h(x)$ of fixed bit-length n
- ▶ **ease of computation:** given h and x , $h(x)$ is easy to compute
 - there is a polynomial-time algorithm to compute $h(x)$

- A **collision** of a hash function is a

- ▶ pair of inputs x_1, x_2 , with $h(x_1) = h(x_2)$

bit strings of any length: $\{0,1\}^*$



Any hash function h has collisions!



Minimal Number of Collisions of a hash function

- **Basic pigeonhole principle**

- ▶ If n pigeonholes are occupied by $n + 1$ pigeons
then at least one pigeonhole is occupied with more than one pigeons

- **Generalization**

- ▶ If n pigeonholes are occupied by $k \cdot n + 1$ pigeons
then at least one pigeonhole is occupied with more than k pigeons



- **Consequence for the minimal number of collisions**

- ▶ If a hash function maps $k \cdot n$ messages to n hash values
then there is at least one hash value to which k or more messages hash
 - E.g., if $n = 16$, and $k \cdot n = 64$, then there are 4 or more messages that hash to the same value

Cryptographic Hash Function

- A hash function is **preimage** resistant
 - ▶ if given a randomly chosen $y = h(x)$ but not x it is computationally infeasible to find any pre-image x' with $h(x') = y$
- A hash function is **second preimage** resistant
 - ▶ if given $x, h(x)$ it is computationally infeasible to find a second pre-image $x' \neq x$ with $h(x') = h(x)$
- A hash function is **collision** resistant
 - ▶ if it is computationally infeasible to find a pair x, x' with $x' \neq x$ and $h(x') = h(x)$

Computationally infeasible
here means theoretically computable but impractical (except with negligible probability) as it takes too many resources and too much time to compute!

A **cryptographic hash function** is a preimage resistant and collision resistant hash function

Relations between the Properties

- Collision resistance \Rightarrow 2nd pre-image resistance

\Rightarrow A **cryptographic hash function** is always 2nd pre-image resistant as it is collision resistant

- 2nd pre-image resistance $\not\Rightarrow$ collision resistance
- Collision resistance $\not\Rightarrow$ pre-image resistance
- Pre-image resistance $\not\Rightarrow$ collision resistance
- 2nd pre-image resistance $\not\Rightarrow$ pre-image resistance
- Pre-image resistance $\not\Rightarrow$ 2nd pre-image resistance



Note that some of these implications do hold for a narrower definition of a hash function mapping long fixed length-messages to much shorter hashes

Example Proof of the Relations

- Collision resistance \Rightarrow 2nd pre-image resistance

Proof by contradiction

- ▶ Assume h is collision resistant but not 2nd pre-image resistant, then given $x, h(x)$ we can find an x' such that $h(x') = h(x)$.
- ▶ Thus, we have found the collision (x, x')
- ▶ This contradicts our assumption which thus cannot hold

Example Proof of the Relations

Collision resistance $\not\Rightarrow$ pre-image resistance

Constructive proof

- ▶ Assume g is collision resistant n -bit hash function
- ▶ Define $h(x) = \begin{cases} 1 \parallel x & \text{if the bitlength of } x \text{ is } n \\ 0 \parallel g(x) & \text{otherwise} \end{cases}$
- ▶ Then $h(x)$ is a $(n + 1)$ -bit hash function that is collision resistant but not pre-image resistant



Note that $a \parallel b$ stands for the concatenation of two bit-strings a and b



A similar proof can be used to prove that 2nd-pre-image resistance does not imply pre-image resistance

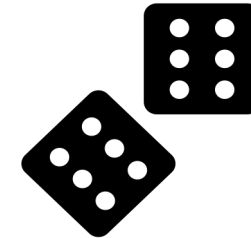
Related Terms and Synonyms

- **Cryptographic hash function = Secure hash function**
 - ▶ pre-image resistant + collision resistant
 - ▶ thereby also second-preimage resistant
- **One way hash function**
 - ▶ pre-image resistant
- **Second preimage resistant = weak collision resistant**
 - ▶ as it is implied by collision resistant
- **Collision resistant = strong collision resistant**
- **Output of hash function = hash value = message digest = hash**

Ideal Hash Function through Random Oracle Model

- **An ideal n -bit hash function h would operate as follows**

- ▶ Upon receipt of a message m it has not seen before
 - Pick an n -bit value uniformly at random from $\{0,1\}^n$ and return it as $h(m)$
- ▶ Upon receipt of a message m it has seen before
 - Return the same value $h(m)$, that was picked when the message was new



- **This ideal hash function is as pre-image and collision resistant as possible**

- **We can thus use it to determine an upper bound on**

- ▶ how pre-image resistant a real-world hash function can be
- ▶ how collision resistant a real-world hash function can be

Complexity of Attacks against Ideal Hash Function

Pre-image attack: Given a hash value y

- Randomly select x and compute $h(x)$
- Compare $h(x)$ to y
 - ▶ Stop if $h(x) = y$
 - ▶ Return to Step 1 otherwise
- **Requires $0.69 \cdot 2^n = O(2^n)$ hash computations to find a pre-image with probability $\frac{1}{2}$**

Collision attack:

- Randomly select x and compute $h(x)$, store result
- Compare each newly computed hash with the values already stored
 - ▶ Stop if $h(x) = h(x')$ and output (x, x')
 - ▶ Return to Step 1 otherwise
- **Requires $1.18 \cdot 2^{n/2} = O(2^{n/2})$ hash computations to find a collision with probability $\frac{1}{2}$**

- Both statements on the complexities can be proven by the solution to flavors of the so-called Birthday Problem

Example Proof of Complexity of Pre-image Attack

The 1st birthday problem

- ▶ Given N different balls in a jar and one fixed ball \hat{x}
- ▶ How many times do we need to pull from the jar independently and uniformly at random with put back until with probability P we pulled \hat{x} at least once?





Solution

- ▶ If we chose one ball x , then the probability that $x \neq \hat{x}$ is $1 - \frac{1}{N}$
- ▶ The probability that we are unsuccessful k -times in a row is $(1 - \frac{1}{N})^k$
- ▶ The probability P that we picked \hat{x} at least once if we pick k -times is thus

$$P = 1 - (1 - \frac{1}{N})^k \sim 1 - e^{-\frac{k}{N}} \quad (\text{using the approximation } 1 - x \sim e^{-x} \text{ (} x \ll 1 \text{)})$$

- ▶ Thus $k \sim \ln[1/(1 - P)] N$ and in particular for $P = \frac{1}{2}$ we get $k \sim 0.69 \cdot N$

1st birthday problem

 Given **253** students, the **probability** that at least one of them has **February 2nd** as its birthday is **1/2** 
 

Similar but Omitted: Proof of Complexity of Collision Attack




Birthday Paradoxon

- ▶ Given N different balls in a jar
- ▶ How many times do we need to pull independently and uniformly at random with put back from the jar until with probability P we drew the same ball \hat{x} twice?

Solution

- ▶ We need to draw $k \sim \sqrt{2 \ln[1/(1-P)] N}$ times and in particular for $P = \frac{1}{2}$ we get $k \sim 1.18 \cdot \sqrt{N} = 1.18 \cdot N^{\frac{1}{2}}$

Birthday paradox

 Given **23** students, the  **probability** that at least two of share the same birthday is **1/2** 

Examples for Hash Functions and their Properties

Algorithm	Maximum Message Size in Bit	Block Size in Bit	Rounds	Size of Hash Value	Year
MD5	2^{64}	512	64	128	1991
SHA-1	2^{64}	512	80	160	1993
SHA-2-224	2^{64}	512	64	224	2002
SHA-2-256	2^{64}	512	64	256	
SHA-2-384	2^{128}	1024	80	384	
SHA-2-512	2^{128}	1024	80	512	
SHA-3-256	unlimited	1088	24	256	2015
SHA-3-512	unlimited	576	24	512	

- MD5 and SHA-1 are not considered collision resistant anymore and should no longer be used
- SHA-2 not broken yet, but break needs to be feared

Example Time-Lines of Breaks of MD5 and SHA-1

MD5

- 1993: Collision found by Boer and Bosselaers
- 1996: Attack that found a collision in a modified version of MD5
- 2004: Wang et al. found collisions in MD5 and others
- 2005: Further make collision finding feasible on a laptop (8 hours to find a collision)
- 2006: Black et al. implemented a toolkit for collisions in MD5
- 2007: Stevens et al. find collisions in less than 10 seconds on a on a 2.6Ghz Pentium 4
- 2009: MD5 attacks successfully used to fake certificates
- March 2011 IETF recommendation: MD5 should not be used any more where collision resistance is needed

SHA-1

- 2004: 2nd preimage attack on SHA-1 in 2^{106}
- 2005: Attack found by Wang et al. that finds a collision with 2^{69} hash operations
- 2013: Attack by Stevens et al. finds identical prefix collision in 2^{61} and chosen prefix collision in $2^{77.1}$
- 2015: Attack by Stevens et al. that finds a Free-Start Collision on 76-step SHA-1 in 2^{50} hash operations
- 2017: Collision on SHA-1 found
- 2016/2017 SHA1 was phased out starting from 2016/17 by all major browsers
- SHA-1 is not used anymore in the context of certificates

Overview

- **Definition and security of integrity protection**

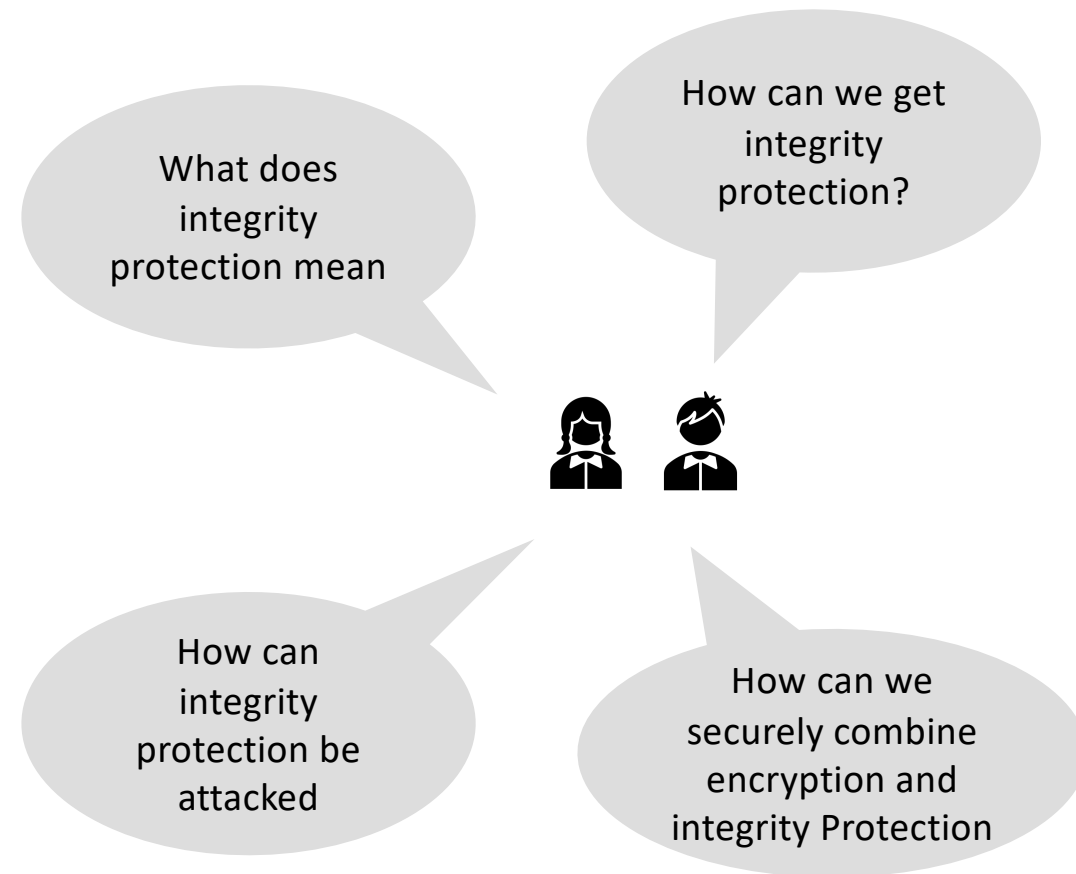
- ▶ Intuition
- ▶ More formal definition

- **Message Authentication Codes**

- ▶ Based on cryptographic hash functions
- ▶ Based on symmetric ciphers

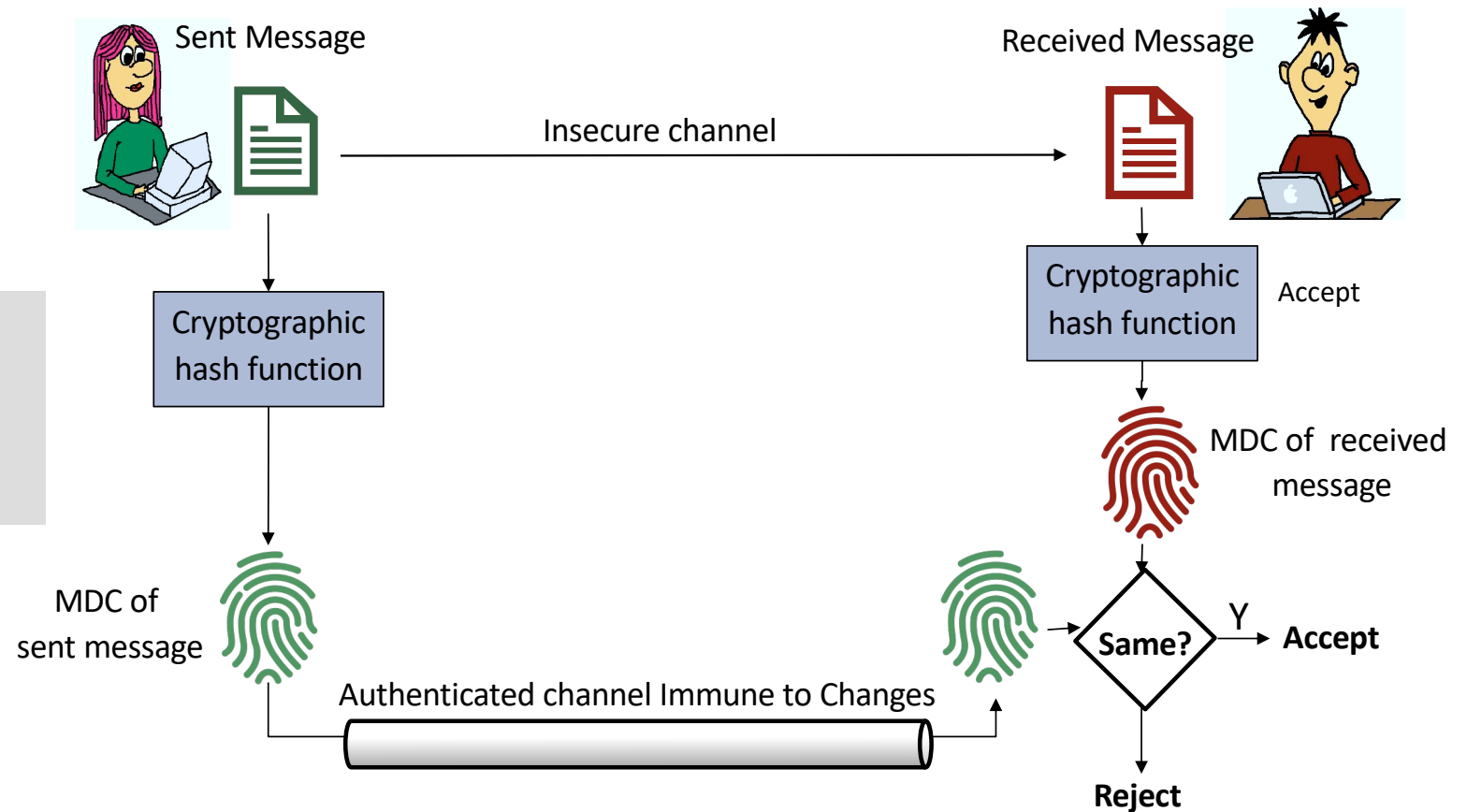
- **Combining Encryption and Integrity Protection**

- Based on cryptographic hash functions
 - ▶ Based on symmetric ciphers

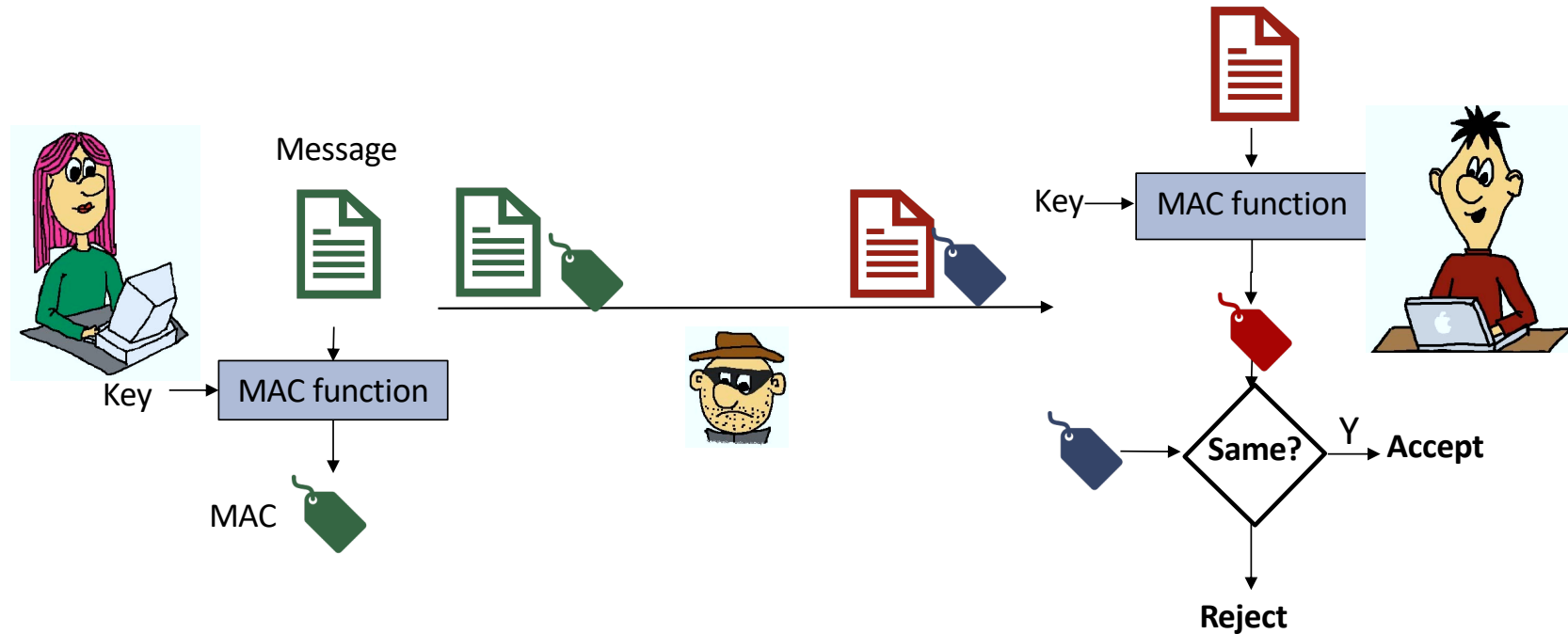


Modification Detection Codes

Modification Detection Codes
can be implemented by
cryptographic hash functions



Message Authentication Codes



- MACs require a secret key as additional input
- MAC functions can be constructed from **cryptographic hash functions** or **block ciphers**

Definition of a Message Authentication Code

- A Message Authentication Code (MAC) is a family of functions MAC_K parameterized by a secret key K with the following properties
 - ▶ **Ease of computation** – given K and x , $MAC_K(x)$ is easy to compute
 - ▶ **Compression** – MAC_K maps an input x of arbitrary finite bit-length to an output $MAC_K(x)$ of fixed bit-length n
 - ▶ **Computation resistance** – for every K and any given number of pairs $(x_i, MAC_K(x_i))$ it is without knowledge of K computationally infeasible to compute any pair $(x, MAC_K(x))$ with x different from all x_i
 - Note that such pairs $(x_i, MAC_K(x_i))$ can typically be obtained by an attacker by eavesdropping
- MACs can be constructed from **cryptographic hash functions** or **block ciphers**

HMAC: Bellare, Canetti, and Krawczyk 1996

- Let h be a cryptographic hash function, then for a message M and key K

$$\text{HMAC}_K(M) = h(K \oplus \text{opad} \parallel h(K \oplus \text{ipad} \parallel M))$$

where **opad** and **ipad** are constant values.

▶ $\text{ipad} = 0x36\dots0x36$

▶ $\text{opad} = 0x5C\dots0x5C$

- **HMAC is computation resistant if h is cryptographic hash function**
 - ▶ HMAC construction does not introduce any new risk
 - ▶ **ipad** and **opad** guarantee that different keys are used in the inner and outer hash computation
 - The two keys will differ in half of the bits because of the choice of **ipad** and **opad**

Can't we just use $h(K \parallel M)$ as MAC?

- **Unfortunately, no!** Simple constructions like that are typically **insecure**
- Many hash functions (e.g., MD2, SHA-1, SHA-2) operate on blocks of M
 - ▶ $M = M_0 \parallel M_1 \parallel \dots \parallel M_n$
 - ▶ h operates on the first block M_0 which is then used as first state to operate on M_1, \dots
 - ▶ Thus, $h(M)$ is the initial state of $h(M \parallel X)$
 - I.e., from known hashes of shorter messages, we can construct hashes of longer messages
 - ▶ I.e., knowing $h(K \parallel M)$ we can compute $h(K \parallel M \parallel X)$ without knowing the key

CMAC: Constructing a MAC from a Block Cipher

- CMAC uses a block cipher E_K of block length $b = 64$ or $b = 128$

- A message M is split into n blocks of length b :

$$M = M_1 \parallel M_2 \parallel \dots \parallel M_n$$

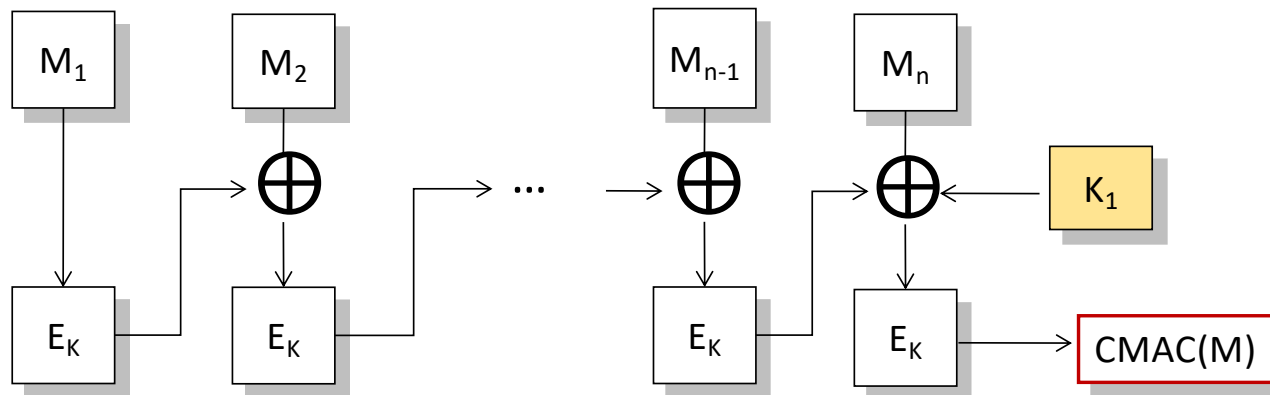
- If the last block M_n is not of length b it is padded with $10 \dots 0$ until it is b bit long

- CMAC computation is equivalent to

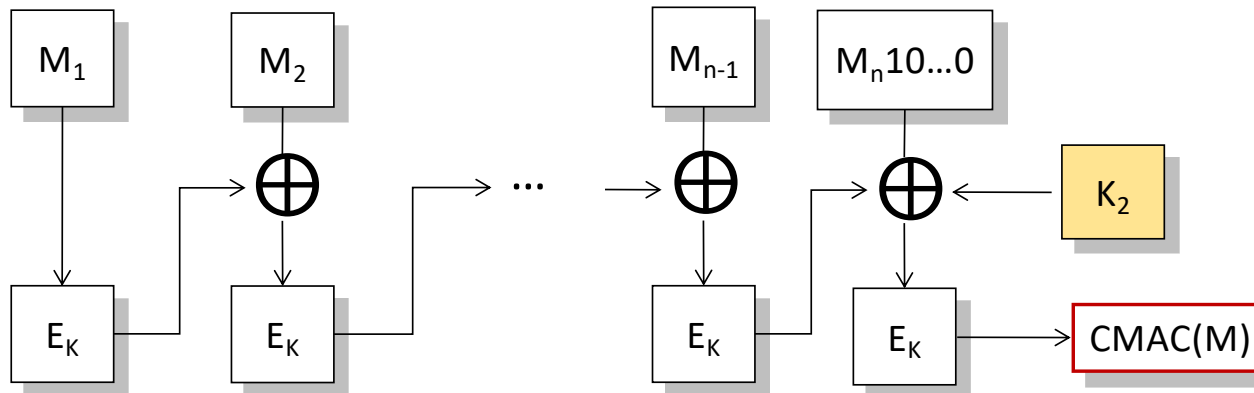
- ▶ Applying CBC Mode of encryption to the message with an IV of all zeros
- ▶ **Except** that the last block is additionally masked with
 - A sub-key K_1 if M_n is of bit length b and with
 - A sub-key K_2 if M_n was padded to be of full bit length b
- ▶ The resulting last ciphertext block is the CMAC of the message

Illustration of the CMAC Computation

If M_n has block length b



If M_n is padded to b bits



K1 and K2 are derived from K

- ▶ $L = E_K(0^b)$, where 0^b is the bitstrings of b zeros
- ▶ $R_{128} = 0^{120}10000111$
- ▶ $R_{64} = 0^{59}11011$

• Then K_1 is computed by

- ▶ If $MSB_1(L) = 0$, $K_1 = L \ll 1$
- ▶ Else $K_1 = L \oplus R_b$

• K_2 is computed by

- ▶ If $MSB_1(K_1) = 0$, $K_2 = K_1 \ll 1$
- ▶ Else $K_2 = (K_1 \ll 1) \oplus R_b$

Rational for the Two Different Keys

- **Let's assume we have a one block message $M = 011$**
 - ▶ then $\text{CMAC}_K(M) = E_K(01110 \dots 0 \oplus K_2)$
- **The one block message $M' = 01110 \dots 0$ has $\text{CMAC}_K(M') = E_K(01110 \dots 0 \oplus K_1)$**
- **So, if K_1 and K_2 were the same,**
 - ▶ then $\text{CMAC}_K(M)$ would be the same as $\text{CMAC}_K(M')$
 - ▶ Thus, an attacker could replace M with M' without the receiver noticing it

Why Do we need the Masking with K_1 and K_2

- Using a “pure” **CBC-MAC is insecure!**

- ▶ I.e., without the masking by K_1 or K_2 in the last step

- **A CBC-MAC allows for forgery in some specific settings**

- ▶ For example, let M and P be two one-block messages and MAC_K be a CBC-MAC

- $MAC_K(M) = E_K(M)$

- $MAC_K(P) = E_K(P)$

- ▶ If an attacker observes $M, MAC_K(M)$ and $P, MAC_K(P)$

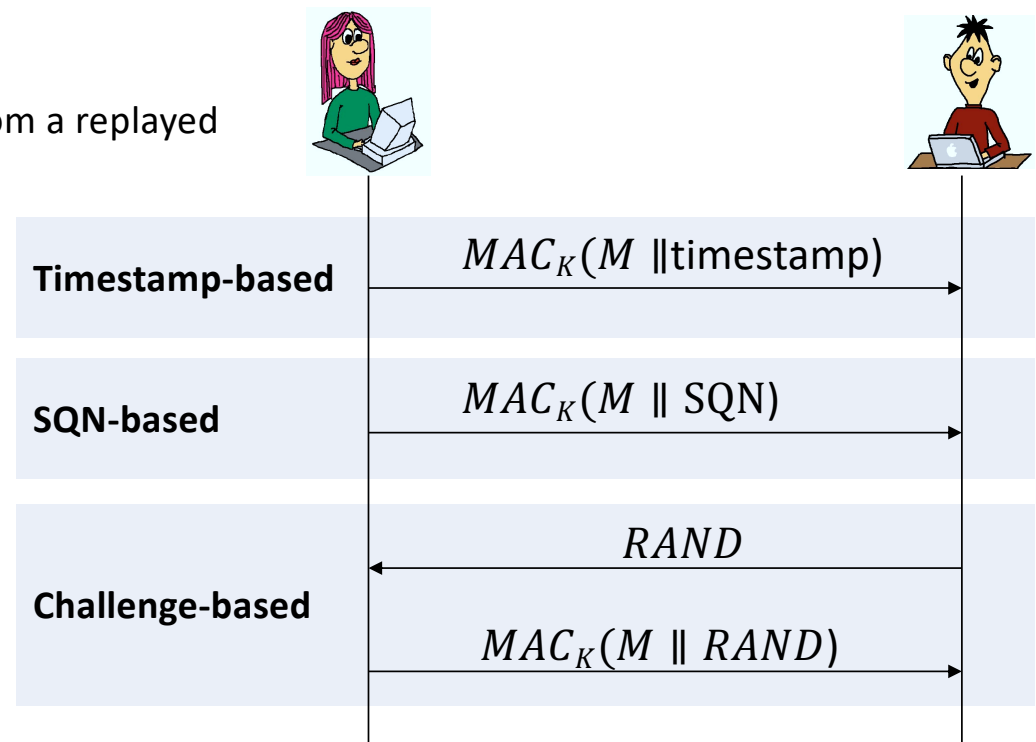
- he can forge a valid CBC-MAC on $M \parallel (P \oplus MAC_K(M))$ without knowing K because:

- $MAC_K(M \parallel (P \oplus MAC_K(M))) = E_K(E_K(M) \oplus P \oplus MAC_K(M)) = E_K(P \oplus MAC_K(M) \oplus MAC_K(M)) = E_K(P) = **MAC_K(P)**$

- **The masking with K_1 and K_2 solves this problem**

Replay Protection

- **A MAC computed over a message alone**
 - ▶ does not protect against replay of the protected message
- **Replay protection requires additional input**
 - ▶ Make a message sent twice distinguishable from a replayed message
- **Additional input**
 - ▶ Counters
 - Time stamps
 - Sequence numbers (*SN*)
 - ▶ Random numbers as challenges (*RAND*)



Replay Protection

	Advantage	Disadvantage	Main Use
Timestamps	No explicit initial value needs to be known by sender and receiver	Require time synchronization between sender and receiver	Whenever sender and receiver are time-synchronized anyway
SNs	Simple, no time-synchronization required	Requires (re-)synchronization of SN, Agreement on initial value, Window of acceptable SNs if in-order delivery of messages cannot be guaranteed	Protect all traffic between two entities once keys are established
RAND	Does not need synchronization, requires random number generator	Requires receiver to challenge the sender and thus adds communication overhead	Mainly used as part of authentication and key agreement protocols, where single messages need to be protected against replay

Overview

- **Definition and security of integrity protection**

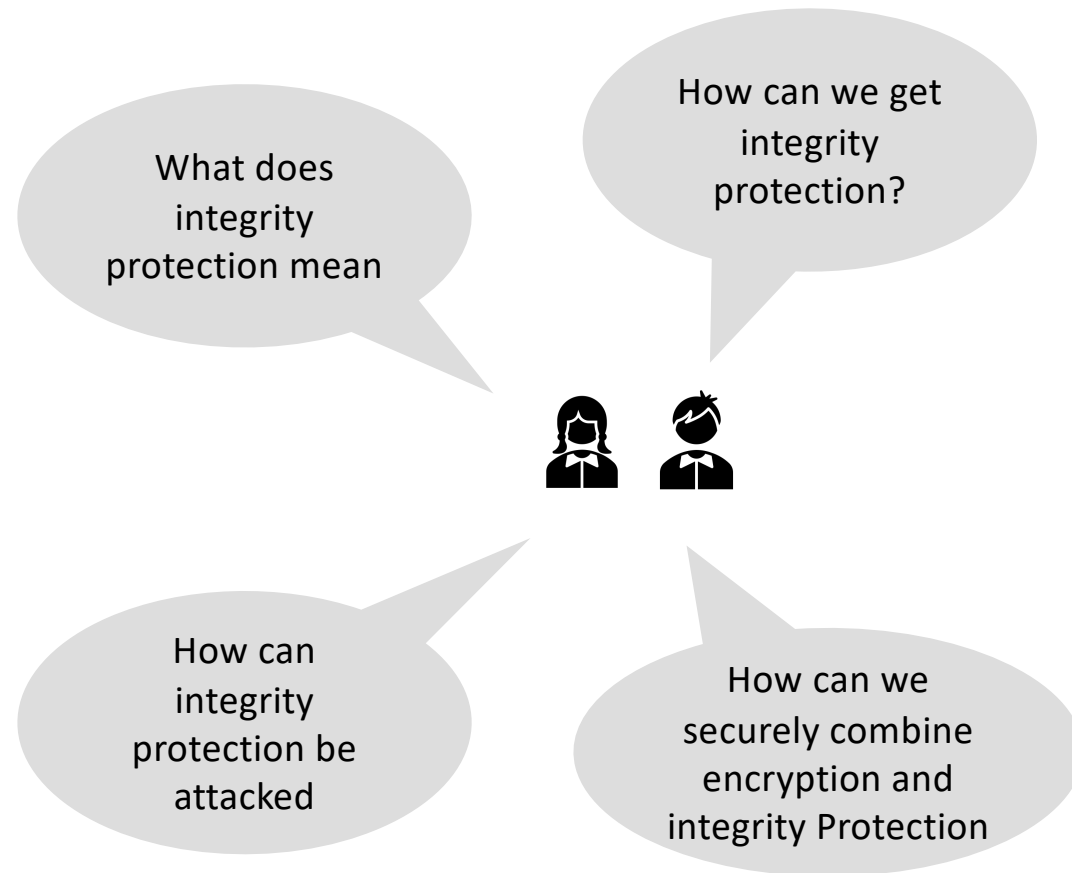
- ▶ Intuition
- ▶ More formal definition

- **Message Authentication Codes**

- ▶ Based on cryptographic hash functions
- ▶ Based on symmetric ciphers

- **Combining Encryption and Integrity Protection**

- Based on cryptographic hash functions
 - ▶ Based on symmetric ciphers



Combining Integrity Protection and Encryption

Encrypt, then MAC: $E_{K_1}(M) \parallel MAC_{K_2}(E_{K_1}(M))$

- Encrypt plaintext with K_1
- Compute MAC on encrypted plaintext with K_2

MAC, then Encrypt: $E_{K_1}(M \parallel MAC_{K_2}(M))$

- Encrypt plaintext with K_1
- Compute MAC on encrypted plaintext with K_2
- MAC can only be checked AFTER decryption

Encrypt and MAC: $E_{K_1}(M) \parallel MAC_{K_2}(M)$

- Encrypt plaintext with K_1
- Compute MAC on plaintext with K_2
- MAC may reveal information on M
- MAC can only be checked AFTER decryption

Special authenticated modes of encryption

- E.g., Galois Counter Mode (GCM)
- E.g., Counter mode with CBC MAC (CCM)
- Typically take an **encrypt then MAC** approach

Example: Galois Counter Mode of Encryption (GCM)

- **Mode of encryption that also provides integrity protection**
 - ▶ Authenticated Encryption with Associated Data (AEAD) Mode
 - Allows for additional data to be integrity protected but not encrypted
- **Based on a block cipher with 128-bit blocklength**
- **GCM can be used as MAC alone**
 - ▶ called **GMAC** then
- **Properties**
 - ▶ Can use IVs of arbitrary length
 - ▶ Easy to implement very efficiently in hardware
 - ▶ Very good software performance

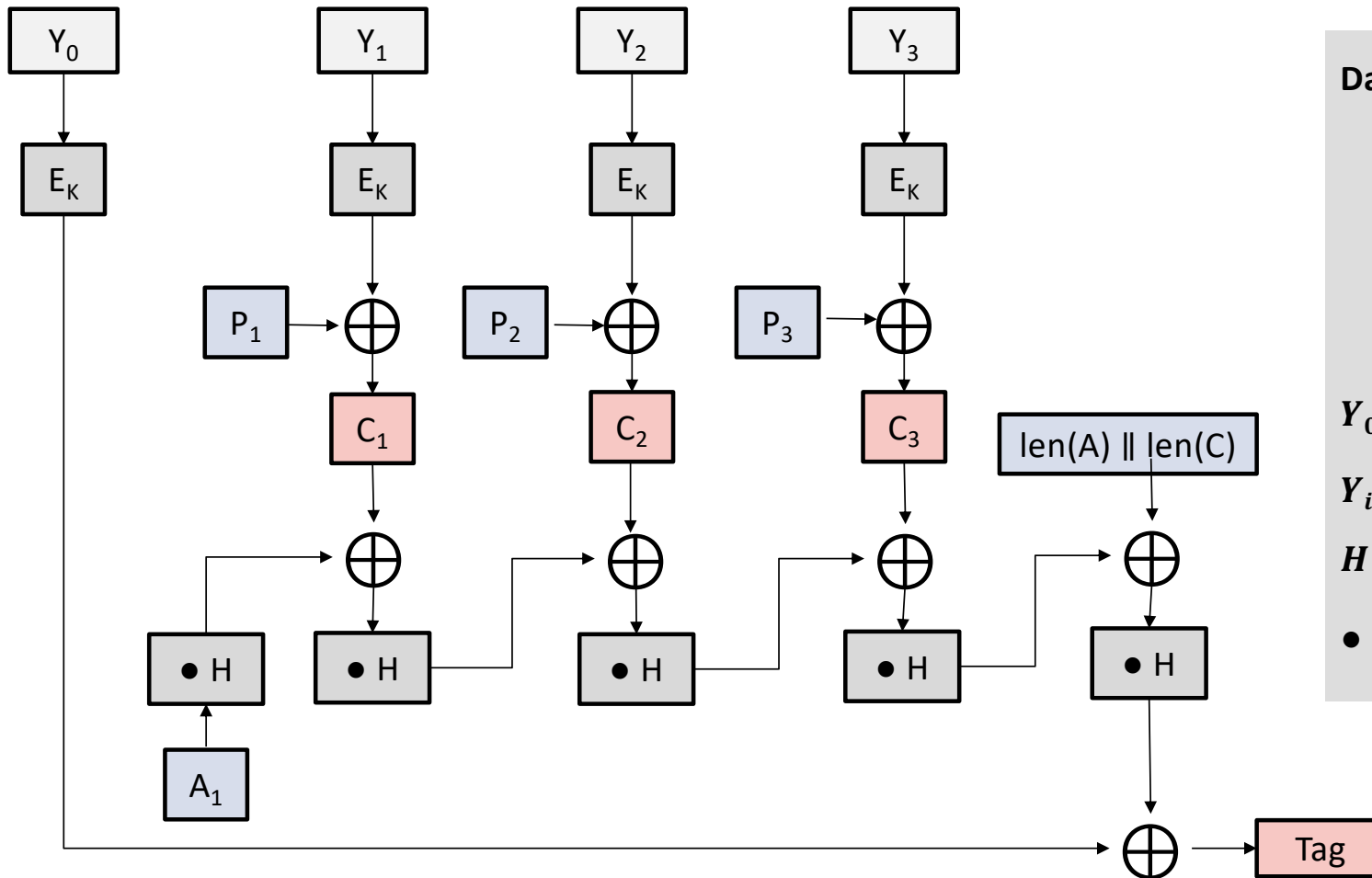
Data blocks to protect

$A_1 \parallel \dots \parallel A_m \parallel P_1 \parallel \dots \parallel P_n$

A_i ($i = 1, \dots, m$) are to be integrity protected only

P_i ($i = 1, \dots, n$) are to be integrity protected and encrypted

Illustration of GCM Encryption and Integrity Protection Operation



Data blocks to protect

$A_1 \parallel P_1 \parallel P_2 \parallel P_3$

A_1 integrity protected

$P_i (i = 1, \dots, 3)$ integrity protected and encrypted

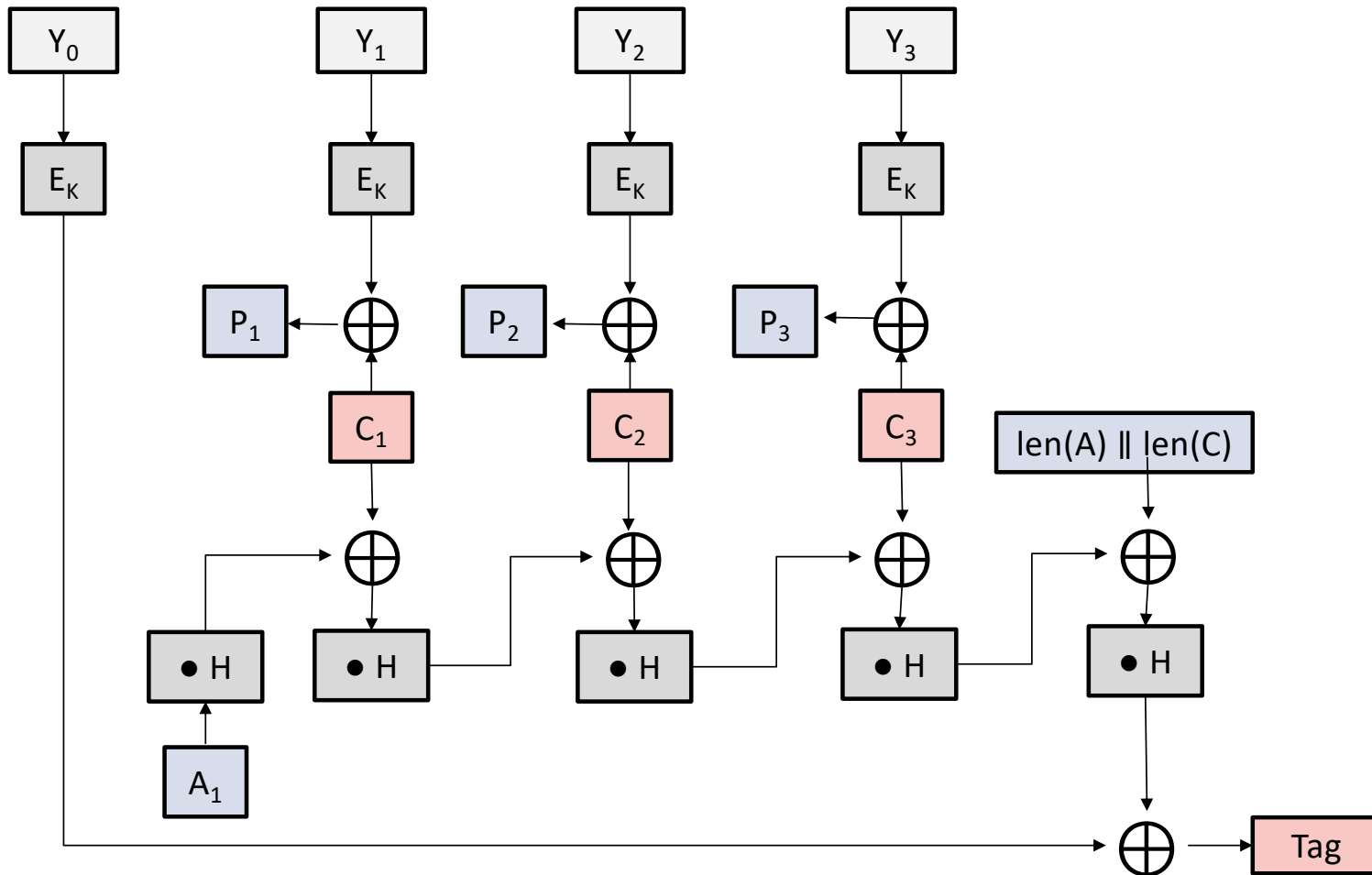
Y_0 Initial counter value

$Y_i = Y_{i-1} + 1$

$H = E_K(0^{128})$

\bullet = Multiplication in $GF(2^{128})$

Illustration of GCM Decryption and Integrity Verification Operation



GCM in Formulars

Data to be protected

$$M = A_1 \parallel \dots \parallel A_m \parallel P_1 \parallel \dots \parallel P_n$$

Initialization:

Y_0 Initial counter value

$$Y_i = Y_{i-1} + 1$$

$$H = E_K(0^{128}) \text{ where } 0^{128} = \underbrace{0 \dots 0}_{128}$$

\bullet = Multiplication in $\text{GF}(2^{128})$

Encryption: $C_i = E_K(Y_i) \oplus P_i$ for $(i = 1, \dots, n)$

Integrity Protection: $T_0 = 0$

$$T_i = (T_{i-1} \oplus A_i) \bullet H \text{ for } i = 1, \dots, m$$

$$T_{m+i} = (T_{m+i-1} \oplus C_i) \bullet H \text{ for } i = 1, \dots, n$$

$$T_{m+n+1} = (T_{m+n} \oplus (\text{len}(A) \parallel \text{len}(C))) \bullet H$$

$$GMAC_K(M) = T_{m+n+1} \oplus E_K(Y_0)$$



Note: if P_n is not of full block length, then C_n is not of full block length

If A_m or C_n are not of full block length, they are padded with zeros in the $GMAC$ computation

Reminder: Multiplication in $\text{GF}(2^{128})$

- $\text{GF}(2^{128})$ is the finite field with 2^{128} elements
 - ▶ It is unique up to isomorphism
- GCM uses the irreducible polynomial $f(x) = 1 + x + x^2 + x^7 + x^{128}$
- Identify each 128-bit string $a = a_0 \dots a_{127}$ with the polynomial $a(x) = \sum_{i=0}^{127} a_i x^i$
- **Multiplication** of a and b in $\text{GF}(2^{128})$ is then defined as
 - ▶ bit string representation of $a(x) \cdot b(x) \bmod f$:

$$\left(\sum_{i=0}^{127} a_i x^i\right) \cdot \left(\sum_{i=0}^{127} b_i x^i\right) \bmod f$$

Summary

- **Message Authentication Codes** provide integrity protection

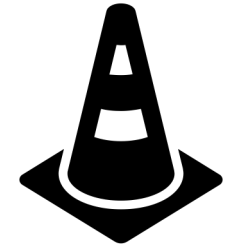
- ▶ MACs can be constructed from cryptographic hash functions: HMAC
- ▶ MACs can be constructed from block ciphers: CMAC
- ▶ Simple constructions like $h(M \parallel K)$ or CBC-MAC are insecure

- **Cryptographic hash functions**

- ▶ Are pre-image resistant and collision resistant
- ▶ Finding a pre-image with probability $\frac{1}{2}$ requires at most $O(2^n)$ hash computations for an ideal hash function
- ▶ Finding a second pre-image with prob. $\frac{1}{2}$ requires at most $O(2^n)$ hash computations
- ▶ Finding a collision with prob. $\frac{1}{2}$ requires at most at most $O(2^{n/2})$ hash computations

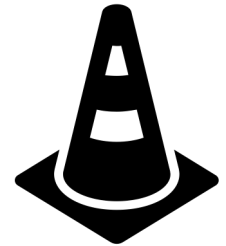
- **Replay protection** requires additional input to an integrity protection mechanism

- ▶ E.g., a counter, a time stamp, or a random number selected by the receiver



Summary

- Securely **combining** encryption and integrity protection
 - ▶ Requires an encrypt-then-MAC type of an approach
 - Special modes of encryption which also provide integrity protection use this as well
 - ▶ Other approaches are insecure or unnecessarily expensive
- The **GCM Mode** of encryption is an example for an AEAD cipher
 - ▶ Provides encryption and integrity protection
 - ▶ Makes use of CTR mode for encryption
 - ▶ Can additionally protect the integrity of data which is not encrypted



References

- **Johannes Buchmann, Einführung in die Kryptographie, Springer Verlag 2016**
 - ▶ Chapter 11 on Hash Functions and Message Authentication Codes
- **W. Stallings, Cryptography and Network Security: Principles and Practice, 8th edition, Pearson 2022**
 - Chapters 12: Message Authentication Codes
- **Specifications**
 - ▶ HMAC: NIST Specification FIPS 198-1
 - <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.pdf>
 - ▶ CMAC:
 - ▶ GCM and GMAC NIST Special Publication 800-38D
 - <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf>



IT-Security

Chapter 4: Asymmetric Cryptography

Public key encryption, Digital Signatures, Diffie-Hellman Key Agreement

Prof. Dr.-Ing. Ulrike Meyer



Overall Lecture Context

- **In the security mechanisms we covered so far**
 - ▶ Alice and Bob needed to share the same secret key
- **In this chapter we learn how asymmetric cryptosystems work**
 - ▶ Alice can share a single public key with multiple other parties and keeps a private key to herself
 - ▶ In an asymmetric encryption scheme,
 - anyone in possession of Alice's public key can encrypt messages for Alice
 - but only Alice can (with the private key) decrypt messages
 - ▶ In a digital signature scheme
 - only Alice can sign a messages
 - anyone in possession of the public key can verify a signature on a message

Overview

- **Basic Number Theory**

- ▶ Finite Fields, greatest common divisor, Fermat's theorem
- ▶ Factorization
- ▶ Discrete Logarithms

- **Digital Signatures**

- ▶ Intuition on integrity protection with digital signatures
- ▶ RSA as signature scheme
- ▶ Digital signature standard

- **Public Key Encryption Schemes**

- ▶ Intuition
- ▶ RSA as encryption scheme

- **Diffie-Helman Key Agreement**

- ▶ Basic idea
- ▶ Man-in-the-middle attack

- **Quantum Computers**

Modular Arithmetic and Residue Class Rings

Let $\mathbb{Z}_n = \{\bar{0}, \bar{1}, \bar{2}, \dots, \overline{n-1}\}$ with $\bar{k} = \{x \in \mathbb{Z} \mid x \bmod n = k\}$

Addition: $\mathbb{Z}_n \times \mathbb{Z}_n \rightarrow \mathbb{Z}_n, \bar{a} + \bar{b} := \overline{a+b}$

Then, for all $\bar{a}, \bar{b} \in \mathbb{Z}_n$ it holds that

$$\bar{a} + \bar{b} = \bar{b} + \bar{a}$$

$$(\bar{a} + \bar{b}) + \bar{c} = \bar{a} + (\bar{b} + \bar{c})$$

$$\bar{0} + \bar{a} = \bar{a}$$

$$\bar{a} + \overline{n-a} = \bar{n} = \bar{0}$$

Multiplication: $\mathbb{Z}_n \times \mathbb{Z}_n \rightarrow \mathbb{Z}_n, \bar{a} \cdot \bar{b} := \overline{ab}$

Then, for all $\bar{a}, \bar{b} \in \mathbb{Z}_n$ it holds that

$$\bar{a} \cdot \bar{b} = \bar{b} \cdot \bar{a}$$

$$(\bar{a} \cdot \bar{b}) \cdot \bar{c} = \bar{a} \cdot (\bar{b} \cdot \bar{c})$$

$$\bar{1} \cdot \bar{a} = \bar{a}$$

\bar{a} is called invertible mod n if there is an \bar{x}

$\in \mathbb{Z}_n$ such that $\bar{a} \cdot \bar{x} = \bar{1}$

For ease of reading, we will denote \bar{k} as $k \bmod n$ in the rest of this lecture

Thus, $(\mathbb{Z}_n, +, \cdot)$ forms a **commutative ring with 1**

Example: Addition and Multiplication in \mathbb{Z}_6

+	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

•	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

- Invertible in \mathbb{Z}_6 :
 - ▶ 1, 5
 - Not invertible in \mathbb{Z}_6 :
 - ▶ 0, 2, 3, 4
 - Not all elements of $\mathbb{Z}_6 \setminus \{0\}$ are invertible
- $\Rightarrow (\mathbb{Z}_6, +, \cdot)$ is a ring but not a finite field

Example: Addition and Multiplication in \mathbb{Z}_5

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

•	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

- Invertible in \mathbb{Z}_5 :

- ▶ 1, 2, 3, 4

- Not invertible in \mathbb{Z}_5 :

- ▶ 0

- All elements of $\mathbb{Z}_5 \setminus \{0\}$ are invertible

$\Rightarrow (\mathbb{Z}_5, +, \bullet)$ is a finite field

Extended Euclidian Algorithm

Let $\text{gcd}(n, k)$ denote the greatest common divisor of n and k

Then there are integers x, y such that $xn + yk = \text{gcd}(n, k)$

Euclidian algorithm computes $\text{gcd}(n, k)$

Input: integers k, n with $n > k > 1$

Set $r_0 = n, r_1 = k$

WHILE $r_{i+2} > 0$

 Compute q_{i+1}, r_{i+2} with $r_i = q_{i+1} \cdot r_{i+1} + r_{i+2}$

END(WHILE)

RETURN $\text{gcd}(n, k) = r_{i+1}$

Extended Euclidian algorithm

Additionally computes x, y

Set $u_0 = v_1 = 1, u_1 = v_0 = 0$

WHILE $r_{i+2} > 0$

 Compute $u_{i+2} = u_i - q_{i+1} \cdot u_{i+1}$

 Compute $v_{i+2} = v_i - q_{i+1} \cdot v_{i+1}$

END(WHILE)

RETURN $x = u_{i+1}$ and $y = v_{i+1}$

Example

Euclidian algorithm to compute $\gcd(595, 408)$

Set $r_0 = 595, r_1 = 408$

$$r_0 = q_1 \cdot r_1 + r_2$$

$$595 = 1 \cdot 408 + 187$$

$$r_1 = q_2 \cdot r_2 + r_3$$

$$408 = 2 \cdot 187 + 34$$

$$r_2 = q_3 \cdot r_3 + r_4$$

$$187 = 5 \cdot 34 + \mathbf{17}$$

$$r_3 = q_4 \cdot r_4 + r_5$$

$$34 = 2 \cdot 17 + 0$$

$\Rightarrow \gcd(408, 595) = 17$

Extended Euclidian algorithm additionally computes x, y

Set $u_0 = v_1 = 1, u_1 = v_0 = 0$

$$u_2 = u_0 - q_1 u_1$$

$$v_2 = v_0 - q_1 v_1$$

$$u_2 = 1 - 1 \cdot 0 = 1$$

$$v_2 = 0 - 1 \cdot 1 = -1$$

$$u_3 = u_1 - q_2 u_2$$

$$v_3 = v_1 - q_2 v_2$$

$$u_3 = 0 - 2 \cdot 1 = -2$$

$$v_3 = 1 - 2 \cdot (-1) = 3$$

$$u_4 = u_2 - q_3 u_3$$

$$v_4 = v_2 - q_3 v_3$$

$$u_4 = 1 - 5 \cdot (-2) = \mathbf{11}$$

$$v_4 = -1 - 5 \cdot (3) = \mathbf{-16}$$

$$\Rightarrow \mathbf{11 \cdot 595 + (-16) \cdot 408 = 17}$$

Correctness of the Euclidian Algorithm

Observation: $\gcd(n, k) = \gcd(n - k, k)$

Proof:

- ▶ If d divides n and k , then there are r, s with $n = rd$ and $k = sd$
- ▶ Thus $n - k = (r - s)d$, so that d also divides $n - k$
- ▶ Thus, any divisor of n and k also divides $n - k$
- ▶ Vice versa if $d|k$ and $d|n - k$, then there are w, t with $n - k = wd$ and $k = td$
- ▶ Thus $n = n - k + k = (w + t)d$ and any divisor of $n - k$ and k also divides n

Consequence: $\gcd(n, k) = \gcd(n \bmod k, k) \Rightarrow \gcd(n, k) = \gcd(r_2, k) = \gcd(r_2, r_3) \dots$

Applying this repeatedly until the remainder $r_{i+2} = 0$ gives us $r_{i+1} = \gcd(r_{i-1}, r_i) = \gcd(n, k)$

Existence of Multiplicative Inverses

$a \in \mathbb{Z}_n$ is invertible mod $n \Leftrightarrow a$ and n are relatively prim $\Leftrightarrow \gcd(n, a) = 1$

Proof of “ \Rightarrow ” : Assume a is invertible

\Rightarrow there is an integer x such that $xa = 1 \pmod n$

\Rightarrow there is an integer k such that $xa = 1 + kn$

\Rightarrow there is an integer k such that $xa + (-k)n = 1$

Now if there was an integer d s.t. $d|a$ and $d|k$

$\Rightarrow d| xa + (-k)n$ and thus: $d| 1$

$\Rightarrow d = 1$ and thus a and n are relatively prime

Proof of “ \Leftarrow ”: Assume a and n are relatively prime.

Then $\gcd(a, n) = 1$

\Rightarrow there are integers x, y such that $xa + yn = 1$

$\Rightarrow xa = 1 - yn = 1 \pmod n$

$\Rightarrow x$ is the inverse of $a \pmod n$

\mathbb{Z}_n^* := Set of invertible elements in \mathbb{Z}_n

For p prime, $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$ and $(\mathbb{Z}_p, +, \cdot)$ is a field

Euler's φ function

- The number $|\mathbb{Z}_n^*|$ of invertible elements of \mathbb{Z}_n is called $\varphi(n)$

- For a prime number p it holds that $\varphi(p) = p - 1$

 - ▶ All elements of $\mathbb{Z}_p \setminus \{0\}$ are invertible mod p

$$\Rightarrow \varphi(p) = p - 1$$

- If $n = pq$ where p and q are **two different** prime numbers, then

$$\varphi(n) = (p - 1)(q - 1)$$

 - ▶ Not invertible: $p, 2p, 3p, \dots, (q - 1)p, qp \rightarrow q$ elements

 - ▶ Not invertible: $q, 2q, \dots, (p - 1)q \rightarrow$ another $p - 1$ elements

 - ▶ The other $n - q - (p - 1) = n - q - p + 1$ elements are invertible

$$\Rightarrow \varphi(n) = (p - 1)(q - 1)$$

Examples:

$$\mathbb{Z}_5^* = \{1, 2, 3, 4\},$$

$$\mathbb{Z}_4^* = \{1, 3\},$$

$$\mathbb{Z}_{10}^* = \{1, 3, 7, 9\}$$

Euler's Theorem

Euler's theorem:

For any $a \in \mathbb{Z}_n^*$: $a^{\varphi(n)} = 1 \pmod n$

Proof:

- ▶ If $a, b \in \mathbb{Z}_n^*$, then $a \cdot b \in \mathbb{Z}_n^*$
- ▶ Multiplying all elements of \mathbb{Z}_n^* with some $a \in \mathbb{Z}_n^*$ just reorders them:
 - Assume x is the product of all different $x_1, \dots, x_{\varphi(n)} \in \mathbb{Z}_n^*$
 - Then, for any $a \in \mathbb{Z}_n^*$: $ax_1ax_2 \dots ax_{\varphi(n)} = a^{\varphi(n)}x = x$
 - otherwise $ax_i = ax_j$ for some $i \neq j$
 - Multiplying the above equation with x^{-1} on both sides yields $a^{\varphi(n)} = 1 \pmod n$

Consequence:

For any $a \in \mathbb{Z}_n^*$ and any integer s it holds that $a^{\varphi(n)s+1} = a \pmod n$

Generalization of Euler's Theorem

Generalization:

Let $n = pq$ where p and q are **two different** prime numbers then

for all $a \in \mathbb{Z}_n$ it holds that $a^{\varphi(n)+1} = a \pmod n$

Proof: For $a = 0$ the equation obviously holds

For all invertible $a \in \mathbb{Z}_n$ we already proofed it on the last slide

So, lets assume an $a \in \mathbb{Z}_n$ that is not invertible

- ▶ Then it is either divisible by p or by q (or both but then $a = 0$).
- ▶ Let's assume a is not divisible by p but divisible by q .
- ▶ Then, $a^{p-1} = 1 \pmod p$ and $a^{q-1} = 0 \pmod q$
- ▶ Thus, $a^{\varphi(n)+1} = (a^{p-1})^{q-1}a = a \pmod p$ and $a^{\varphi(n)+1} = (a^{q-1})^{p-1}a = a \pmod q$
- ▶ Thus, there are integers r and s with $a^{\varphi(n)+1} = a + rp$ and $a^{\varphi(n)+1} = a + sq$

- ▶ Consequently, $rp = sq$ such that $q \mid r$
- ▶ So, there is an integer l with $r = lq$
- ▶ Thus, $a^{\varphi(n)+1} = a + rp = a + lqp = a + ln$
 $\Rightarrow a^{\varphi(n)+1} = a \pmod n$

The Factorization Problem

Definition

Given a composite integer n , find a non-trivial factor of n

Hardness of Factorization

- ▶ No known polynomial time algorithms for factorization on **classical computers**
- ▶ Best current algorithms for **classical computers** have sub-exponential run-time
 - Pollard's Rho Method
 - Quadratic Sieve
 - Number Sieve
 - ...

The Discrete Logarithm Problem

Definition **DL Problem**

Given a cyclic group G , a generator $g \in G$, and g^x but not x , find the **discrete logarithm** x .



Definition **Decisional Diffie-Hellman Problem**

Given a cyclic group G , a generator $g \in G$, and g^x, g^y, g^z but not x, y, z , decide if $g^{xy} = g^z$.

- The security of many asymmetric cryptosystems is based on the hardness of the discrete logarithm problem or the decisional Diffie-Hellman problem
- Relation between the two problems
 - ▶ If in a group the discrete logarithm problem can be solved, the DDH problem can also be solved

Overview

- **Basic Number Theory**

- ▶ Finite Fields, greatest common divisor, Fermat's theorem
- ▶ Factorization
- ▶ Discrete Logarithms

- **Digital signature schemes**

- ▶ Intuition
- ▶ RSA as signature scheme
- ▶ Digital signature standard

- **Public Key Encryption Schemes**

- ▶ Intuition
- ▶ RSA as encryption scheme

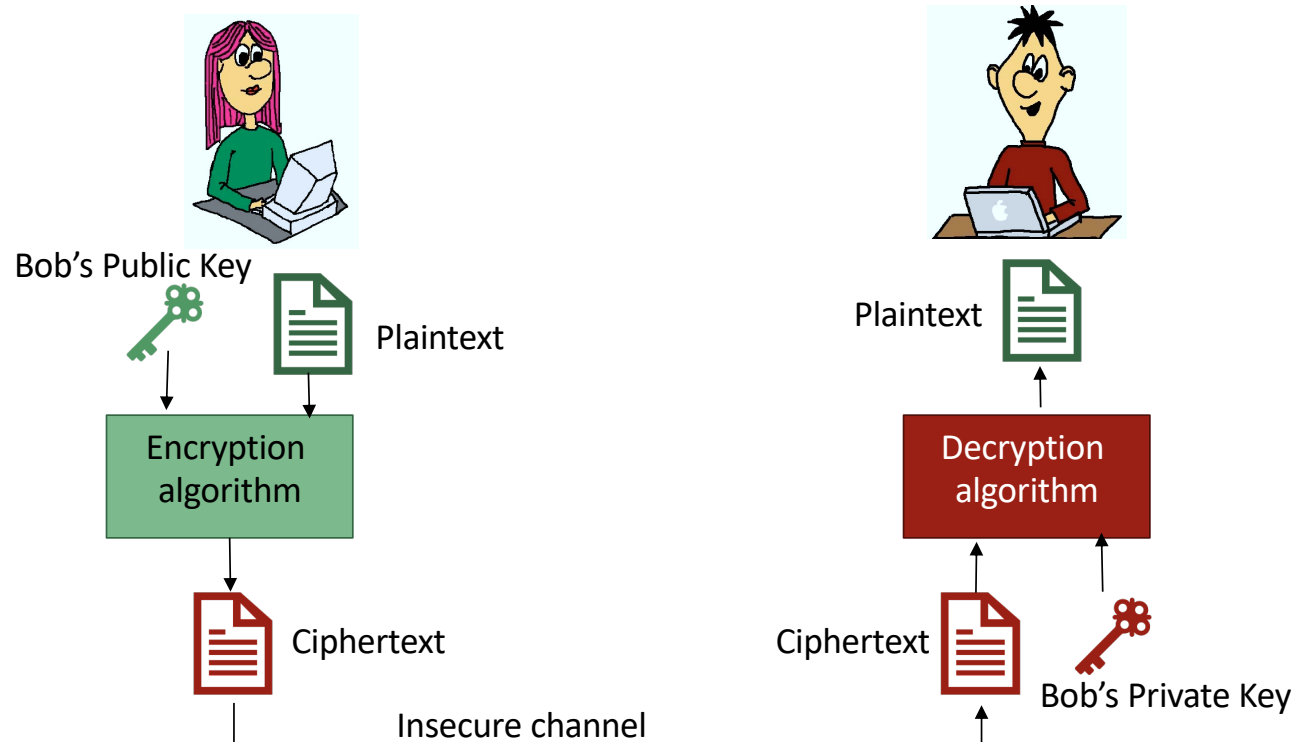
- **Diffie-Helman Key Agreement**

- ▶ Basic idea
- ▶ Man-in-the-middle attack

- **Quantum Computers**

Intuition Public Key Encryption

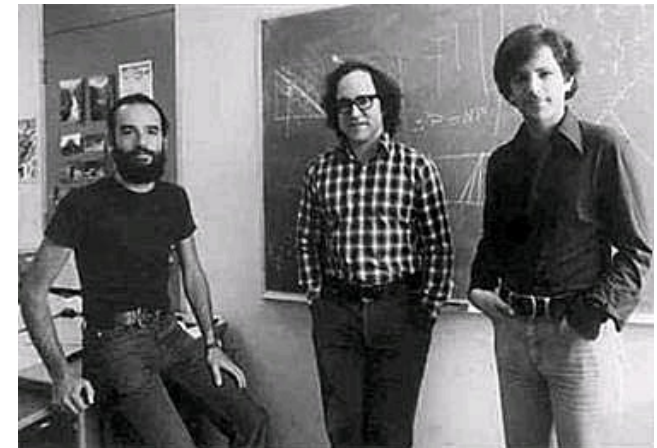
- Alice wants to send a **confidential** plaintext to Bob
- Alice has an authentic copy of Bob's **public key**
- Alice uses Bob's public key to **encrypt** plaintext to ciphertext
- Bob uses his **private key** to **decrypt** ciphertext to plaintext
- Decryption is **"difficult"** without the **private key**



Note: The definition of an encryption scheme presented in Chapter 2 also holds for asymmetric encryption!

RSA

- **First asymmetric encryption scheme invented in 1977**
 - ▶ By Ron Rivest, Adi Shamir, and Leonard Adleman at MIT
 - ▶ Original idea of asymmetric encryption goes back to Diffie and Hellman, though
- **Patented from 1983 to 2000**
- **Supports different key lengths and variable block sizes**
 - ▶ Currently, 2048 bit keys are considered sufficient
 - ▶ Implies a block length of 2048 bit
- **Requires plaintext blocks to be represented as integers**
 - ▶ Requires a coding scheme that converts bit strings in integers



RSA Key Generation

Public Key

- ▶ Randomly select two different large prime numbers p, q
- ▶ Set $n := pq$
- ▶ Chose $e \in \mathbb{Z}_n$ such that e is invertible mod $\varphi(n)$
- ▶ Set public key to (n, e)

Private Key

- ▶ Compute $d \in \mathbb{Z}_n$ such that $ed = 1 \pmod{\varphi(n)}$
 - \exists integer k such that $ed = 1 + k\varphi(n)$
- ▶ Set private key to d

Side Notes



- ▶ Large prime numbers can be found by
 - Choosing random numbers of appropriate size
 - Testing for primality with probabilistic primality tests
- ▶ If the desired bit length of the modulus is k than p and q should be $k/2$ -bit prime numbers
- ▶ Choose $e \in \mathbb{Z}_n$ randomly; check if $\gcd(e, n) = 1$
- ▶ Compute d from e with the Extended Euclidian Algorithm

RSA Operation

Encryption

For a public RSA key $pk = (e, n)$,

$$E_{pk} : \mathbb{Z}_n \rightarrow \mathbb{Z}_n$$

$$E_{pk}(m) = c = m^e \bmod n$$

Decryption

For the corresponding private RSA key $sk = d$

$$D_{sk} : \mathbb{Z}_n \rightarrow \mathbb{Z}_n$$

$$D_{sk}(c) = c^d = m \bmod n$$

Correctness of RSA

For any ciphertext $c \in \mathbb{Z}_n$:

$$c^d = m^{ed} \bmod n = m^{\varphi(n)k+1} \bmod n = m \bmod n$$

Small Example:

Key generation:

Let $p = 3$, $q = 5$, then $n = pq = 15$

$$\varphi(n) = 2 \cdot 4 = 8$$

Chose $e = 3$, then e is invertible mod $\varphi(n)$ as 8 and 3 are relatively prime

Setting $d = 3$ we get $ed = 9 = 1 \bmod 8$

Encryption of $m = 7$:

$$m^e \bmod n = 7^3 \bmod 15 = 343 \bmod 15 = 13$$

Decryption of $c = 13$:

$$c^d \bmod n = 13^3 \bmod 15 = 2197 \bmod 15 = 7$$

Efficient Modular Exponentiation

- **RSA Encryption and Decryption: $x^k \bmod n$**
- **“Naïve” modular exponentiation**
 - ▶ Requires k modular multiplications
 - ▶ Problem: the size of the exponent is of the same order as the size of the modulus n
 - ▶ Naïve modular exponentiation is not efficient

- **More efficient modular exponentiation**
- **Idea: Use the binary representation of k**
 - ▶ $k = \sum k_i 2^i = k_0 + 2(k_1 + 2(k_2 + \dots) \dots)$
where $k_i \in \{0,1\}$
 - ▶ Then we get $x^k = \prod x^{k_i 2^i}$
 - ▶ So, all we need to do is square and multiply

Example

- ▶ $k = 37 = 1 + 2^2 + 2^5$
- ▶ So $x^{37} = x \cdot x^{2^2} \cdot x^{2^5} = (((((x^2)^2)^2 x)^2)^2)^2 x$
- ▶ Two multiplications by x and 5 squares

RSA Security (1)

Theorem:

Let p, q be prime numbers and $n = p \cdot q$

Then n can be efficiently factorized iff $\varphi(n)$ can be computed efficiently

Proof:

“ \Rightarrow ”: If n can be efficiently factorized then p and q can efficiently be computed from n and therefore

$\varphi(n) = (p - 1) \cdot (q - 1)$ is efficiently computable

“ \Leftarrow ”: If $\varphi(n)$ is known, then one can compute p and q

from the two equations $n = p \cdot q$ and $\varphi(n) = (p - 1) \cdot (q - 1)$

 **Factorizing n is equivalent to computing $\varphi(n)$**

RSA Security (2)

Theorem:

Let p, q be prime numbers and $n = p \cdot q$ and (e, n) a public RSA key and d the corresponding private key. Then d can be efficiently computed from (e, n) iff n can be factorized efficiently.

Proof:

“ \Rightarrow ”: There is a probabilistic polynomial-time algorithm that computes p and q from d, e , and n

“ \Leftarrow ”: clear: if we can factorize n we have p and q and can compute $\varphi(n)$ and can thus compute d as the inverse of e mod $\varphi(n)$



Computing d is equivalent to factorizing n

RSA Security (3)

Summary:

- ▶ Compute a private RSA key d from public key (e, n) is equivalent to factorizing n
- ▶ Factorizing n is equivalent to computing $\varphi(n)$



It is still unclear if there is a way to decrypt RSA-encrypted messages without knowledge of the private key d

Recall Hardness of Factorization:

- ▶ For classical computers, there is currently no polynomial-time algorithm for factorization

Chosen Plaintext Attack Against RSA

Recall from Chapter 2: chosen plaintext attack against a cipher

- ▶ Attacker can obtain ciphertext for plaintexts of its choice

Example: RSA can always be attacked in a chosen plaintext setting

- ▶ Any attacker with access to the public key (e, n) can generate ciphertexts for plaintexts of its choice
 - Attacker chooses m and computes $c = m^e \bmod n$



For deterministic asymmetric ciphers we always need to consider a chosen plaintext setting as realistic

Semantic Security

Definition: Semantic Security

- ▶ Assume a challenger chooses two plaintexts m_1 and m_2
- ▶ He encrypts the plaintexts with a public key pk $c_1 = E_{pk}(m_1)$ and $c_2 = E_{pk}(m_2)$
- ▶ He then provides m_1, m_2, c_1, c_2 and pk to an adversary
- ▶ Then the public key encryption schemes is said to be **semantically secure**
 - if the adversary cannot guess with a probability larger than $\frac{1}{2}$ which ciphertext encrypts which plaintext



Deterministic asymmetric ciphers like (textbook) RSA are not semantically secure

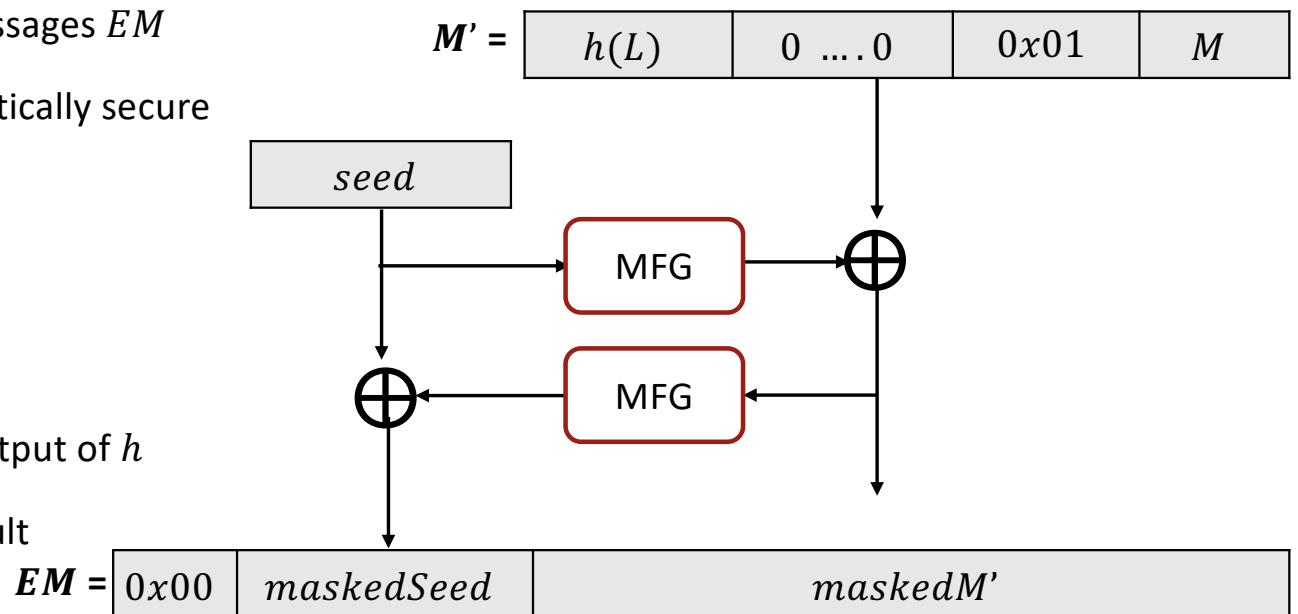
Turning RSA into a Semantically Secure Cipher with OAEP

- **The Optimal Asymmetric Encryption Padding OAEP**

- ▶ Converts message M into encoded messages EM
- ▶ Uses random seed to make RSA semantically secure

- **Notations**

- ▶ M : bit-string message to encrypt
- ▶ h : hash function
- ▶ $seed$: random seed, same length as output of h
- ▶ L : optional label, empty string by default
- ▶ MGF: mask generation function
- ▶ Padding with zeros:
 - let n be a k -byte modulus, then $k - |M| - 2|h(L)| - 2$ bytes of zero bytes are used as padding



Backdoors in Key Generation

• Idea

- ▶ Whenever RSA is used,
 - keys must be generated
- ▶ Whoever implements these key generation
 - can manipulate the code such that keys generated with it include a backdoor
- ▶ This backdoor allows him to
 - retrieve the private key corresponding to a public key generated with his implementation

• Underlining Model

- ▶ Manufacturer (Attacker)
 - Designer of the backdoor
 - Integrates the backdoor in the key generation code
- ▶ User (Victim)
 - In possession of a device or piece of code for key generation, e.g. for RSA, manipulated by the manufacturer
 - Can observe public and private keys generated by his device
- ▶ External attacker
 - Can observe public keys used by the user

Backdoor for RSA Key Generation

Naïve RSA Backdoor

- ▶ Key generation code with backdoor
 - Fix a prime number p
 - Choose a second prime number q at random
 - Set $n = qp$
 - Select e relatively prime to $\varphi(n)$ and d such that $ed = 1 \bmod \varphi(n)$

Exploiting the backdoor

- ▶ If manufacturer sees that user uses (e, n)
 - compute q by n/p , from q, p, e compute d

Unfortunately

- ▶ External attacker that observes two public keys (e, n) and (e', n') can compute $p = \gcd(n, n')$
 - Thus, any external attacker that suspects this backdoor can check for it
- ▶ User can check if the code/devices has this backdoor in the same way

Backdoor for RSA Key Generation

Better RSA Backdoor

- ▶ Manufacturer's RSA key pair (E, N) and D
- ▶ Key generation code with backdoor
 - Pick random prime numbers p and q and set $n = pq$
 - Compute $e = p^E \bmod N$
 - Check if e is invertible $\bmod \phi(n)$
 - If yes, compute the inverse d and output $(e, n), d$
 - If no, pick a new prime number p and start again

Exploiting the backdoor

- ▶ If manufacturer sees that client uses (e, n)
- ▶ Compute $e^D \bmod N = p$ and can use this to compute q and then d

External attacker and user

- ▶ Cannot check for this backdoor as they do not have the private key D
- ▶ To the user e looks as if it was randomly picked



Backdoors like this exist for the key generation operations of many public key cryptosystems

Overview

- **Basic Number Theory**

- ▶ Finite Fields, greatest common divisor, Fermat's theorem
- ▶ Factorization
- ▶ Discrete Logarithms

- **Digital signature schemes**

- ▶ Intuition
- ▶ RSA as signature scheme
- ▶ Digital signature standard

- **Public Key Encryption Schemes**

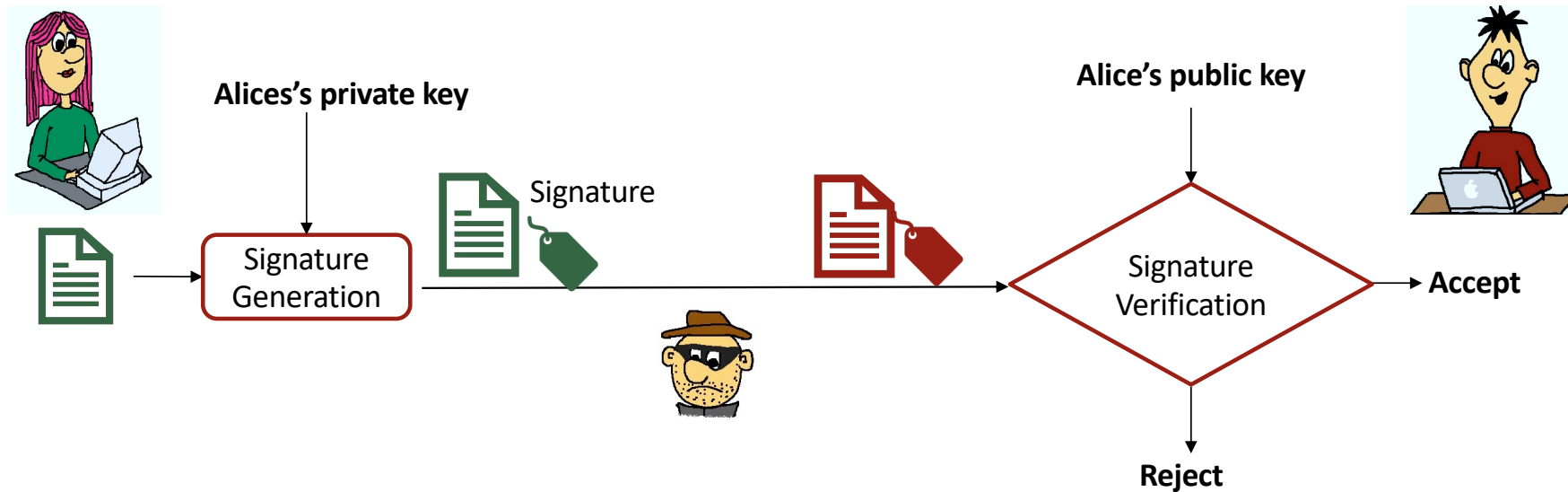
- ▶ Intuition
- ▶ RSA as encryption scheme

- **Diffie-Helman Key Agreement**

- ▶ Basic idea
- ▶ Man-in-the-middle attack

- **Quantum Computers**

Intuition Digital Signatures



- Alice uses her private key to generate a signature on the message
- Anyone in possession of Alice's public key can verify the signature
- Difficult to generate a message, signature pair that is accepted by the signature verification
 - ▶ Without access to the private key

Definition Digital Signature Scheme

A digital **signature scheme** consists of

- ▶ A key generation algorithm that
 - generates a public key pk for signature verification
 - generates a private key sk for signature generation
- ▶ A family of signature generation algorithms sig_{sk} that
 - takes a message M as input and outputs the signature $\text{sig}_{sk}(M)$
- ▶ A family of signature verification algorithms ver_{pk} that
 - takes a message M and a signature $\text{sig}_{sk}(M)$ as input and
 - returns success or failure

Naïve RSA Signatures (Insecure!)

Key generation as in RSA Encryption

Public Key

- ▶ Randomly select two large prime numbers p, q
- ▶ Set $n := pq$
- ▶ Chose $e \in \mathbb{Z}_n$ such that e is invertible mod $\varphi(n)$
- ▶ Set public key $pk = (n, e)$

Private Key

- ▶ Compute private key $sk = d \in \mathbb{Z}_n$ such that $ed = 1 \pmod{\varphi(n)}$

Signature generation

- ▶ signature s on message m : $s = m^d \pmod{n}$

Signature verification

- ▶ $s^e = m^{de} \stackrel{?}{=} m$

Vulnerable to existential forgery

- ▶ Attacker can choose signature s and compute $m = s^e$ and then claim that (m, s) is a valid signature

RSA Signature Scheme

Key generation as in Naïve RSA

Signature generation

- ▶ Let h be a publicly known cryptographic hash function
- ▶ Signature s on m is $s = h(m)^d$

Signature verification

- ▶ On receipt of (\bar{m}, \bar{s}) verifier checks if $h(\bar{m}) \stackrel{?}{=} \bar{s}^e \pmod{n}$

Secure against existential forgery

- ▶ Attacker cannot find a message m such that $h(m) = s^e$ as h is pre-image resistant

Hashing before signing is also required for security 
reasons in many other asymmetric signature schemes

Attacks on Digital Signatures

Attack result

- ▶ **Total break:** (partial) recovery of the signature key
- ▶ **Universal forgery:** forge signatures on any message of the attacker's choice
- ▶ **Selective forgery:** forge a signature on a specific chosen message
- ▶ **Existential forgery:** merely results in some valid message/signature pair not already known to the adversary

Power of attacker

Strength of attacker increases

- ▶ **Key-Only Attack:** Attacker only in possession of the public verification key
- ▶ **Known-Message Attack:** Attacker observes some message/signature pairs; tries to generate another valid pair
- ▶ **Chosen-Message Attack:** Attacker can choose messages and can make the signer sign them; tries to generate another valid pair

Digital Signature Algorithm

- **Adopted as standard by NIST in 1994**
- **Standardized in FIPS 186**
- **Security is based on the DDH assumption**
 - ▶ Related to but stronger than the Discrete Logarithm problem
- **Can be defined over different cyclic groups for which DDH assumption seems to hold, e.g.**
 - ▶ Cyclic sub-groups of order q of \mathbb{Z}_p^* , where p and q are prime numbers where q divides $(p - 1)$
- **Variants for other cyclic groups exist**
 - ▶ E.g. ECDSA on specific elliptic curves over a finite field

Key Generation for DSA

Public parameters

- ▶ Two prime number p, q with $q | (p-1)$
- ▶ $x \in \mathbb{Z}_p^*$ such that $g := x^{\frac{p-1}{q}} \bmod p \neq 1$
 - The smallest interger i or which $g^i = 1 \bmod p$ is $i = q$
 - Thus, g generates a sub group of order q in \mathbb{Z}_p^*
- ▶ Cryptographic hash function h

Private key

- ▶ Chose $a \in \{1, \dots, q - 1\}$ uniformly at random and set $sk = a$

Public key

- ▶ Set $A = g^a \bmod p$ as public key pk

Example

Parameters

- ▶ $p = 11, q = 5$
- ▶ Select $x = 2$, then $g = 4$

Private key

- ▶ Chose $a = 3$

Public key

- ▶ Set $A = g^a \bmod p = 4^3 \bmod 11 = 9$

DSA Operation

Signature generation on message m

- ▶ Chooses $k \in \{1, \dots, q - 1\}$ uniformly at random

- ▶ Signer computes

$$r = (g^k \bmod p) \bmod q$$

$$s = k^{-1}(h(m) + ar) \bmod q$$

- ▶ Signature: $\text{sig}_{sk}(m) = (r, s)$

Signature verification

- ▶ Upon receipt of m, r, s the verifier
- ▶ Checks if $r \in \{1, \dots, q - 1\}$ and $s \in \{1, \dots, q - 1\}$
- ▶ Computes $u_1 = h(m)s^{-1} \bmod q$, $u_2 = rs^{-1} \bmod q$
- ▶ Computes $v = g^{u_1} A^{u_2} \bmod p \bmod q$
- ▶ Accept signature if $v = r$, reject otherwise

Correctness of Verification

- Upon receipt of m, r, s the verifier computes

$$\begin{aligned}v &= g^{u_1} A^{u_2} \bmod p \bmod q \\&= g^{h(m)s^{-1}} A^{rs^{-1}} \bmod p \bmod q \\&= g^{h(m)s^{-1} + ars^{-1}} \bmod p \bmod q \\&= g^{s^{-1}(h(m)+ar)} \bmod p \bmod q \\&\rightarrow = g^{s^{-1}sk} \bmod p \bmod q \\&= g^{s^{-1}sk} \bmod p \bmod q \\&= r\end{aligned}$$

g was selected such that $g^q = 1 \bmod p$, thus
 $g^k \bmod p = g^{k \bmod q} \bmod p$

Reusing k leads to a total break of DSA

Assume k is used to sign two known messages m_1 and once for m_2 , then

$$r = (g^k \bmod p) \bmod q \text{ (same for both messages)}$$

$$s_1 = (k^{-1}(h(m_1) + ar)) \bmod q$$

$$s_2 = (k^{-1}(h(m_2) + ar)) \bmod q$$

$$\text{Thus, } s_1 - s_2 = k^{-1}(h(m_1) - h(m_2)) \bmod q$$

$$\text{and therefore: } k = (s_1 - s_2)^{-1}(h(m_1) - h(m_2)) \bmod q$$

$$\text{And thus, } a = r^{-1}(s_1 k - h(m_1)) \bmod q$$

I.e., private key a can be computed by anyone observing the messages and signatures if the same k is used twice

MACs versus Digital Signatures

- **MACs can provide**

- ▶ Message integrity
- ▶ Origin authentication

- **Require verifier to share a secret key with MAC producer**

- **Signature Schemes can provide**

- ▶ Message integrity
- ▶ Origin authentication
- ▶ Broadcast authentication
- ▶ Non-repudiation

- **Require verifier to obtain an authentic copy of public key of signer**

Overview

- **Basic Number Theory**

- ▶ Finite Fields, greatest common divisor, Fermat's theorem
- ▶ Factorization
- ▶ Discrete Logarithms

- **Digital signature schemes**

- ▶ Intuition
- ▶ RSA as signature scheme
- ▶ Digital signature standard

- **Public Key Encryption Schemes**

- ▶ Intuition
- ▶ RSA as encryption scheme

- **Diffie-Helman Key Agreement**

- ▶ Basic idea
- ▶ Man-in-the-middle attack

- **Quantum Computers**

Diffie-Hellman (DH) Key Agreement

- **Oldest public key mechanism**
 - ▶ Invented in 1976
- **Is a key establishment protocol by which two parties can**
 - ▶ Establish a symmetric secret key K
 - ▶ Based on publicly exchanged values
- **Security based on hardness of discrete logarithm problem**
 - ▶ Any polynomial-time algorithm that solves the DL problem also solves the computational DH-problem:
 - Given a prime number p , a generator g of \mathbb{Z}_p^* , g^a, g^b find $K = g^{ab}$
 - ▶ It is unknown if the computational DH-problem can be solved without solving the DL problem



Diffie-Hellman Key Agreement

Public parameters

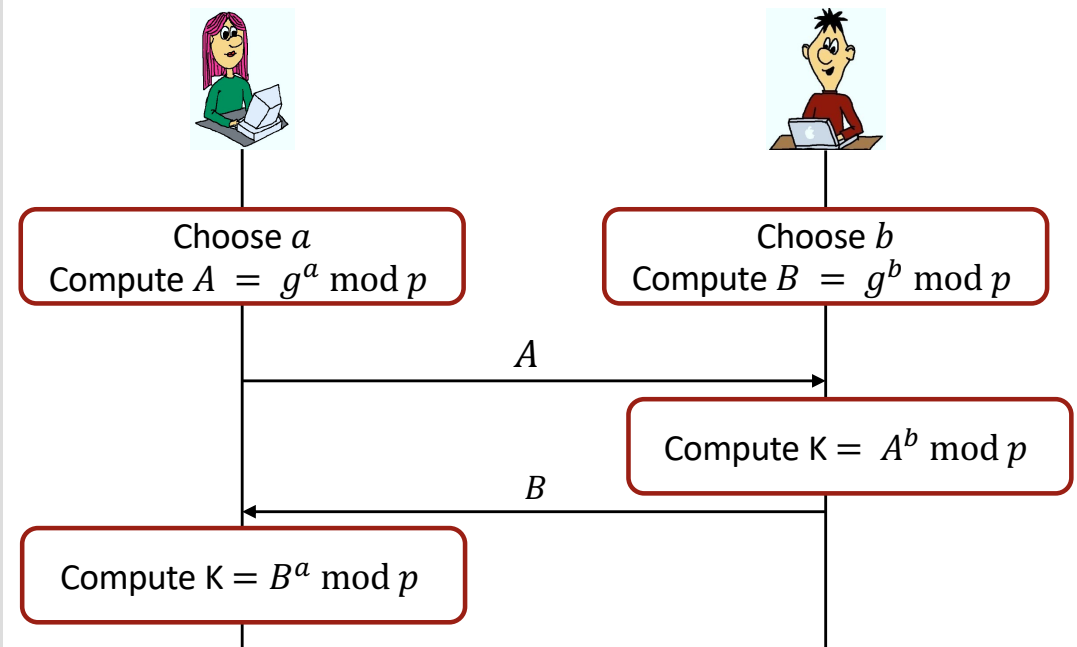
- ▶ Prime number p , generator g of \mathbb{Z}_p^*

Private values

- ▶ Private DH-value of Alice
 - $a \in \{2, \dots, p - 2\}$ chosen uniformly at random
- ▶ Private DH-value of Bob
 - $b \in \{2, \dots, p - 2\}$ chosen uniformly at random

Public values

- ▶ Public DH-value of Alice $A = g^a \bmod p$
- ▶ Public DH-value of Bob $B = g^b \bmod p$



$$\text{As } A^b \bmod p = g^{ab} = g^{ba} = B^a \bmod p$$

Alice and Bob now share the secret key $K = g^{ab}$

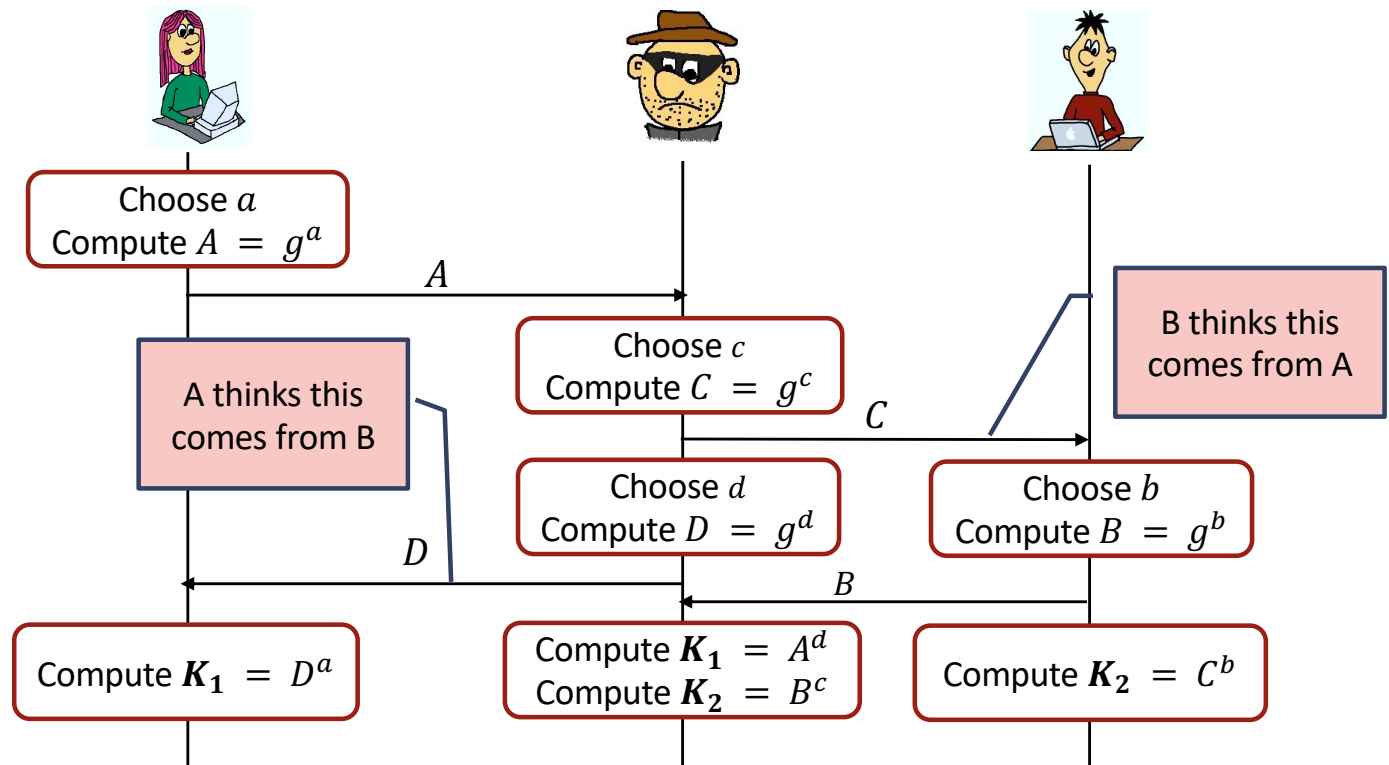
Man-in-the-Middle Attack

All computations are done mod p and a, b, c, d are chosen from $\{2, \dots, p - 2\}$

Result

- ▶ A shares K_1 with attacker
 - but thinks she shares it with B
- ▶ B shares K_2 with attacker
 - but thinks he shares it with A
- ▶ A and B do not share key
 - but they think they do

⇒ **Attacker can eavesdrop!**



Symmetric vs. Asymmetric Cryptography

Symmetric Cryptography

- ▶ More efficient
 - Often used to encrypt large amounts of data
- ▶ Higher number of secret keys required
 - $n(n - 1)/2$ keys required to enable pairwise confidential communication between n parties
- ▶ Secret keys need to be distributed
 - Need to ensure confidentiality and authenticity

Asymmetric Cryptography

- ▶ Less efficient
 - Rarely used to encrypt longer messages
- ▶ Lower number of private keys required
 - n keys required in order to enable pairwise confidential communication between n parties
- ▶ Only public keys need to be distributed
 - Need to ensure authenticity of public keys but not confidentiality



In practice, the best of both worlds is often combined: asymmetric cryptography is used to establish secret keys which are then used for symmetric encryption and integrity protection

Overview

- **Basic Number Theory**

- ▶ Finite Fields, greatest common divisor, Fermat's theorem
- ▶ Factorization
- ▶ Discrete Logarithms

- **Digital signature schemes**

- ▶ Intuition
- ▶ RSA as signature scheme
- ▶ Digital signature standard

- **Public Key Encryption Schemes**

- ▶ Intuition
- ▶ RSA as encryption scheme

- **Diffie-Helman Key Agreement**

- ▶ Basic idea
- ▶ Man-in-the-middle attack

- **Quantum Computers**

Quantum Computers and Traditional Asymmetric Schemes

- **1994 Peter Shor developed two polynomial time quantum algorithms**
 - ▶ A **factorization algorithm** that can factorize large compound numbers
 - ▶ A **discrete logarithm algorithm** that can compute the discrete logarithm x of $g^x \bmod p$ for a given prime number p and generator g
- **All classical asymmetric schemes can be broken with a large enough quantum computer, e.g.**
 - ▶ RSA signature scheme and RSA encryption scheme
 - ▶ DSA
 - ▶ Diffie-Helman Key Agreement
 - ▶ Elliptic Curve Cryptosystems like ECDSA, ECDH
- **Lead to NIST calls for quantum secure encryption, signature, and key agreement schemes**
 - ▶ New post quantum algorithms selected in 2022

Quantum Computers and Traditional Symmetric Schemes

- **Grover's algorithm (1996) enables breaking symmetric encryption schemes like AES in $O(2^{n/2})$ where n is the bit length of the key**
 - ▶ Thus, it is currently believed that doubling the key size for symmetric encryption suffices
- **No known algorithm to find collisions for hash functions faster than on classical computers yet**
 - ▶ Cryptographic hash functions are currently believed not to be affected by quantum computers

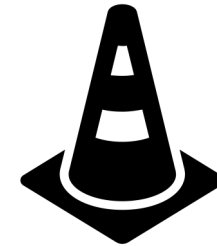
Summary

- **Asymmetric encryption schemes: confidentiality**

- ▶ Most prominent example: RSA
 - Security depends on hardness of factorization

- **Digital signature schemes: integrity protection**

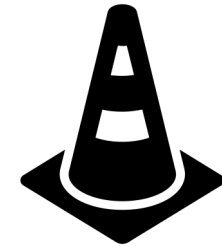
- ▶ Most prominent examples: RSA, DSS
 - Security of DSS depends hardness of computing discrete logarithms
- ▶ All signature schemes require hashing before signing
- ▶ Provide non-repudiation and broadcast integrity protection
 - which cannot be provided by symmetric integrity protection via MACs



Summary

- **Diffie-Helman Key Agreement: establish secret key**

- ▶ Can be used to establish a shared secret key for a symmetric scheme
- ▶ Is itself an asymmetric scheme
- ▶ Security depends on hardness of discrete logarithm
- ▶ Is in its basic version vulnerable to a man-in-the-middle attack



- **All asymmetric schemes require authentic public keys**

- ▶ Need to be able to obtain authentic copy of the public keys of other entities

- **All classical asymmetric schemes can be broken by large enough quantum computers**

References

- **Johannes Buchmann, Einführung in die Kryptographie, Springer Verlag 2016**
 - ▶ Chapter 8
- **W. Stallings, Cryptography and Network Security: Principles and Practice, 8th edition, Pearson 2022**
 - ▶ Chapter 9: Public Key Encryption and RSA
 - ▶ Chapter 10: Other Public Key Cryptosystems
 - Diffie Hellman
 - ▶ Chapter 13: Digital Signatures



IT-Security

Chapter 5: Authentication and Key Establishment

Prof. Dr.-Ing. Ulrike Meyer



Overall Lecture Context

- **In the last chapters we covered**

- ▶ Symmetric and asymmetric mechanisms to provide
- ▶ Integrity protection
 - Message Authentication Codes and digital signatures schemes
- ▶ Confidentiality
 - Symmetric and asymmetric encryption schemes

- **All these mechanisms require keys to be distributed**

- ▶ to the authentic entities

- **In this chapter we learn how to**

- ▶ authenticate entities, i.e., check that they are who they claim to be
- ▶ establish keys between different entities

Overview

● Building Blocks for Entity Authentication

- ▶ Definition of Entity Authentication
- ▶ MAC-based authentication
- ▶ Signature-based authentication

● Key Distribution with trusted Third Parties

- ▶ Key Distribution Centers
- ▶ Certificates and Public Key Infrastructures

● Authenticated Session Key Establishment

- ▶ Definitions around session key establishment
- ▶ Authenticated Diffie Hellman variants
- ▶ Session key establishment w-o DH
- ▶ Session Key derivation principles

● Password-based authentication

- ▶ Password-based user authentication
- ▶ Password-based authenticated key establishment
- ▶ Dictionary attacks on password-based authentication

Definition of Entity Authentication

Unilateral entity authentication of A to B

- ▶ A (claimant) proves its identity to B (verifier)
- ▶ B is assured that A is currently interacting with B

Mutual authentication

- ▶ A authenticates to B and B authenticates to A

Objectives

- ▶ **Correctness**: A can always successfully authenticate to B
- ▶ **Resistance against transferability**: After A authenticated to B successfully, B cannot authenticate as A to C (*)
- ▶ **Resistance against impersonation**: $C \neq A$ cannot make B believe that it is A (*)

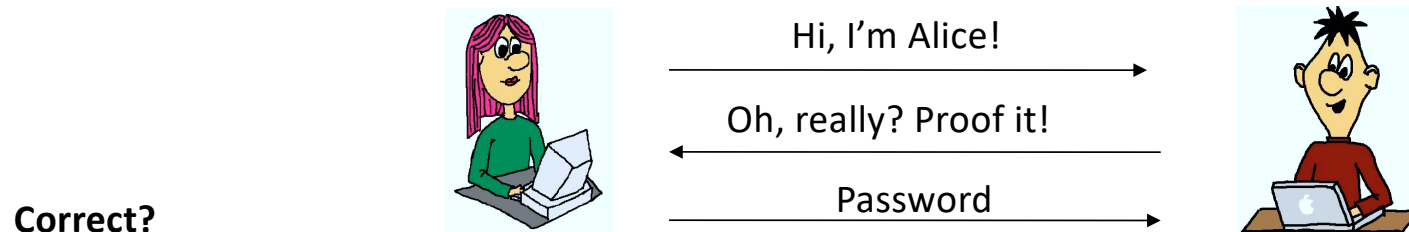
All three objectives still hold

- ▶ if an attacker has observed multiple authentication instances between A and B

(*) Except for with negligible probability: guessing is of course always possible

Example

- Assume **A** and **B** have agreed upon a secret password when they last met
- Now **A** authenticates to **B** with the following protocol



- ▶ Yes!

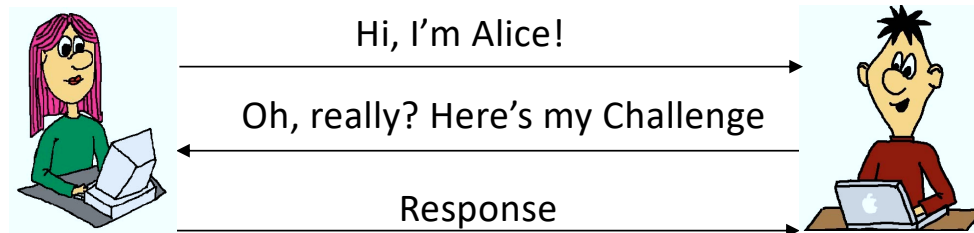
Resistant against transferability?

- ▶ Yes, at least if Alice does not use the password in multiple places

Resistant against impersonation?

- ▶ No! The password is sent in the clear so any eavesdropper can impersonate Alice after the first run of the protocol

Challenge-Response Authentication

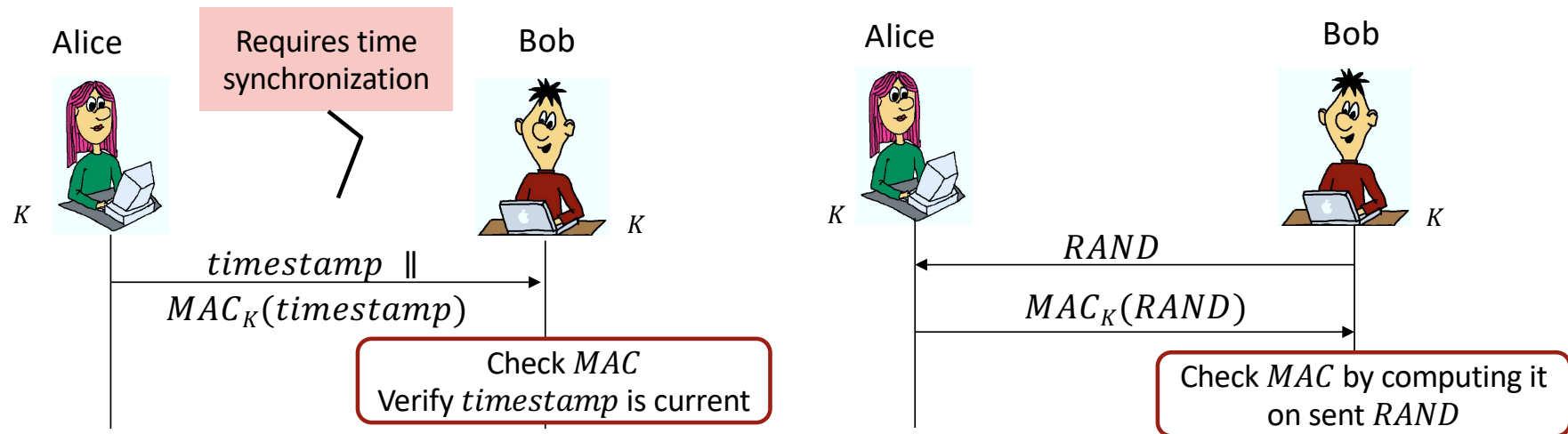


Idea:

- ▶ **B** generates a fresh challenge
 - E.g., a random number or a time stamp (implicit challenge)
- ▶ **A** proves its identity by computing a response that
 - Depends on the challenge and a secret
 - Secret can be a secret key shared with B, a private key of A,...

Response Calculation must guarantee that the objectives hold

Example Building Blocks for Unilateral Entity Authentication based on shared key K

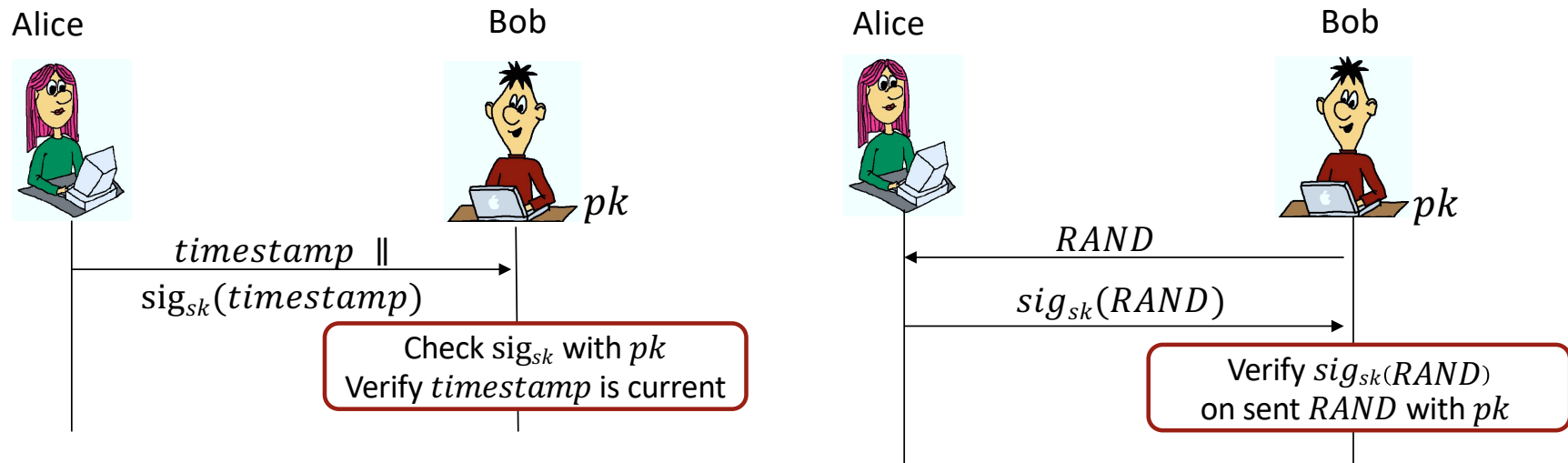


- ▶ Alice computes a MAC on timestamp
- ▶ Sends timestamp and MAC to Bob
- ▶ Bob verifies MAC by computing MAC on received timestamp and comparing it to received MAC
- ▶ Bob checks if $timestamp$ is in an acceptable range around Bob's current time

- ▶ Bob selects a random number $RAND$ as challenge and sends it to Alice
- ▶ Alice computes a MAC on $RAND$ using K
- ▶ Bob verifies that the received MAC corresponds to the one he computes using $RAND$ as input

Example Building Blocks for Unilateral Entity Authentication

Unilateral authentication of A to B based on a private key sk of Alice assuming Bob knows Alice's public key pk

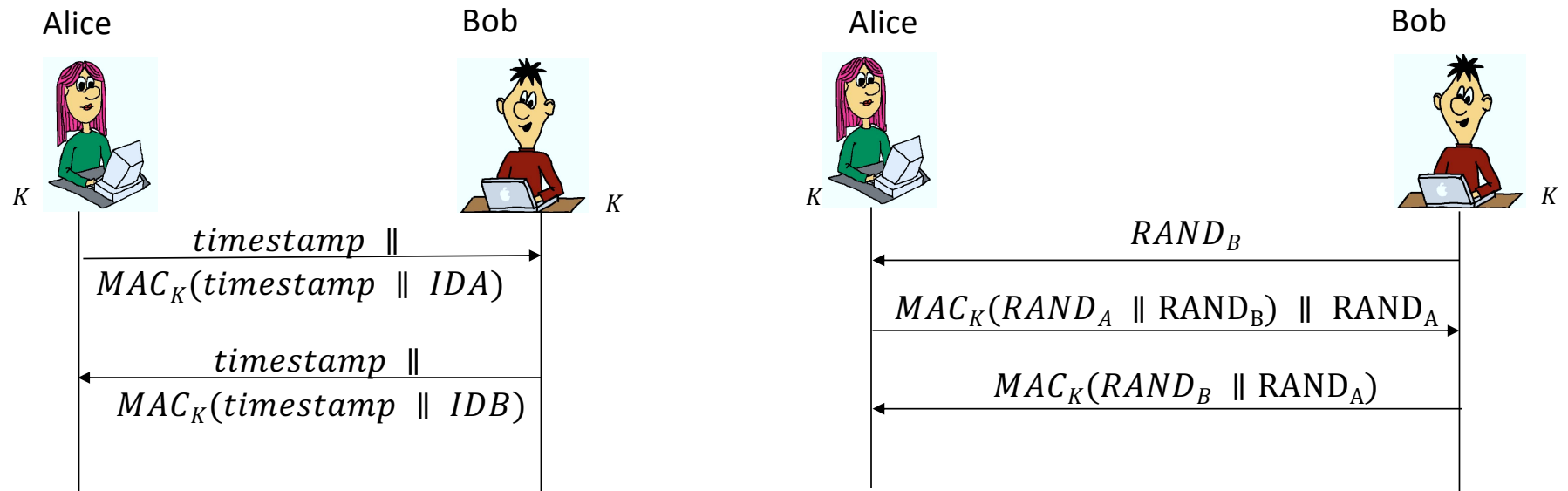


- ▶ Alice computes a signature on the current timestamp (implicit challenge) using sk
- ▶ Sends the $timestamp$ and the signature to Bob
- ▶ Bob verifies signature with pk and checks if $timestamp$ is in an acceptable range

- ▶ Bob selects a random number $RAND$ as challenge and sends it to Alice
- ▶ Alice computes a signature on $RAND$
- ▶ Bob verifies that the received signature is a signature on the sent $RAND$

Example Building Blocks for Mutual Entity Authentication

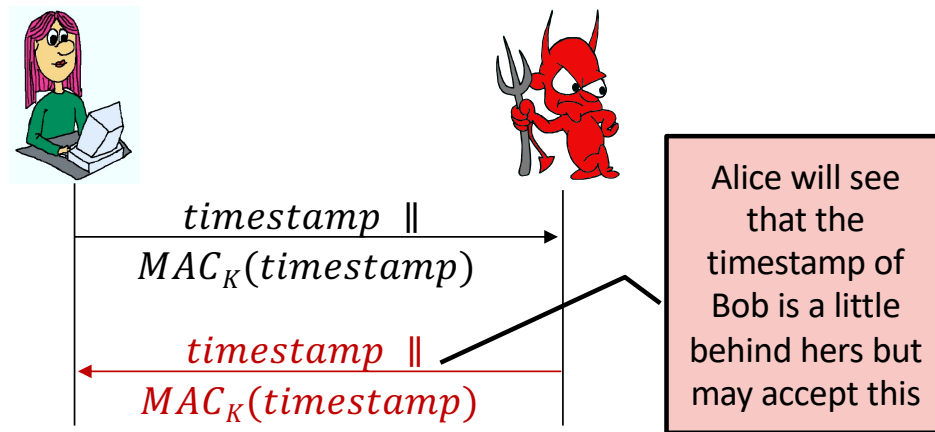
- Mutual authentication of A to B and B to A based on a shared secret key K



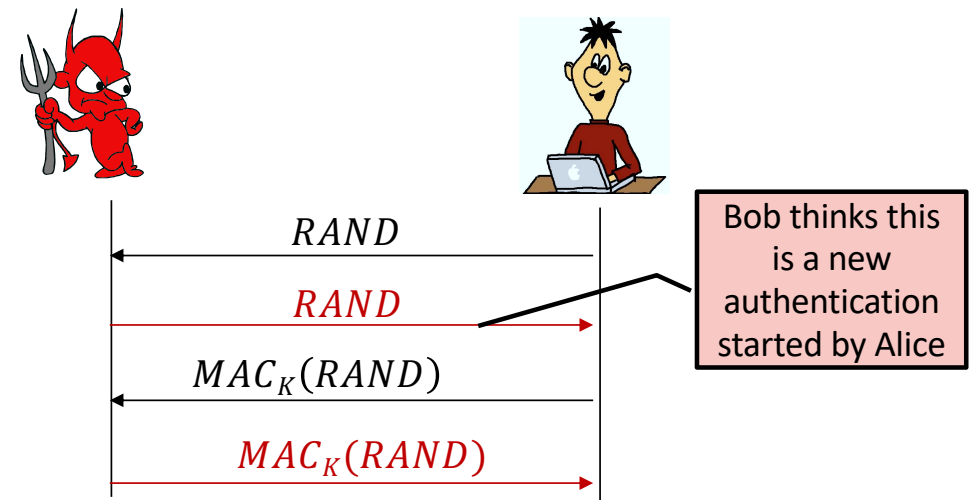
Does work with signatures just as well

Example for Insecure Building Blocks for Mutual Authentication

Simply combining the building blocks for unilateral authentication **MAY NOT be SECURE**



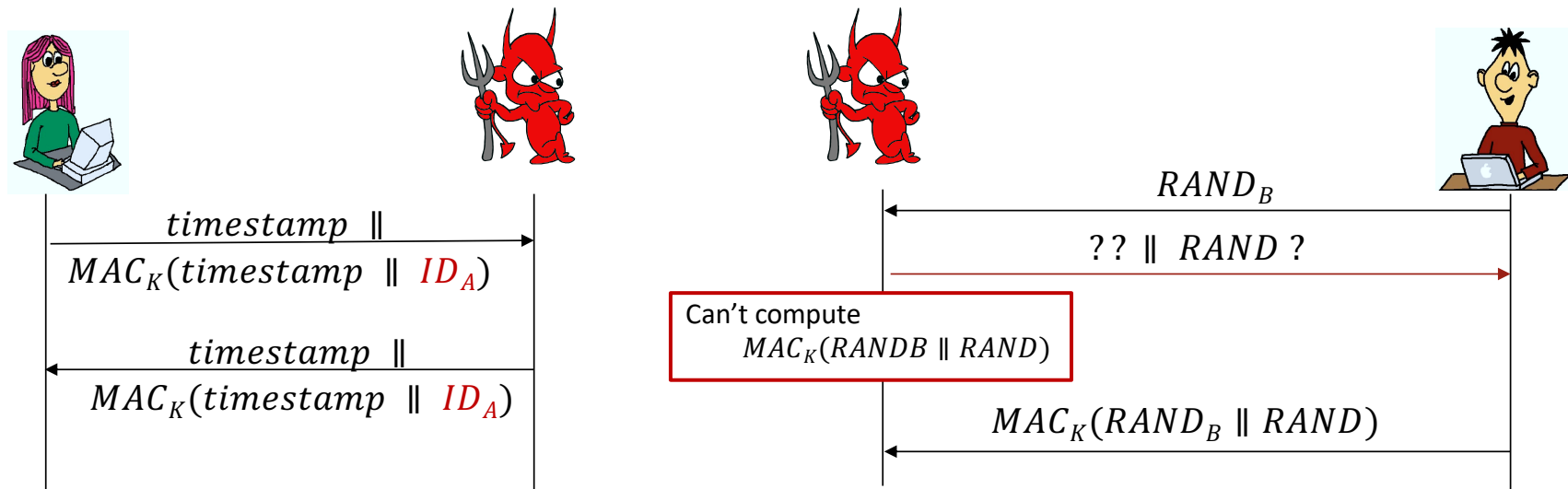
- ▶ Attacker could claim to be Bob and just reflect Alice's message to Alice
- ▶ Not impersonation resistant
- ▶ Need messages of Alice and Bob to be different



- ▶ Attacker could start a second run of the protocol by reflecting $RAND$ back to Bob
- ▶ Wait for Bob's reply
- ▶ Then reflect the MAC computed by Bob back to Bob

Protection against Reflection Attacks

- Making A and B compute MACs on different messages, where each message contains input controlled by the other part protects these building blocks from reflection attacks



- ▶ Attacker can only reflect message including Alice's ID which will be detected by Alice

- ▶ Attacker can only reflect with the random number in Bob's order not in the order expected from Alice

Overview

- **Building Blocks for Entity Authentication**

- ▶ Definition of Entity Authentication
- ▶ MAC-based authentication
- ▶ Signature-based authentication

- **Key Distribution with trusted Third Parties**

- ▶ Key Distribution Centers
- ▶ Certificates and Public Key Infrastructures

- **Authenticated Session Key Establishment**

- ▶ Definitions around session key establishment
- ▶ Authenticated Diffie Hellman variants
- ▶ Session key establishment w-o DH
- ▶ Session Key derivation principles

- **Password-based authentication**

- ▶ Password-based user authentication
- ▶ Password-based authenticated key establishment
- ▶ Dictionary attacks on password-based authentication

Entity Authentication Alone is useless!

- ▶ Authentication exchange typically only guarantees that one specific message originates from a particular entity
- ▶ If hash of previously sent messages is included, these can be authenticated as well
- ▶ But: what about future messages exchanged? And what about encryption?
- **Could keep signing messages if signatures are used**
 - ▶ Very inefficient
- **Could keep computing MACs with key K on all messages**
 - ▶ Key K would be used repeatedly on lots of traffic

Solution: Session Keys

- ▶ Establish new session keys for integrity protection and encryption
- ▶ Thus, create independence across communication sessions
- ▶ Limit amount of data protected under the same key

Session Key Establishment Protocols

A session **key establishment protocol** is a protocol

- ▶ that establishes a shared secret key between two parties

There are two types of key establishment protocols

- ▶ **Key transport protocols**
 - Key generated by one party, securely transported to the other party
- ▶ **Key agreement protocols**
 - shared key is derived from input of both parties, e.g. like in the Diffie-Hellman **key agreement** protocol

Examples

- ▶ **Simple key transport protocol**
 - Assume A and B share a long-term key K
 - A selects a session key SK
 - Computes $E_K(SK)$ and sends it to B
 - B decrypts $E_K(SK)$ with K and thus obtains SK
- ▶ **Diffie-Hellman key agreement (Chapter 4)**
 - Each party selects a random private value
 - Computes a public value based on private one
 - Parties exchange the public values
 - Each computes that key as function of own private and other party's public value

Objectives of Key Establishment Protocols

Authenticated key Establishment

- ▶ **Entity authentication (see above)**
- ▶ **Implicit key authentication:** a party is assured that no other party but a particular second party may gain access to the established key

Explicit key authentication

- ▶ Implicit key authentication
- ▶ **Key confirmation:** a party is assured that a second party has possession of the established key

Additional Objectives

- ▶ **Key freshness:** a party is assured that the key is newly generated and not a replayed old key
- ▶ **Perfect forward secrecy:** a future compromise of long-term keys does not compromise past session keys
- ▶ **Protection against known-key attacks:** the compromise of a past session key does not allow
 - a passive adversary to compromise future session keys
 - an active attacker to impersonate a party in the future

The objectives can hold for none, only one or both parties

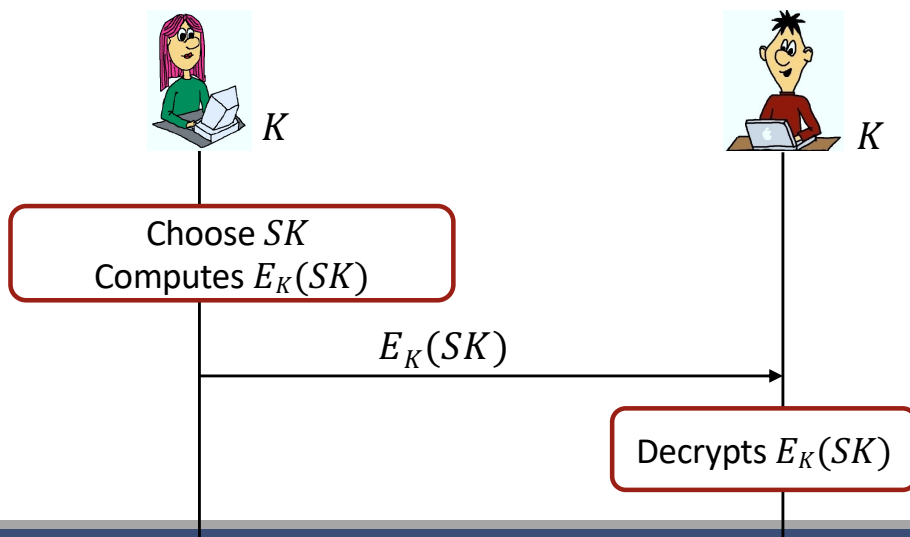
Efficiency Considerations

- **When analyzing the efficiency of protocols, we consider**
 - ▶ Number of messages exchanged between parties
 - ▶ Bandwidth required by the messages (total number of bits transmitted)
 - ▶ Complexity of computations that need to be carried out by the parties
 - ▶ Possibility for pre-computation to reduce the online load during protocol execution

Example: Simple key transport protocol

Simple key transport protocol

- ▶ Assume A and B share a long-term key K
- ▶ A selects a session key SK
- ▶ Computes $E_K(SK)$ and sends it to B
- ▶ B decrypts $E_K(SK)$ with K and thus obtains SK



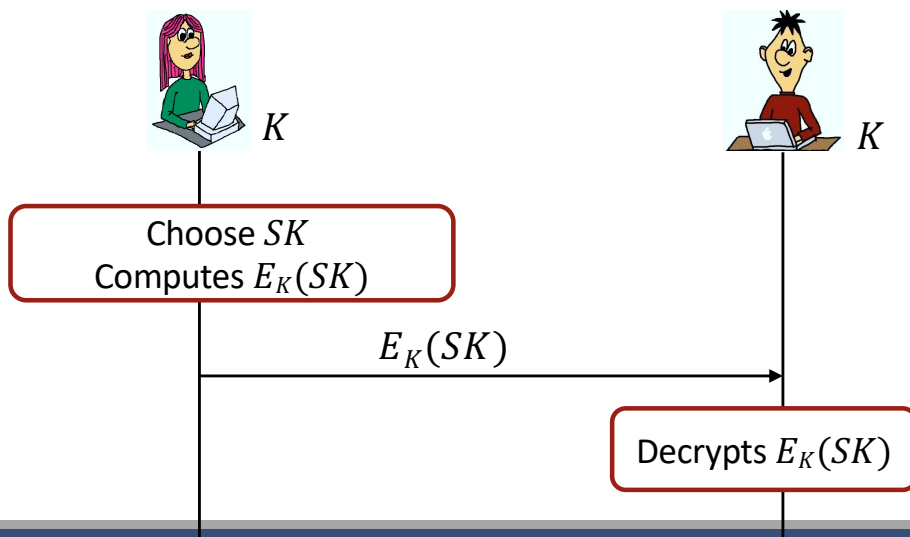
Properties

- ▶ Implicit key authentication
 - Yes, from both parties' point of view
- ▶ Key freshness
 - Yes, from A's point of view
 - No from B's point of view
- ▶ Perfect forward secrecy
 - No
- ▶ Protection against known keys
 - Past session keys have no influence on new future ones
- ▶ Authenticated key establishment
 - No! No entity authentication (replay possible)

Example: Simple key transport protocol

Simple key transport protocol

- ▶ Assume A and B share a long-term key K
- ▶ A selects a session key SK
- ▶ Computes $E_K(SK)$ and sends it to B
- ▶ B decrypts $E_K(SK)$ with K and thus obtains SK

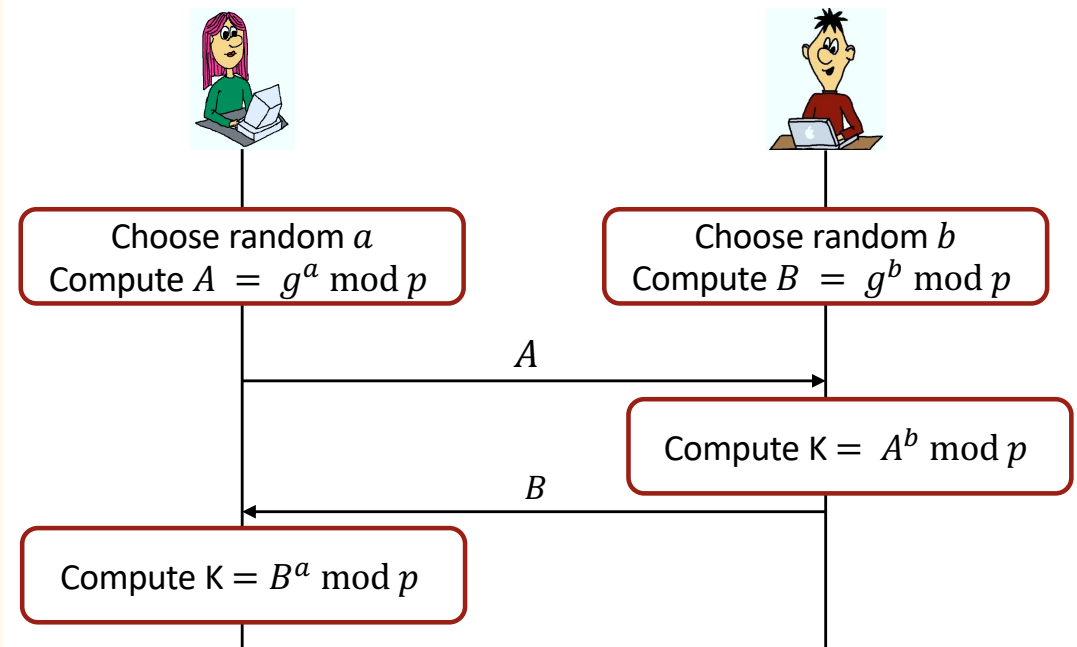


Properties

- ▶ **Implicit key authentication**
 - Yes, from both parties' point of view
- ▶ **Key freshness**
 - Yes, from A's point of view
 - No from B's point of view
- ▶ **Perfect forward secrecy**
 - No
- ▶ **Protection against known keys**
 - Past session keys have no influence on new ones
- ▶ **Authenticated key establishment**
 - No! No entity authentication (replay possible)

Diffie-Hellman Key Agreement

- ▶ Implicit key authentication
 - No
- ▶ Key freshness
 - Yes, from both parties' point of view
- ▶ Perfect forward secrecy
 - Yes, future keys completely independent
- ▶ Protection against known keys
 - Past session keys have no influence on future ones
- ▶ Authenticated key establishment
 - No! No entity authentication (replay possible), no implicit key authentication

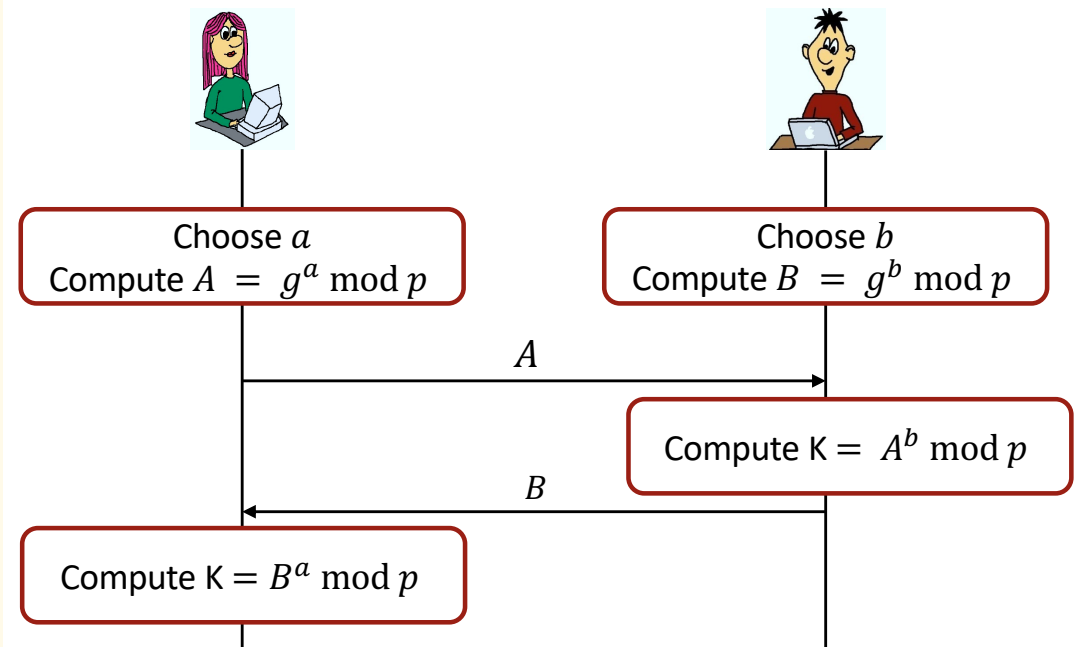


$$\text{As } A^b \text{ mod } p = g^{ab} = g^{ba} = B^a \text{ mod } p$$

Alice and Bob now share the secret key $K = g^{ab}$

Diffie-Hellman Key Agreement

- ▶ **Implicit key authentication**
 - No
- ▶ **Key freshness**
 - Yes, from both parties' point of view
- ▶ **Perfect forward secrecy**
 - Yes, future keys completely independent
- ▶ **Protection against known keys**
 - Yes, past session keys have no influence on future ones
- ▶ **Authenticated key establishment**
 - No! No entity authentication (replay possible), no implicit key authentication



$$\text{As } A^b \text{ mod } p = g^{ab} = g^{ba} = B^a \text{ mod } p$$

Alice and Bob now share the secret key $K = g^{ab}$

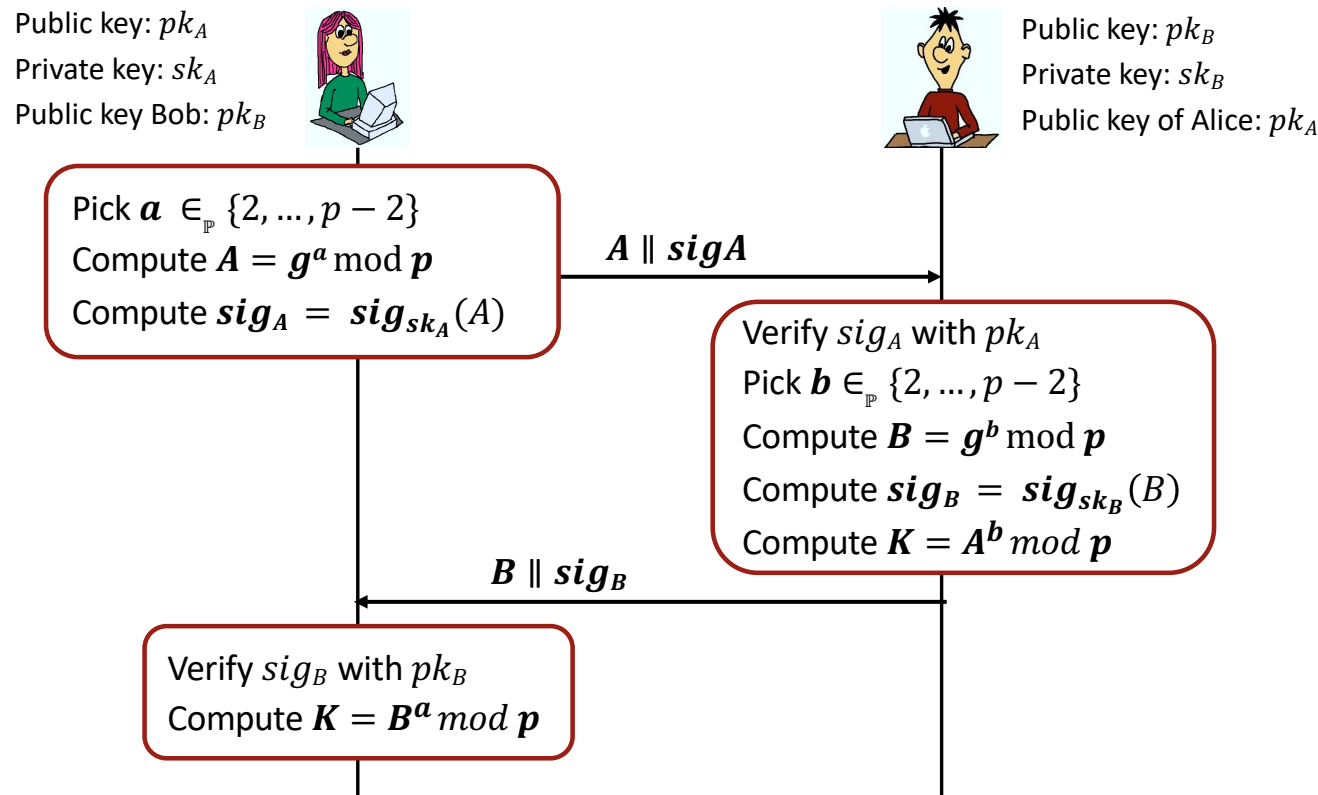
Diffie-Hellman Key Agreement with Implicit Key Authentication

• Implicit key authentication

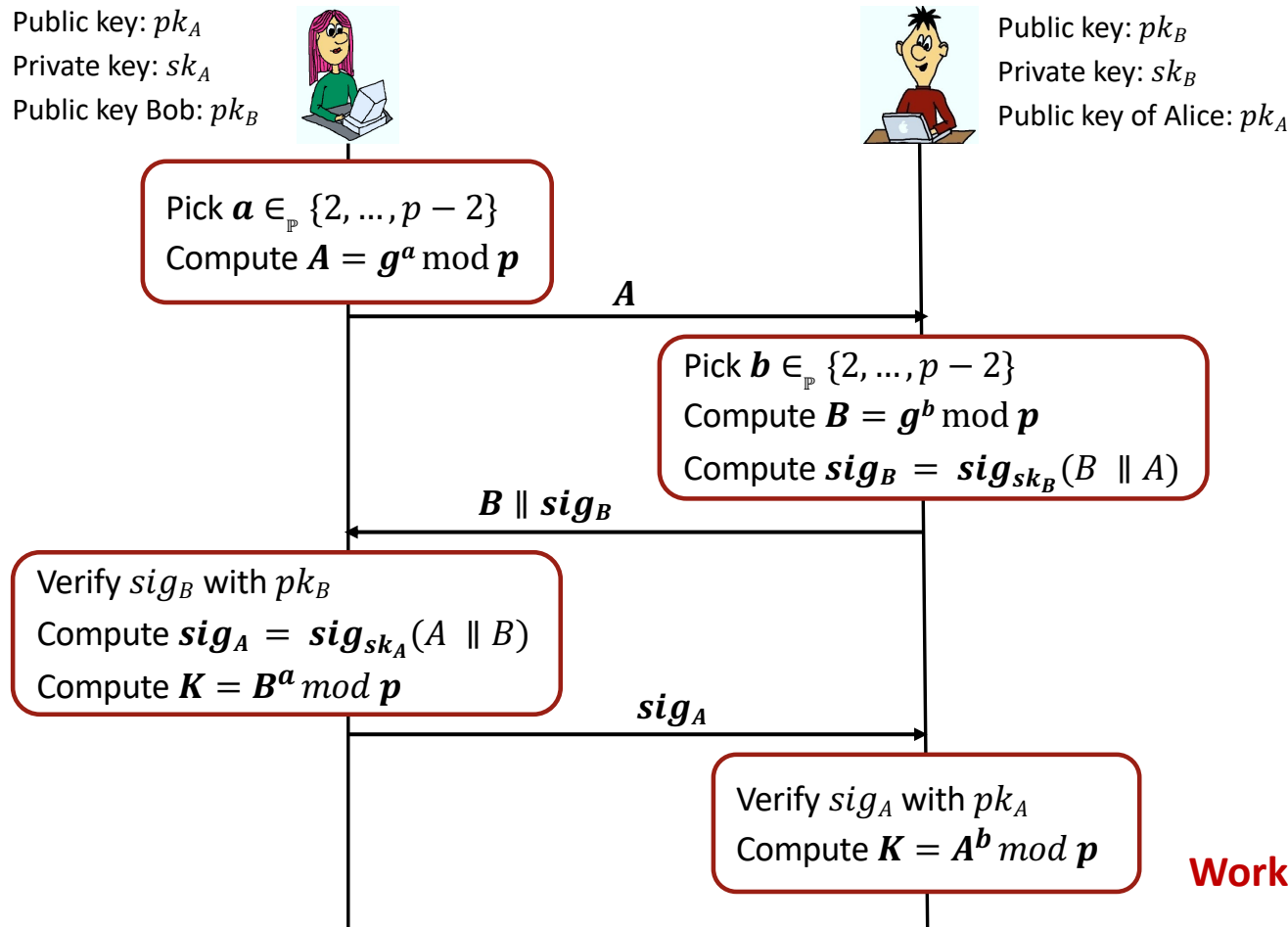
- ▶ Public DH value has been signed by the desired second party
- ▶ Only that party (if any) will be able to compute K

• But: no entity authentication

- ▶ Old messages could be replayed
- ▶ Parties do not get guarantee that other party interacts right now



Authenticated Diffie-Hellman Key Agreement with Signatures



- **Mutual authentication between Alice and Bob**

- ▶ See slide 8
- ▶ A and B act as random values here

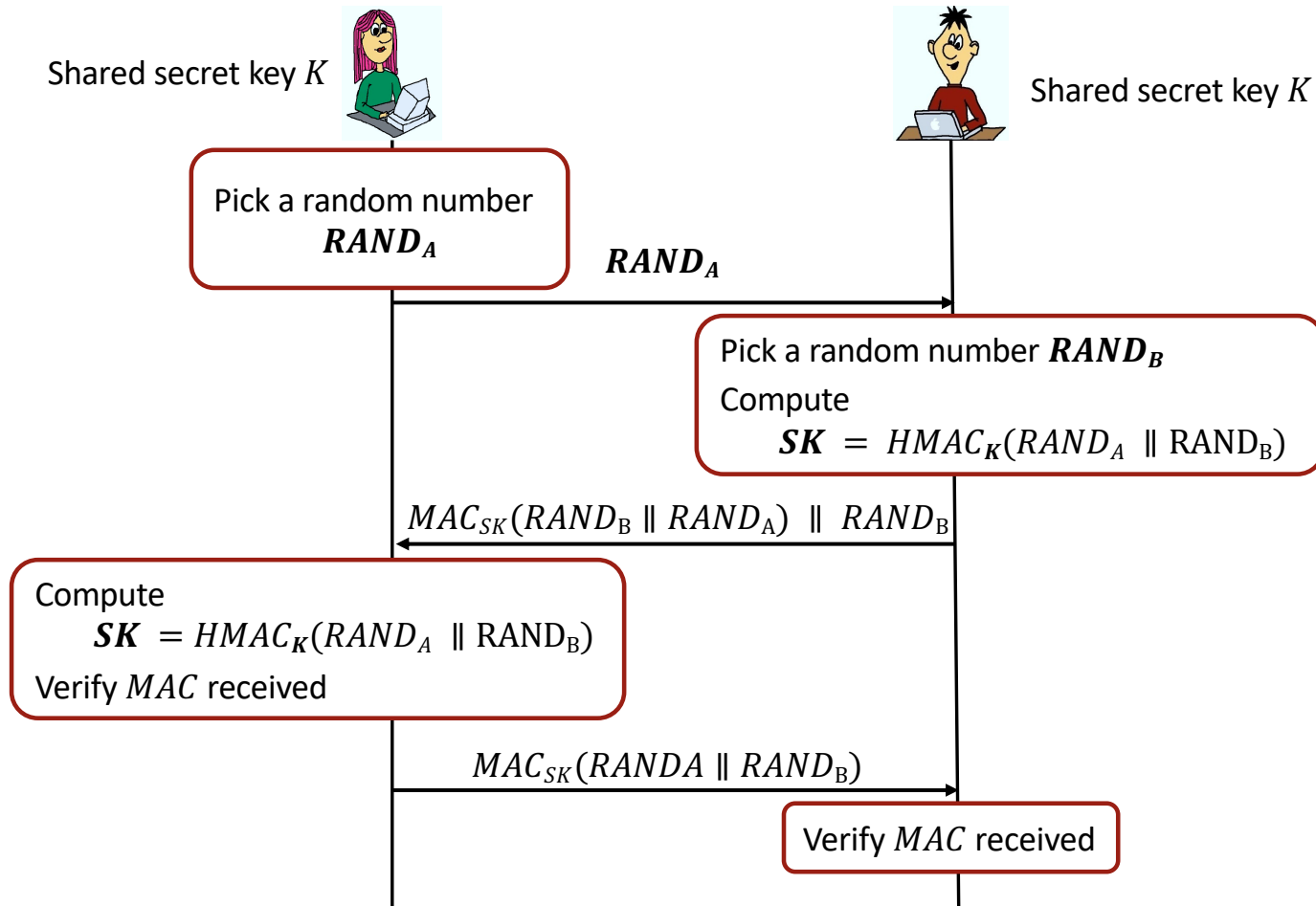
- **Implicit key authentication**

- ▶ Alice is assured that B is from B so only Bob can compute K (and herself)
- ▶ Same holds for Bob

- **Authenticated key agreement**

Works with a shared key and MACs as well!

Example Session Key Establishment without DH

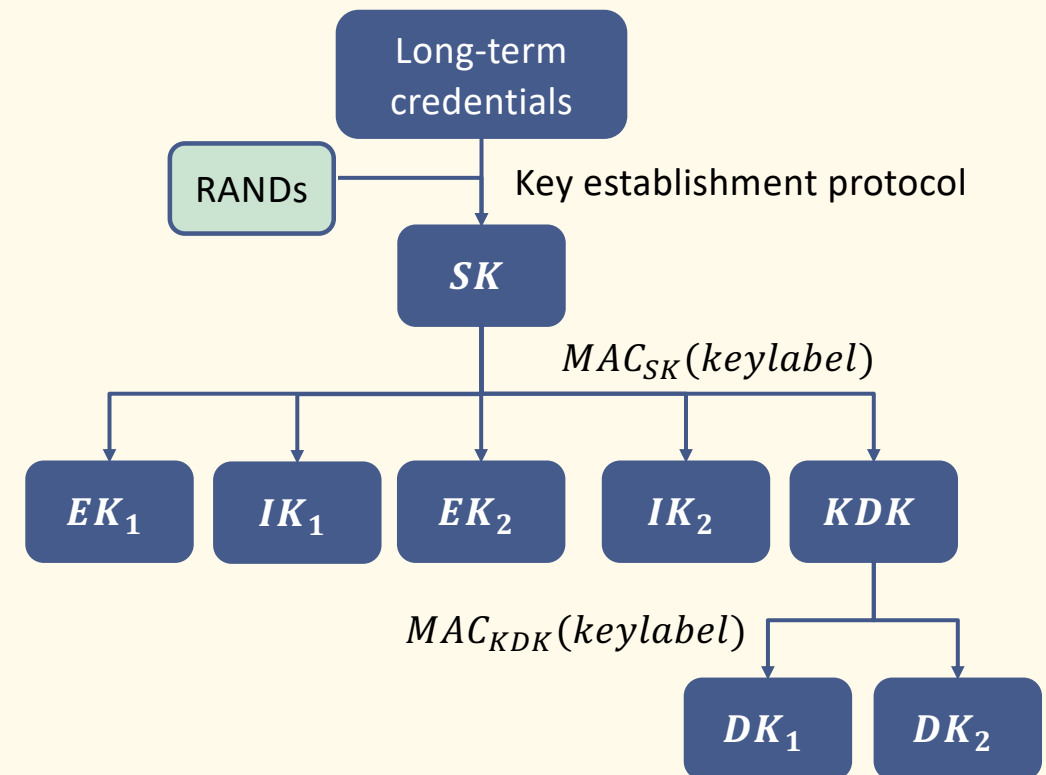


- **Implicit key authentication**
 - ▶ Yes! K required to compute SK
- **Key freshness**
 - ▶ Yes, for both parties
- **Perfect forward secrecy**
 - ▶ No! If K broken and exchange recorded, then SK broken
- **Protection against known keys**
 - ▶ Past session keys have no influence on future ones
- **Authenticated key establishment**
 - ▶ Yes!

Session Key Derivation: Key Hierarchies

- **Key establishment protocols**
 - ▶ establish a session key SK based on long term credentials and session specific random numbers
- SK often used to derive additional keys, e.g.
 - ▶ Integrity key and an encryption key
 - ▶ Different keys for different directions
 - ▶ A key derivation key for future derivations
- **Results in key hierarchy**
 - ▶ Key derivation should be efficient
 - ▶ A break of a lower layer key does not break higher layer keys or keys on the same layer

Example Hierarchy



Overview

- **Building Blocks for Entity Authentication**

- ▶ Definition of Entity Authentication
- ▶ MAC-based authentication
- ▶ Signature-based authentication

- **Key Distribution with trusted Third Parties**

- ▶ Key Distribution Centers
- ▶ Certificates and Public Key Infrastructures

- **Authenticated Session Key Establishment**

- ▶ Definitions around session key establishment
- ▶ Authenticated Diffie Hellman variants
- ▶ Session key establishment w-o DH
- ▶ Session Key derivation principles

- **Password-based authentication**

- ▶ Password-based user authentication
- ▶ Password-based authenticated key establishment
- ▶ Dictionary attacks on password-based authentication

Facilitating Key Distribution with Trusted Third Parties

Assumption so far: Alice and Bob

- ▶ Either already share a secret (long-term) key
- ▶ Or have an authentic copy of each other's public keys

Trusted Third Party

- ▶ Mediator to reduce the number of pre-installed keys required

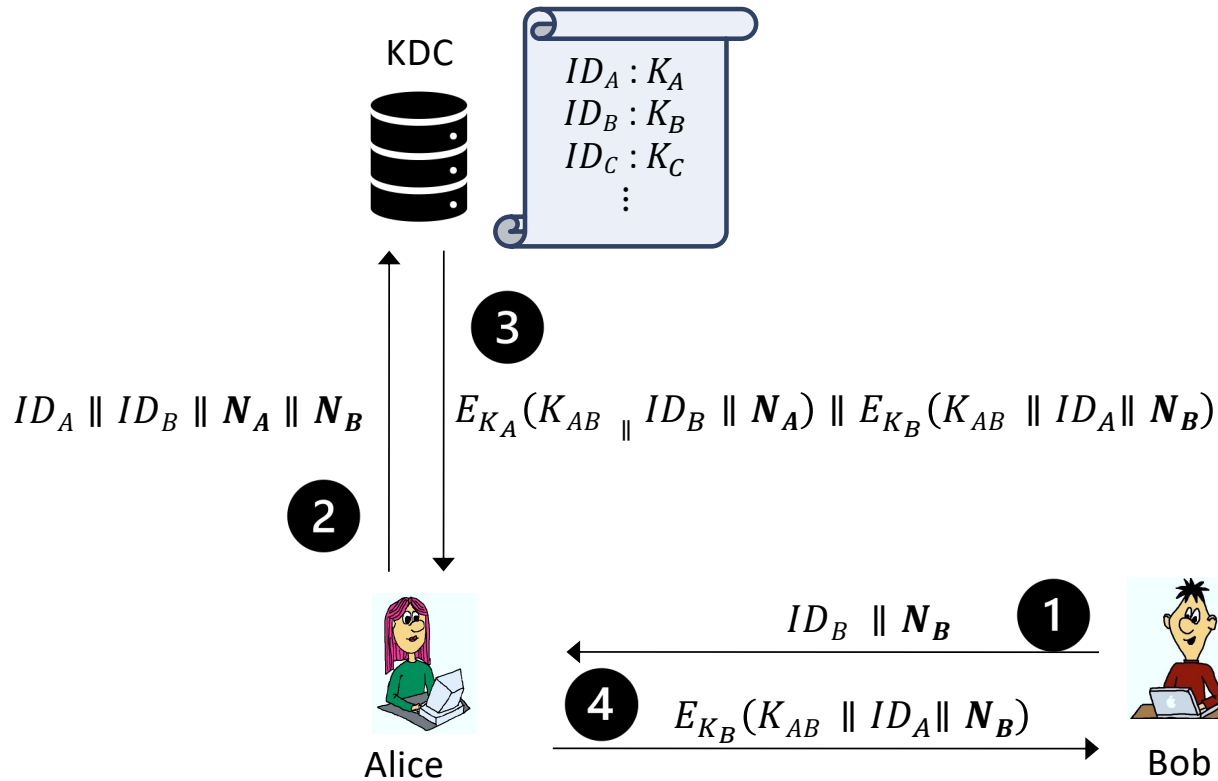
Symmetric Case: Key Distribution Centers

- ▶ Each client shares a **secret key** with the key distribution center
- ▶ The key distribution center helps to establish keys between its clients

Asymmetric Case: Certification Authorities

- ▶ Each client has the public key of a certification authority pre-installed
- ▶ The certification authority helps to distribute authentic copies of public keys

Example: Key Transport with a KDC



- KDC shares a long-term secret key K_A with Alice and K_B with Bob
- Upon request, KDC generates a session key K_{AB} for Alice and Bob
- E_K here stands for an AEAD encryption with K
- N_B and N_A authenticates KDC to Bob and Alice respectively
- Inclusion of ID_B in $E_{K_A}(K_{AB} \parallel ID_B \parallel N_A)$ gives Alice **implicit key authentication** of K_{AB}
- Inclusion of ID_A in $E_{K_B}(K_{AB} \parallel ID_A \parallel N_B)$ gives Bob **implicit key authentication** of K_{AB}
- No **perfect forward secrecy**, no **key freshness**, **protection against known key attacks**

Facilitating Key Distribution with Trusted Third Parties

Assumption so far: Alice and Bob

- ▶ Either already share a secret (long-term) key
- ▶ Or have an authentic copy of each other's public keys

Trusted Third Party

- ▶ Mediator to reduce the number of pre-installed keys required

Symmetric Case: Key Distribution Centers

- ▶ Each client shares a **secret key** with the key distribution center
- ▶ The key distribution center helps to establish keys between its clients

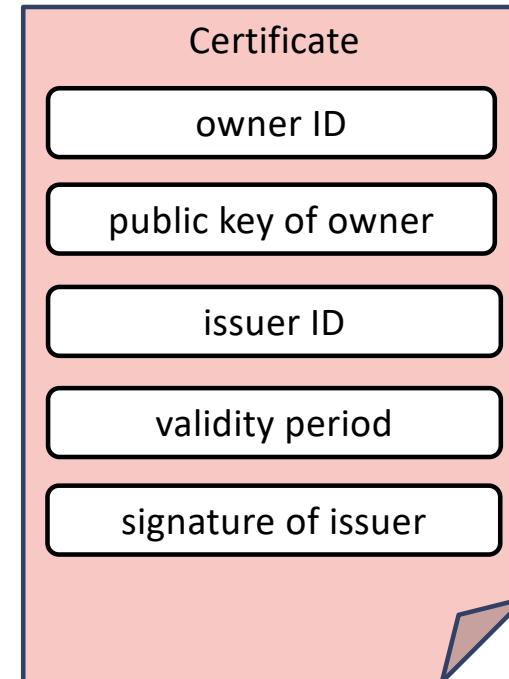
Asymmetric Case: Certification Authorities

- ▶ Each client has the public key of a certification authority pre-installed
- ▶ The certification authority helps to distribute authentic copies of public keys

Certification Authorities and Public Key Infrastructures

- **Certification Authority**

- ▶ Signs a certificate for each of its clients
- ▶ Certificate
 - owner ID: identifier of the owner of the public key
 - public key of owner
 - issuer ID: identifier for the CA that issued the certificate
 - Validity period: not before, until dates defining when this certificate becomes valid and when it expires
 - Signature of the issuing CA on all of the content of the certificate, binds public key to owner ID

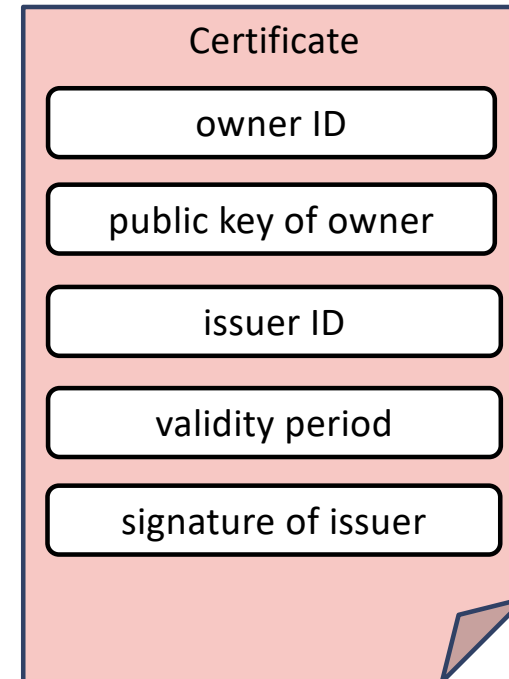


- **Anyone in possession of the public key of the CA**

- ▶ Can verify the **authenticity of the public key** of the owner

Certificate Verification

- **Anyone in possession of the public key of the CA**
 - ▶ Can verify the **authenticity of the public key** of the owner
- **Certificate verification entails**
 - ▶ checking the validity period of the certificate
 - ▶ checking that the owner ID is as expected
 - E.g., in the context of web does the domain name included as identifier in the certificate match the host name part of the URL of the visited website
 - ▶ checking the signature on the certificate with the public key of the issuer
 - ▶ checking the revocation status of the certificate



Certificate Revocation Approaches

Certificates may need to be revoked before they expire

- ▶ Due to stolen devices, precaution after malware infection,...
- ▶ Due to lost passwords unlocking private keys

Certificate revocation lists = CRLs

- ▶ Issuing CA periodically publishes a signed CRL
- ▶ CRL includes serial numbers of all revoked unexpired certificates
- ▶ Disadvantage: revocation only as timely as period used to publish CRLs

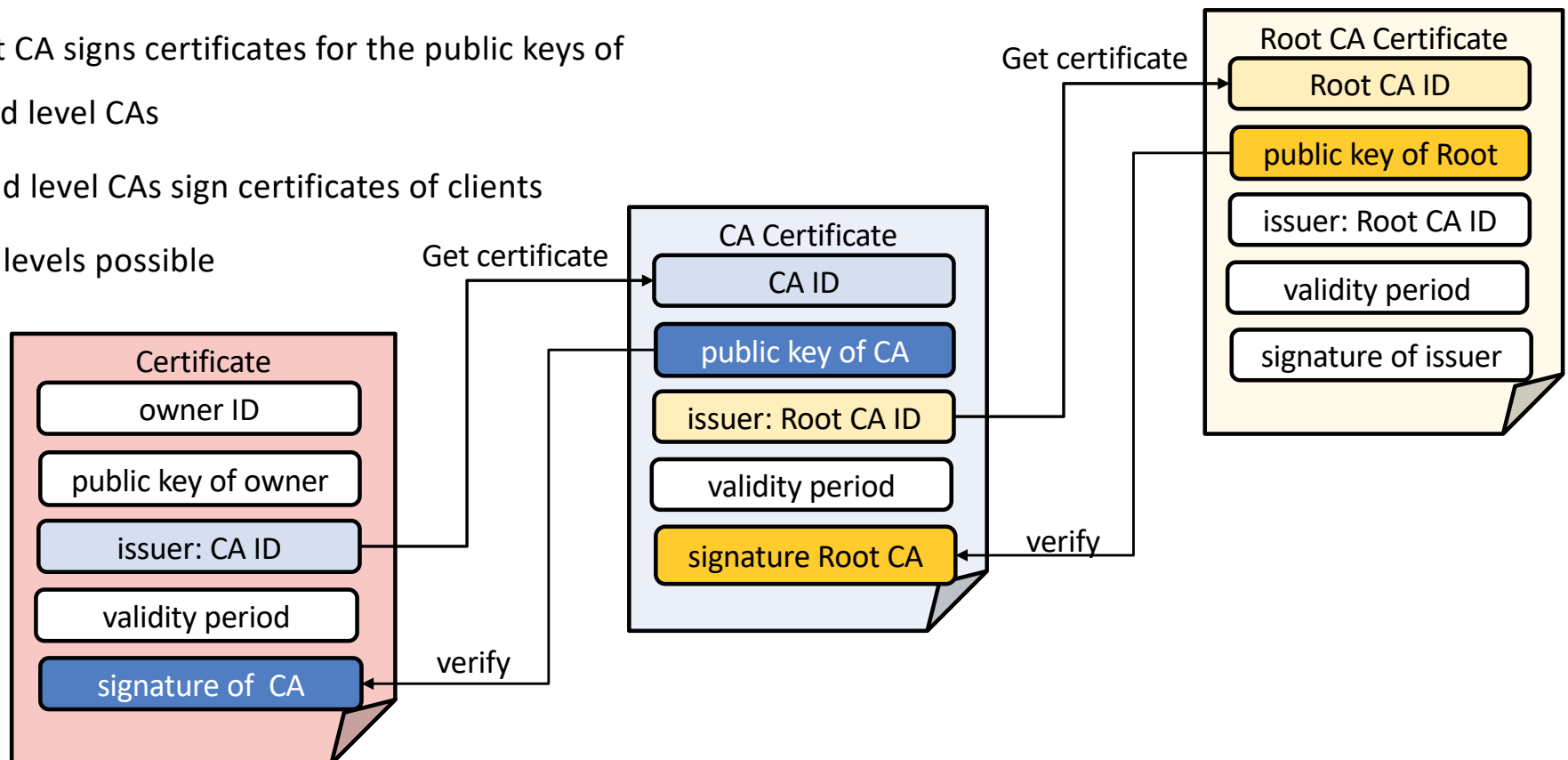
Online Certificate Status Protocol = OSCP

- ▶ Protocol to obtain immediate feedback on the revocation status of certificates
- ▶ Advantage: very timely revocation possible
- ▶ May add additional overhead and requires connectivity to the OSCP server

Chains of Certificates

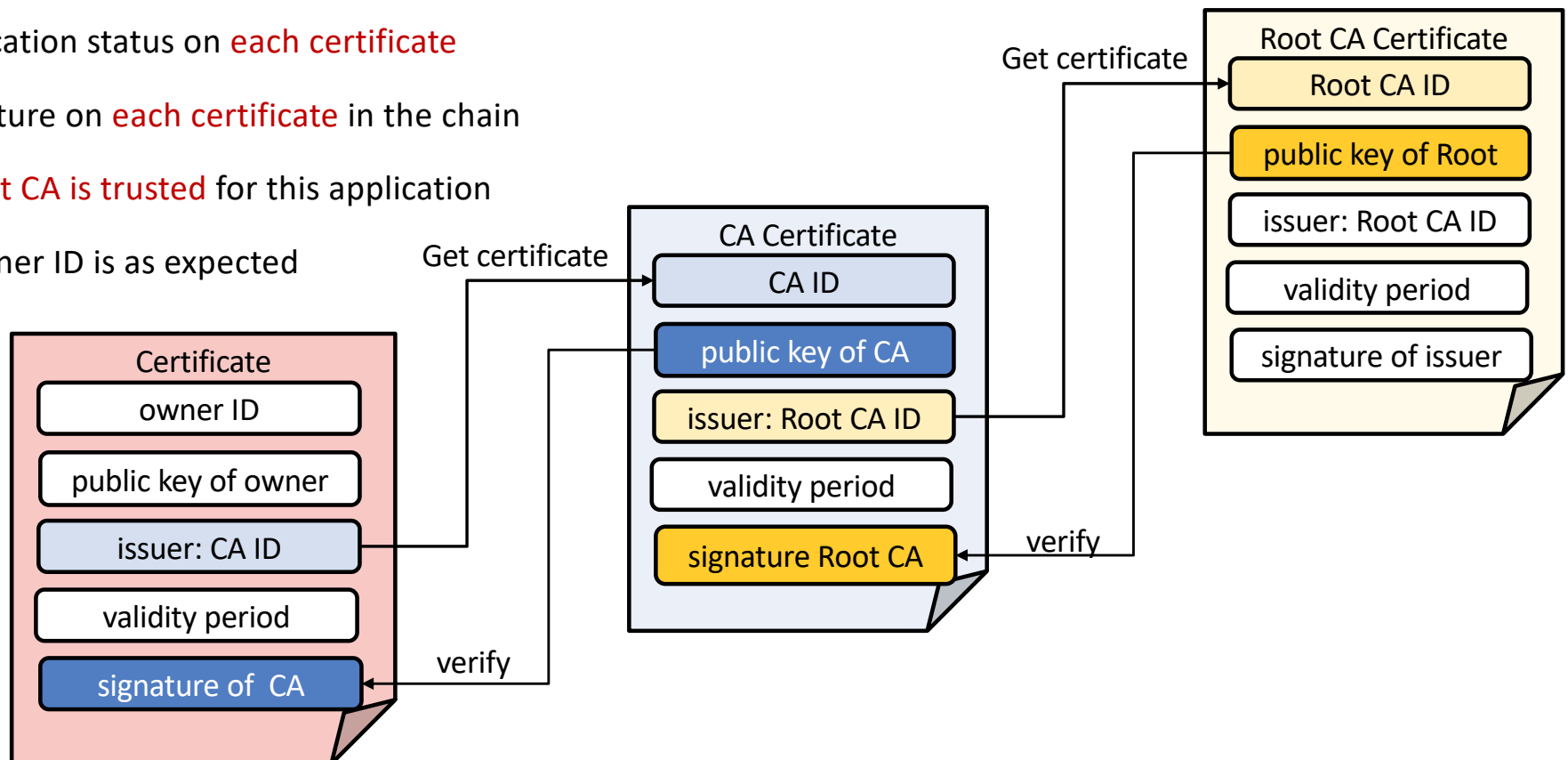
- Hierarchies of certification authorities

- ▶ A root CA signs certificates for the public keys of second level CAs
- ▶ Second level CAs sign certificates of clients
- ▶ More levels possible



Verifying Chains of Certificates

- Check validity period of **each certificate**
- Check revocation status on **each certificate**
- Verify signature on **each certificate** in the chain
- Check if **root CA is trusted** for this application
- Check if owner ID is as expected



Example Secure Authenticated DH with Chain of Certificates

Public key: pk_A
 Private key: sk_A
 Public key of root CA: pk_{root}



Public key: pk_B
 Private key: sk_B
 Public key of root CA: pk_{root}



Pick $a \in_{\mathbb{R}} \{2, \dots, p-2\}$
 Compute $A = g^a \text{ mod } p$

A

Pick $b \in_{\mathbb{R}} \{2, \dots, p-2\}$
 Compute $B = g^b \text{ mod } p$
 Compute $sig_B = sig_{sk_B}(B \parallel A)$

$B \parallel sig_B \parallel cert_B$

Verify chain of certificates $cert_B$
 Verify sig_B with pk_B extracted from B's certificate
 Compute $sig_A = sig_{sk_A}(A \parallel B)$
 Compute $K = B^a \text{ mod } p$

$sig_A \parallel cert_A$

Verify chain of certificates $cert_A$
 Verify sig_A with pk_A extracted from A's certificate
 Verify sig_A with pk_A
 Compute $K = A^b \text{ mod } p$

- $cert_B / cert_A$: chain of certificates starting with a certificate for A / B, where the last one is the root certificate

Overview

- **Building Blocks for Entity Authentication**

- ▶ Definition of Entity Authentication
- ▶ MAC-based authentication
- ▶ Signature-based authentication

- **Key Distribution with trusted Third Parties**

- ▶ Key Distribution Centers
- ▶ Certificates and Public Key Infrastructures

- **Authenticated Session Key Establishment**

- ▶ Definitions around session key establishment
- ▶ Authenticated Diffie Hellman variants
- ▶ Session key establishment w-o DH
- ▶ Session Key derivation principles

- **Password-based authentication**

- ▶ Password-based user authentication
- ▶ Password-based authenticated key establishment
- ▶ Dictionary attacks on password-based authentication

Password-based Authentication

Three main flavors used in practice

- Certificate-based server authentication
- Password-based user authentication
- **Vulnerable to dictionary attacks if password file stolen**

Used, e.g., in
HTTPs



- MAC-based authenticated key exchange
- MAC-key derived from password
- **Vulnerable to dictionary attacks**

Used, e.g., in 4-
Way-Handshake
in WPA2 WLAN



- Password-Authenticated Diffie Hellman
- **Protected against Dictionary attacks**
- Same (one-time) password entered on both devices

Used, e.g., Secure
Authentication of
Equals in WPA3



Password-based Authentication

Three main flavors used in practice

- Certificate-based server authentication
- Password-based user authentication
- **Vulnerable to dictionary attacks if password file stolen**

Used, e.g., in
HTTPs



- MAC-based authenticated key exchange
- MAC-key derived from password
- **Vulnerable to dictionary attacks**

Used, e.g., in 4-
Way-Handshake
in WPA2 WLAN



- Password-Authenticated Diffie Hellman
- Protected against Dictionary attacks
- Same password entered on both devices

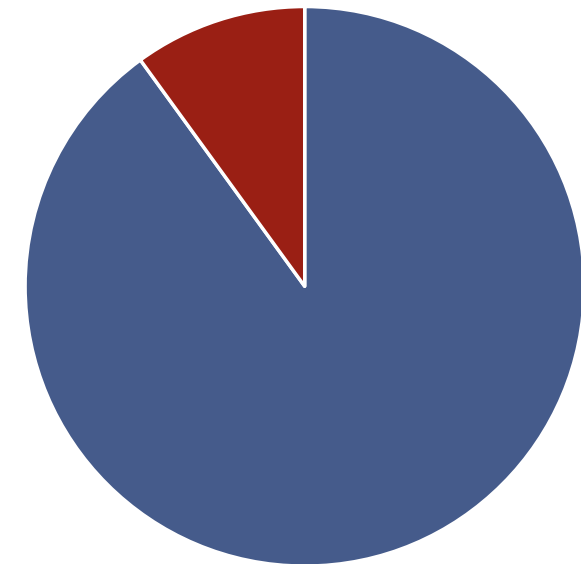
Used, e.g., Secure
Authentication of
Equals in WPA3

Advance
ITSec
Lecture

Password Length and Bit-Equivalence

- **Assume users can chose n character passwords**
 - ▶ small letters = 26 and capital letters = 26
 - ▶ numbers = 10, special characters except for space = 32
- **Then there are 94^n theoretically possible passwords**
 - ▶ $n = 8 \Rightarrow \approx 2^{52}$ possible passwords \triangleq random secret key of 52 bit
 - ▶ $n = 16 \Rightarrow \approx 2^{104}$ possible passwords \triangleq random secret key of 104 bit
- **Users tend NOT to select passwords randomly!**
 - ▶ Mainly because they cannot remember random passwords longer than 8 characters
 - ▶ And on average only one of these

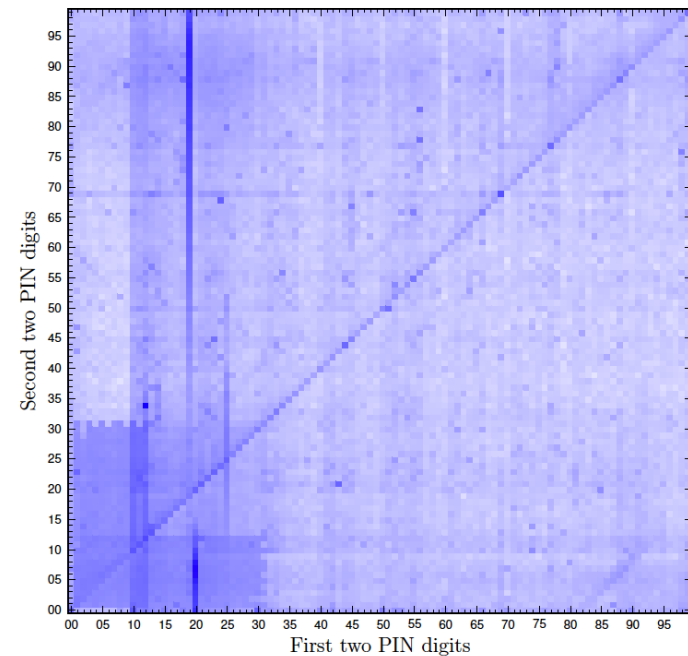
User-selected Passwords



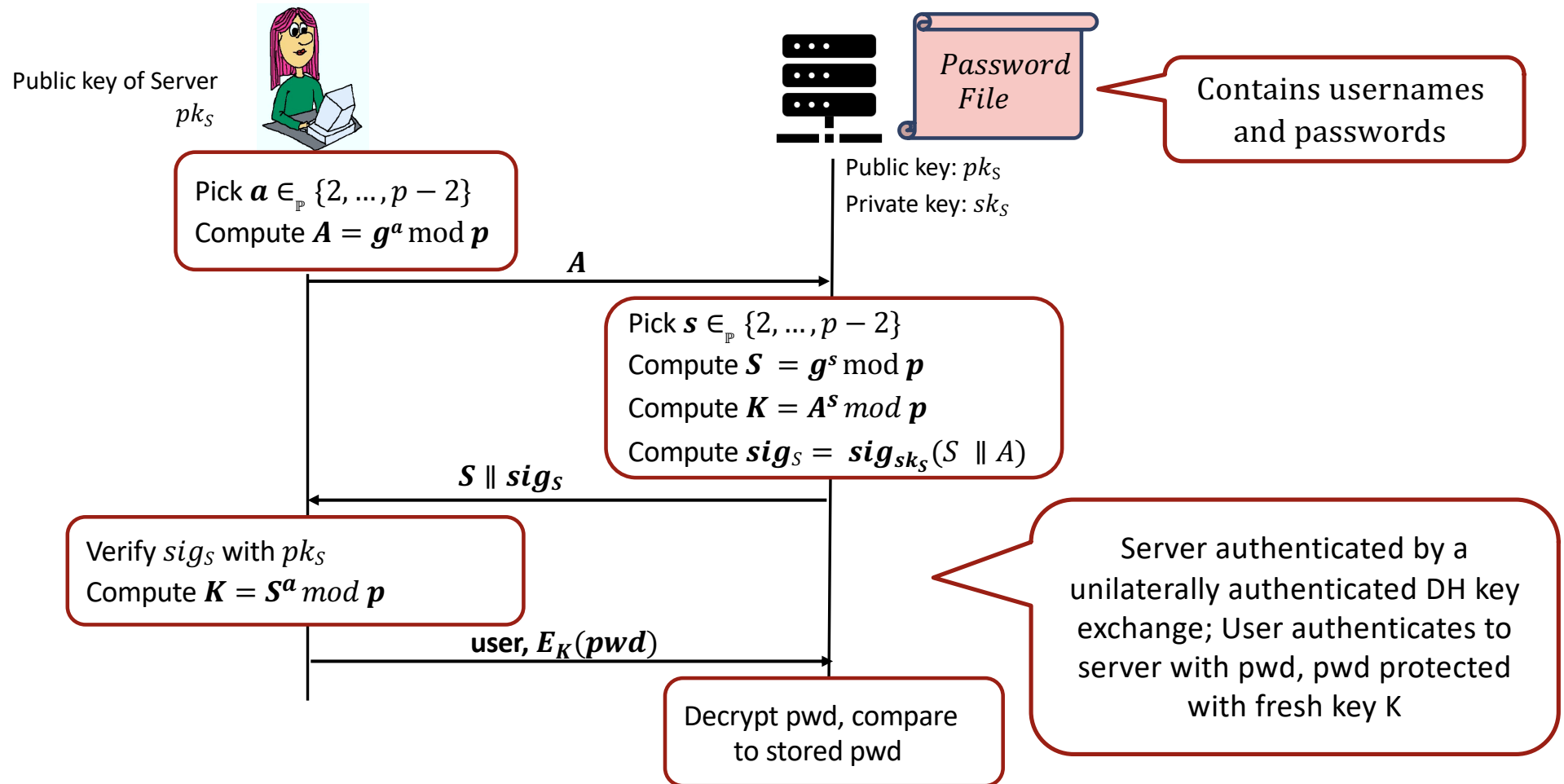
■ Potential Passwords ■ Passwords Selected by Users

Classic Example: User's self-selected Banking PINs 2012

- **Distribution of 4-Digit PINs in a data base of 32 Million Banking PINS**
- **Enforcing rules on the password selection reduces the overall number of possible passwords**
 - ▶ E.g., if 8 characters are used and at least one of them needs to be an upper case letter, one a lower case letter one a number and one a special character
 - ▶ Longer passwords required
- **General recommendation**
 - ▶ Use random passwords and a password manager



Password-based User and Certificate-based Server Authentication



Storing Passwords in Password Files (1)

In the clear?

- ▶ If attacker gains access to the file, break is immediate

User	<i>pwd</i>
Alice	D^6as\$%kjahG
Bob	(*&)A8a;sdifh

Encrypted?

- ▶ No immediate access
- ▶ But: encryption key needs to be stored somewhere
- ▶ Decryption adds overhead

User	<i>pwd</i>	$E_K(pwd)$
Alice	D^6as\$%kjahG	Svl0EKlmp76XcePiC+wL7g
Bob	(*&)A8a;sdifh	1YE/i6MU4lBEnmbq/Wn1Zw

Key

a57987a344d32336

Storing Passwords in Password Files (2)

Store $h(pwd)$ using a cryptographic hash function

- ▶ Attacker only learns hashes from file
- ▶ Cannot compute pre-images of the hashes
- ▶ But: what if multiple users use same pwd?

Better: store random salt and $h(pwd \parallel salt)$

- ▶ Now users using the same passwords will have different hashes

User	<i>pwd</i>	salt	SHA256
Alice	D^6as\$%kjahG		c25559cad0aca1566d4ba7609759e2de824c8af9e1e0b27891e99ac495e77877
Bob	(*&)A8a;sdifh		f69f1260b38daf282d8d729df34e40c0bdf0fb634f72fe7c17b09054d96c5724
Clare	(*&)A8a;sdifh		f69f1260b38daf282d8d729df34e40c0bdf0fb634f72fe7c17b09054d96c5724
Alice	D^6as\$%kjahG	(*daw	3bcc5a93e5510780f3ce13b8f673758cee1e246963be321ced2d6f2d74054558
Bob	(*&)A8a;sdifh	&OGa8	373d0dd007c4409bdc5a05e6174e5322e88cc16d736d71c99a8876f01c70a9d9
Clare	(*&)A8a;sdifh	6YY34	5ee7d56e09d86f7d262fc0d68f27861644252c1dbd80cb59bbd6cedf6c080831

Dictionary Attacks on Password Files

- **Dictionary**

- ▶ List of commonly used passwords

- **Dictionary attack**

- ▶ Try out all passwords in the dictionary

Attack on a stolen password files w/o salts

- ▶ Pre-compute $h(pwd)$ for any pwd in the dictionary
- ▶ Compare computed hashes with stored ones

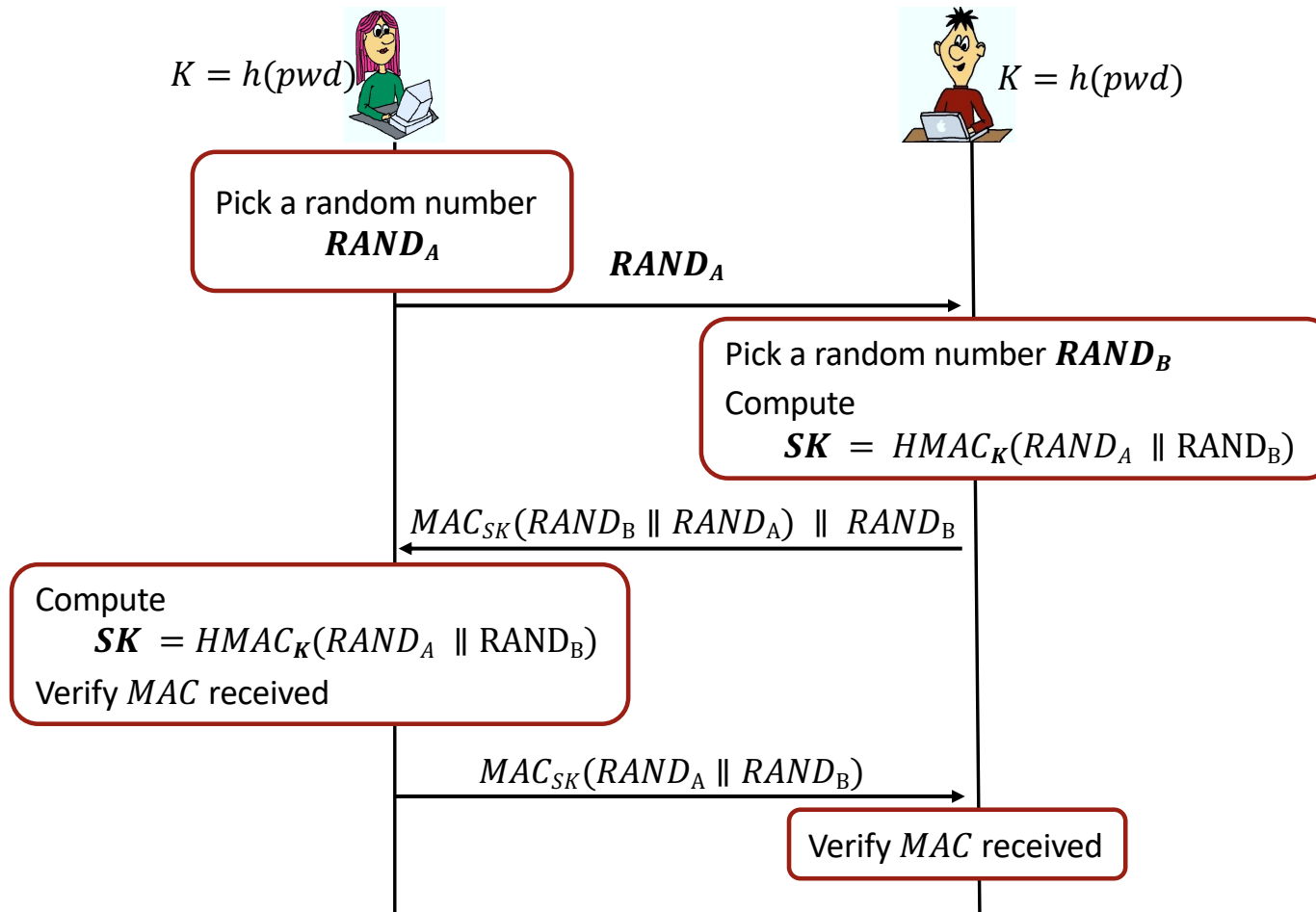
Attack on a stolen password file with salts

- ▶ Compute $h(pwd \parallel salt)$ for any $salt$ in the password file and any pwd in the dictionary
- ▶ Compare computed hashes with stored ones

Salts are pwd-file specific

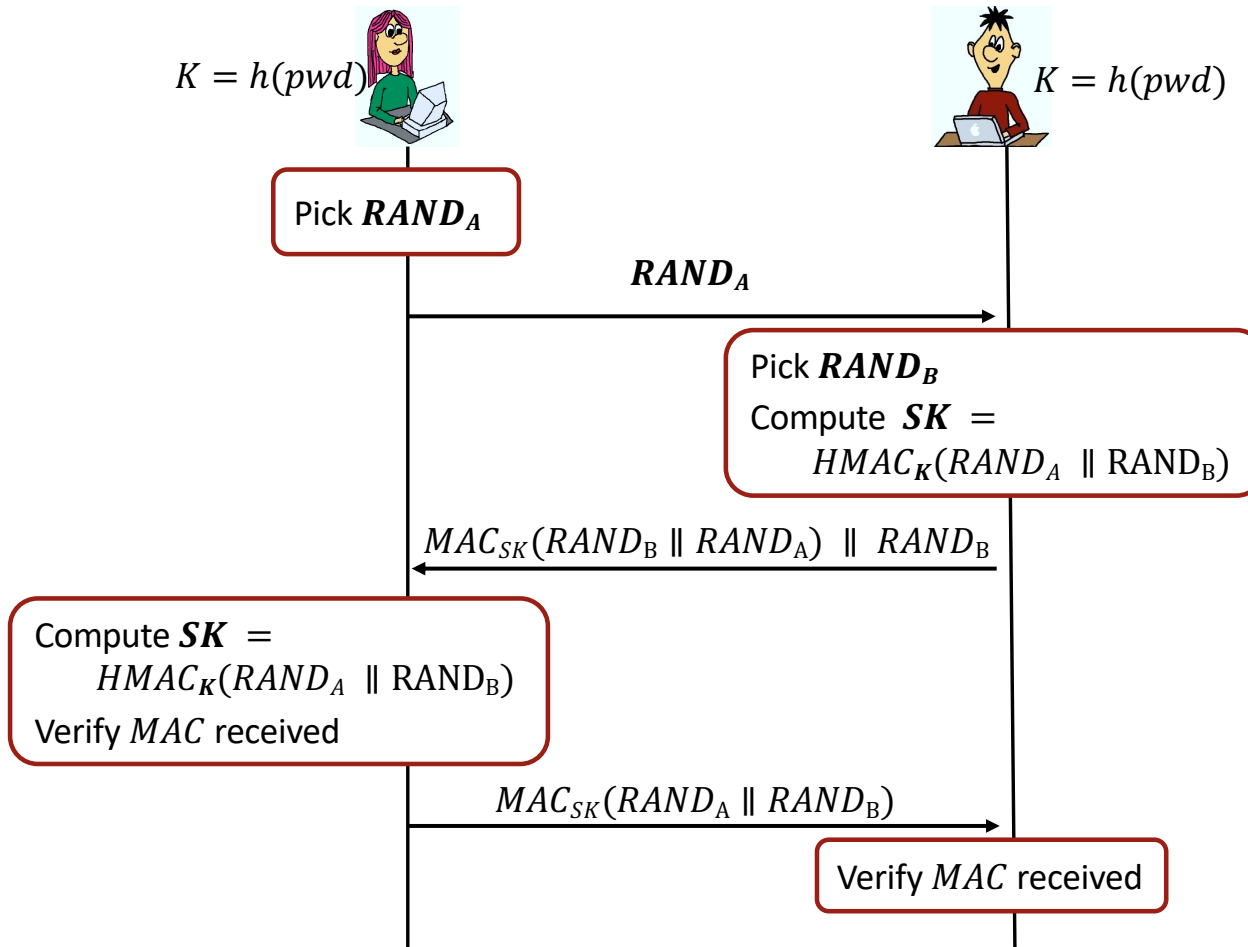
Needs to be done only once

Authentication and Key Agreement with Password-Generated MAC Keys



- Secret key generated by a shared password
- Note that the key is only as strong as the password
 - ▶ K will be 128 bit but will be as easily guessable as the pwd

Dictionary Attack on Password-Authenticated Key Agreement



- Record the message flow

- ▶ $RAND_A, RAND_B$

- ▶ $MAC_{SK}(RAND_B \parallel RAND_A)$

- For **pwd** in the dictionary

- ▶ compute $K = h(pwd)$

- ▶ compute SK from recorded RANs

- ▶ Check if

$MAC_{SK}(RAND_B \parallel RAND_A) =$

$MAC_{SK}(RAND_B \parallel RAND_A)$ recorded

- ▶ If yes: **pwd** = pwd

- ▶ Else: try next **pwd** in dictionary

Summary

- **Entity authentication requires**

- ▶ an unforgeable proof that the other entity is active in the current protocol
- ▶ session key establishment
 - Ensures continuous authentication of the authenticated entity

- **Entity authentication can be**

- ▶ unilateral or mutual
- ▶ be based on
 - secret keys using message authentication codes
 - or public/private key pairs

- **Key Establishment protocols**

- ▶ can be key agreement or key transport protocols

Summary

Potential properties of key establishment protocols

- explicit key authentication
- ▶ entity authentication
 - ▶ implicit key authentication
 - ▶ key confirmation
 - ▶ key freshness
 - ▶ perfect forward secrecy
 - ▶ protection against known key attacks
- authenticated key establishment
-

Summary

- **Trusted third parties can help to**

- ▶ reduce the amount of pre-stored keys that need to be exchanged
- ▶ Key distribution centers are TTPs that
 - help their clients establish symmetric keys
- ▶ CAs are TTP that
 - help to distribute authentic copies of their clients' public keys

- **End-users are often authenticated with the help of passwords**

- ▶ The larger the alphabet and the longer the password the stronger the password is

- **End-users tend to pick specific passwords more often than others**

- ▶ Can compile a dictionary of often picked passwords

References

- **W. Stallings, Cryptography and Network Security: Principles and Practice, 8th edition, Pearson 2022**
 - ▶ Chapter 15: Cryptographic Key Management and Distribution
 - ▶ Chapter 16: User Authentication
- RFC 5869 HMAC-based Extract-and-Expand Key Derivation Function (HKDF)



IT-Security

Chapter 6: Network Security Protocols on Network and Transport Layer

Prof. Dr.-Ing. Ulrike Meyer



Overall Lecture Context

- **In the past lectures we have learned how to**
 - ▶ Protect confidentiality with symmetric or asymmetric encryption
 - ▶ Protect integrity (including replay) with MACs or digital signatures
 - ▶ Establish session keys between authenticated entities
- **In this chapter we will learn how these mechanisms are used in network security protocols**
- **In particular, we will study and compare IPSec, and TLS**

Overview

IPSec

- ▶ Primary use cases
- ▶ Security services offered
- ▶ Authentication and key agreement
- ▶ IP Payload of IP packet protection

TLS

- ▶ Primary use case
- ▶ Security services offered
- ▶ Authentication and key agreement
- ▶ TCP payload protection

Comparison of the protocols

- ▶ Differences
- ▶ Communalities in mechanisms used
- ▶ Overlaps in use cases

Overview

IPSec

- ▶ Main use case
- ▶ Security services offered
- ▶ Authentication and key agreement
- ▶ Payload or packet protection

TLS

- ▶ Main use case
- ▶ Security services offered
- ▶ Authentication and key agreement
- ▶ Payload protection

Comparison of the protocols

- ▶ Differences
- ▶ Communalities in mechanisms used
- ▶ Overlaps in use cases

Overview IPSec Part

Introduction

- ▶ Historical notes
- ▶ Security services offered by IPSec
- ▶ Transport Mode and Tunnel Mode
- ▶ Primary Use Cases

Encapsulating Security Payload Protocol ESP

- ▶ Encryption and Integrity Protection
- ▶ ESP Header
- ▶ MAC computation

Authentication Header Protocol AH

- ▶ Integrity Protection in the two modes
- ▶ ESP Header
- ▶ MAC computation
- ▶ Supported algorithms in AH and ESP
- ▶ Replay protection in AH and ESP

Authentication and Key Agreement with IKEv2

- ▶ The concept of security associations
- ▶ Overview on detailed discussion of IKEv2
- ▶ IP packet processing with IPSec
- ▶ Example use cases

IPsec over the Years

- **IPsec is a protocol family**
- **Originally comprising**
 - ▶ ISAKMP for transporting key management messages
 - ▶ IKEv1 for authenticated key agreement carried over ISAKMP
 - ▶ ESP/AH protocol for encryption and integrity protection
- **Recommended today**
 - ▶ IKEv2 for authentication and key agreement
 - ▶ ESP/AH protocol for encryption and integrity protection
- **We focus on the latest versions of these protocols**

Security Services offered by IPsec

- **Authenticated Session Key Exchange**

- ▶ Using the **Internet Key Exchange Protocol**
- ▶ Based on **pre-shared keys** or based on **certificates**

- **IP packet level encryption and/or IP packet level integrity protection**

- ▶ Including replay protection
- ▶ Using the **Encapsulating Security Payload Protocol**
- ▶ And/or using the **Authentication Header Protocol**

- ▶ **Transport mode**

- Protection of IP payload of all IP packets exchanged between two IPsec-enabled hosts

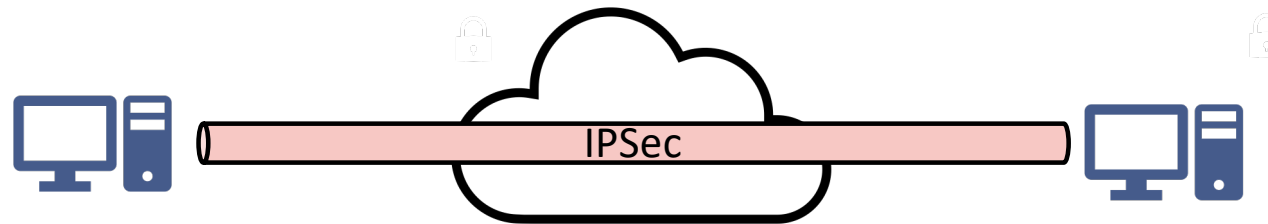
- ▶ **Tunnel mode**

- Protection of complete IP packets routed between IPsec-enabled gateways

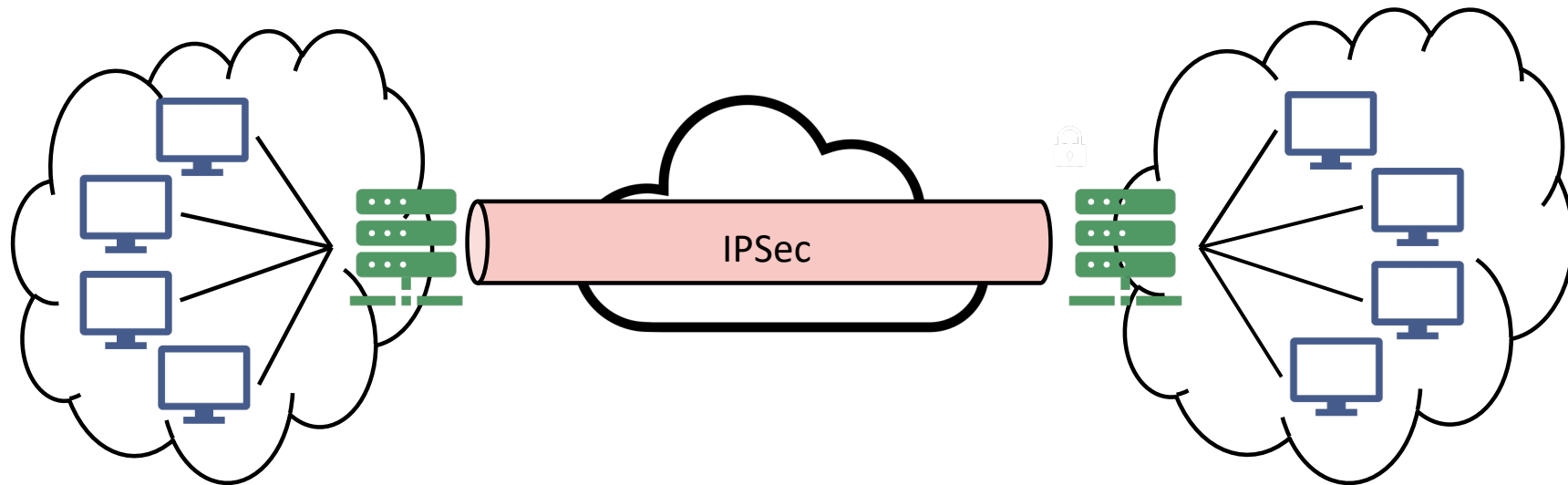
Usable on top of IPv4 and IPv6
Transparent to higher layer protocols

Tunnel Mode and Transport Mode and Primary Use Cases

Transport mode between any two individual nodes

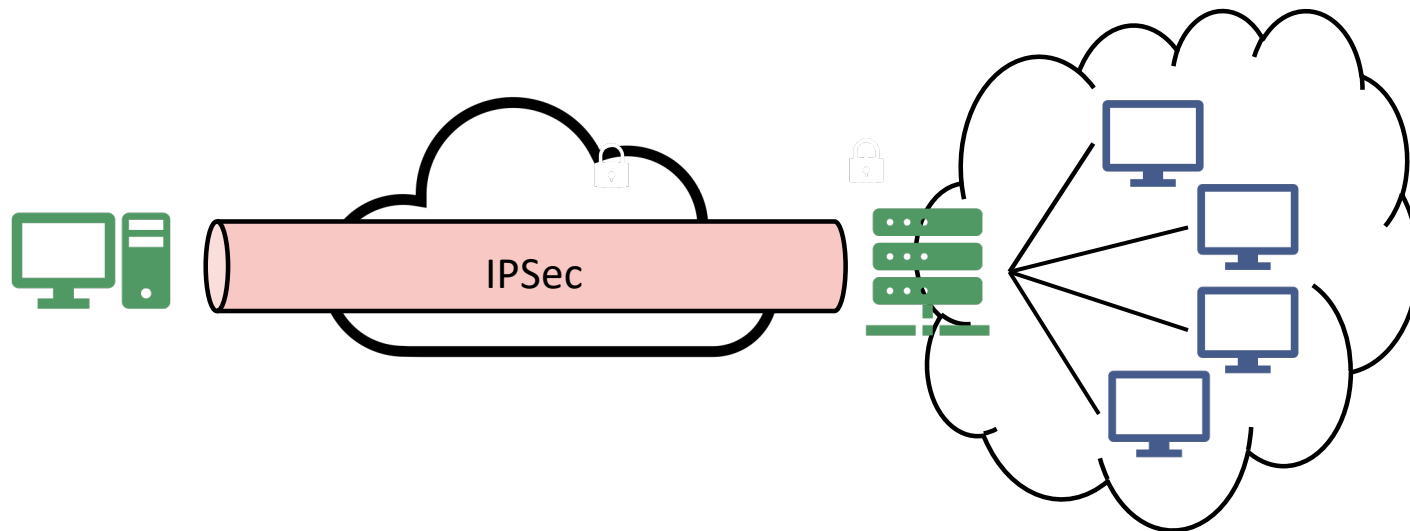


Tunnel mode, e.g., for securely connecting the networks of two branches of a company



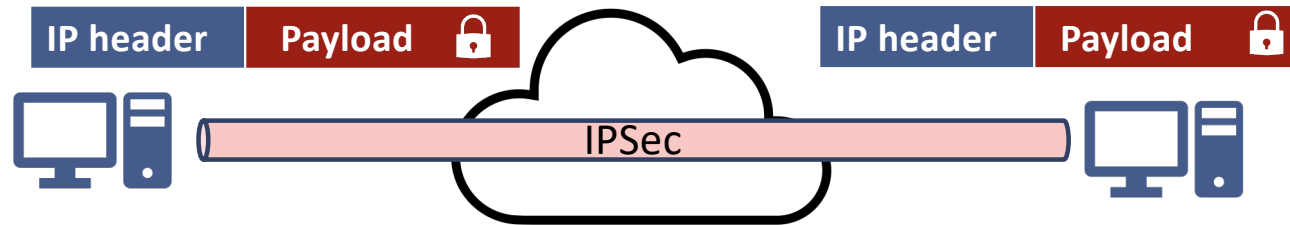
VPN Use Case

IPSec in tunnel mode is also used to connect remote hosts to an internal network

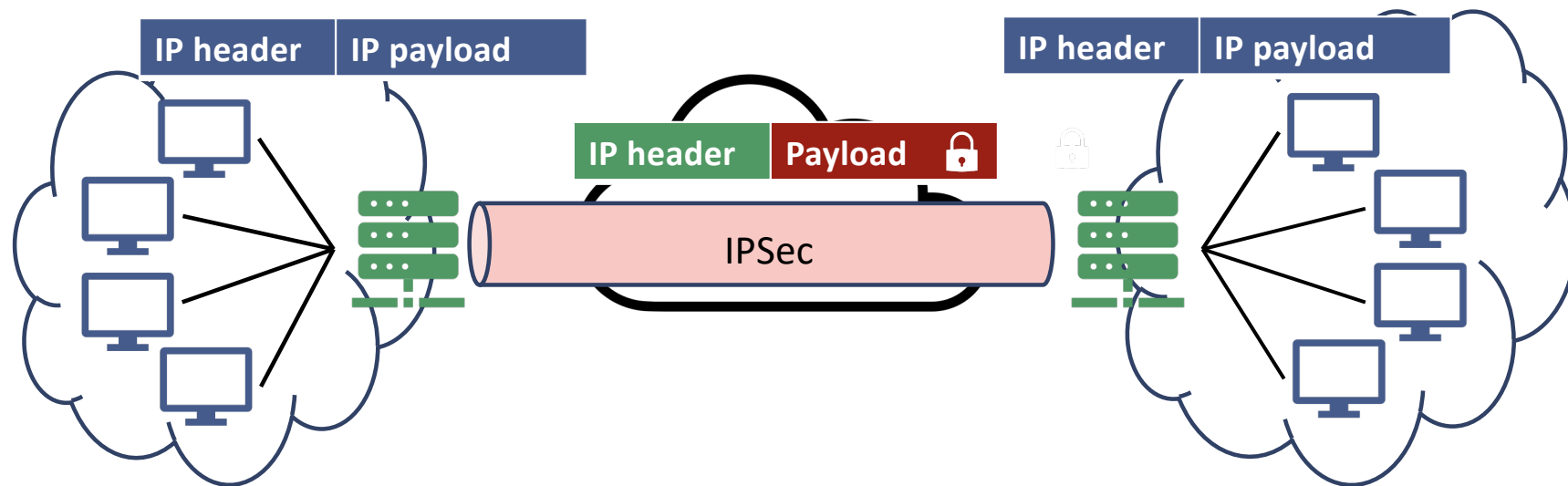


Tunnel Mode and Transport Mode and Primary Use Cases

Transport mode

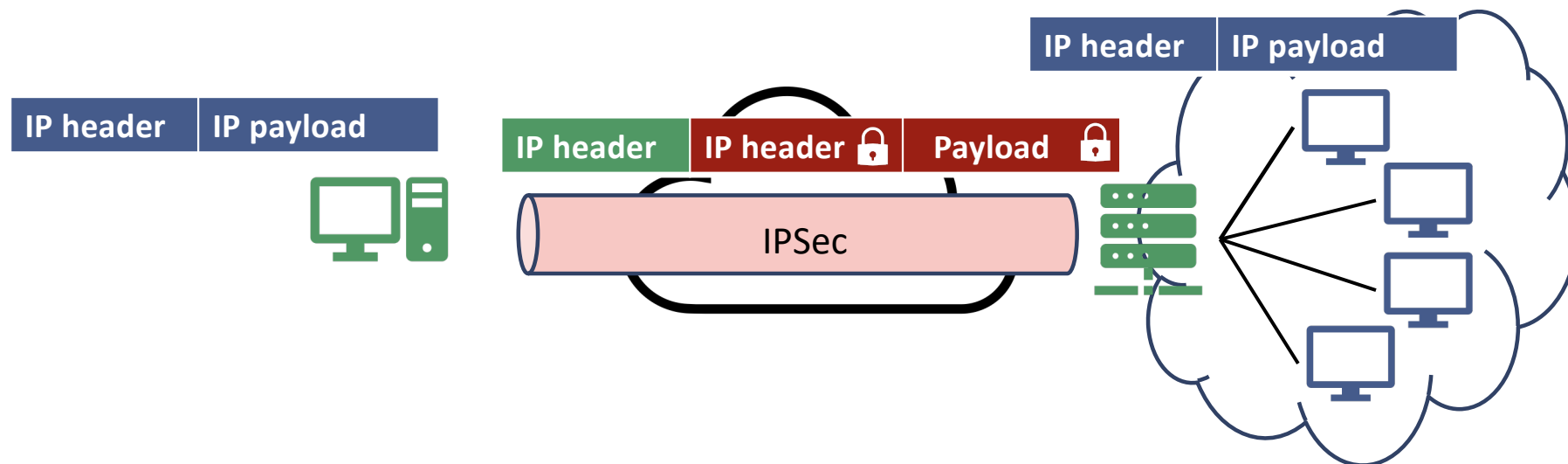


Tunnel mode, e.g., for securely connecting the networks of two branches of a company



VPN Use Case

IPSec in tunnel mode is also used to connect remote hosts to an internal network



Overview IPsec Part

Introduction

- ▶ Historical notes
- ▶ Security services offered by IPsec
- ▶ Transport Mode and Tunnel Mode
- ▶ Primary Use Cases

Encapsulating Security Payload Protocol ESP

- ▶ Encryption and Integrity Protection
- ▶ ESP Header
- ▶ MAC computation

Authentication Header Protocol AH

- ▶ Integrity Protection in the two modes
- ▶ ESP Header
- ▶ MAC computation
- ▶ Supported algorithms in AH and ESP
- ▶ Replay protection in AH and ESP

Authentication and Key Agreement with IKEv2

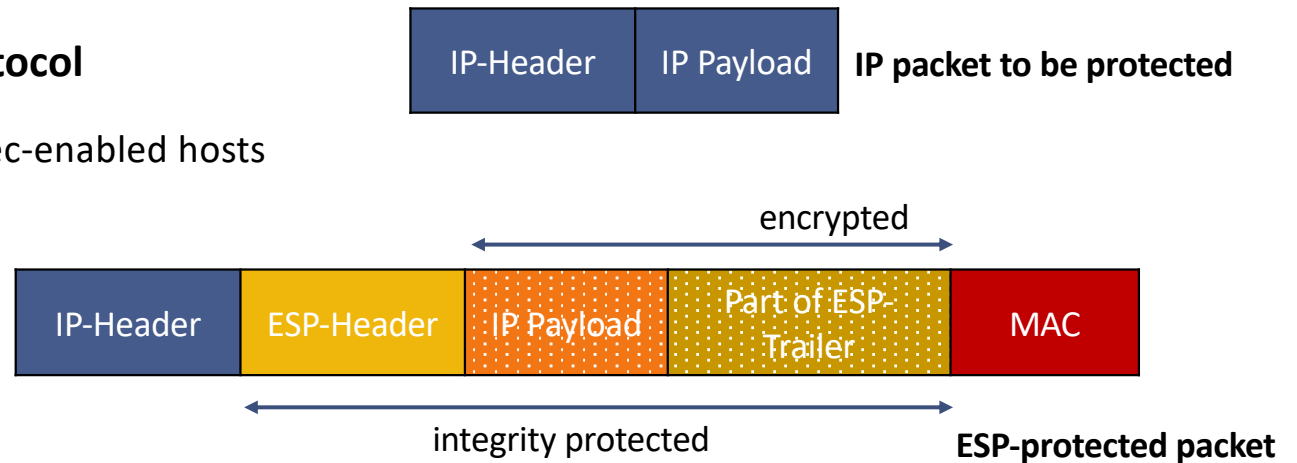
- ▶ The concept of security associations
- ▶ Overview on detailed discussion of IKEv2
- ▶ IP packet processing with IPsec
- ▶ Example use cases

Encryption and Integrity Protection offered by ESP

• Encapsulating Security Payload protocol

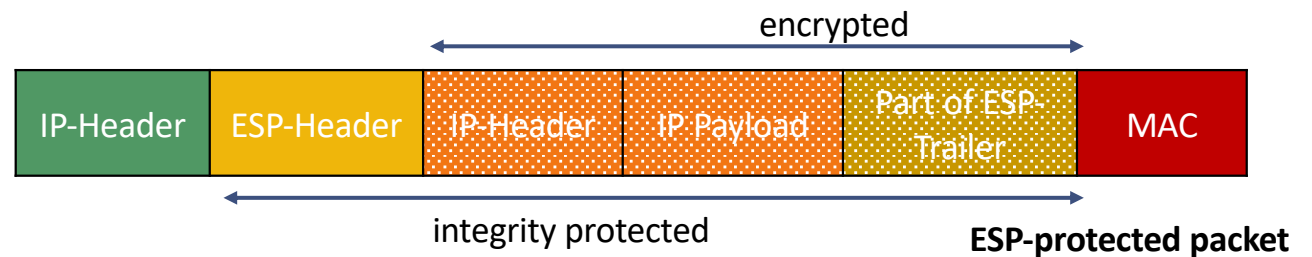
▶ Transport mode between two IPsec-enabled hosts

- Encryption of IP payload
- Integrity protection of IP payload
- Replay protection of IP packets



▶ Tunnel mode between two IPsec-enabled gateway

- Encryption of IP packets including IP headers routed through the gateway
- Integrity protection of IP packets including IP headers routed through the gateway
- Replay protection of IP packets



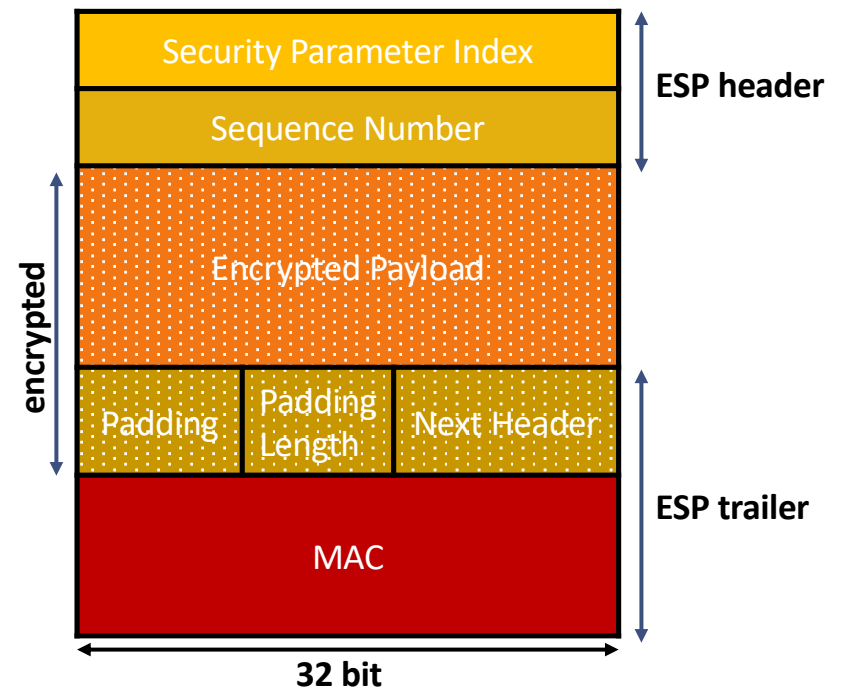
Payload of an ESP Protected IP Packet

ESP header

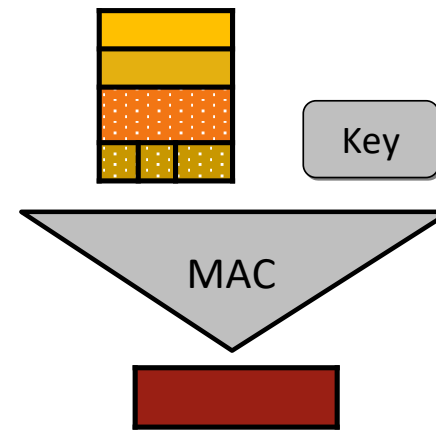
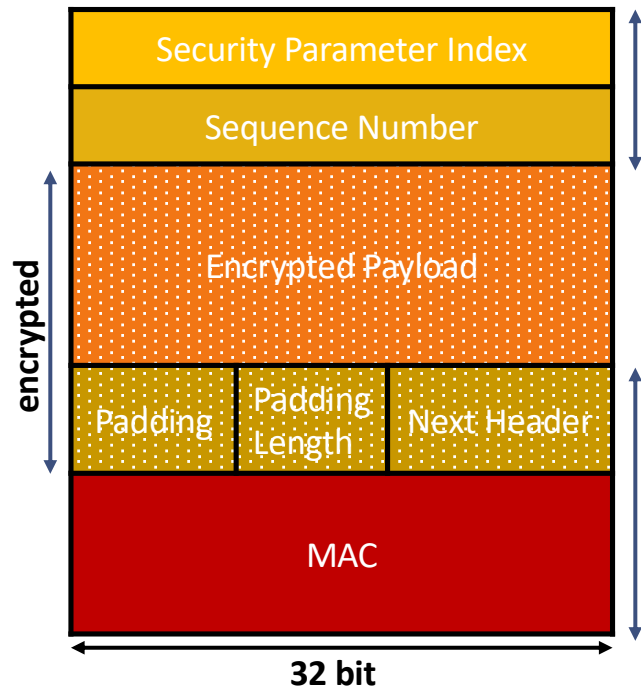
- ▶ **Security Parameter Index (SPI):** 32-bit
 - Identifies a security association (SA)
 - Specified what keys and algorithms to use
- ▶ **Sequence number:** 32-bit number per packet
 - Used for replay protection

ESP trailer

- ▶ **Padding field:** 0-255 padding bits
- ▶ **Padding length:** 8-bit length field
- ▶ **Next header:** 8-bit field indicating type of payload encrypted in the encrypted payload
- ▶ **MAC:** message authentication code



MAC Computation



MAC not computed on IP header

Overview IPsec Part

Introduction

- ▶ Historical notes
- ▶ Security services offered by IPsec
- ▶ Transport Mode and Tunnel Mode
- ▶ Primary Use Cases

Encapsulating Security Payload Protocol ESP

- ▶ Encryption and Integrity Protection
- ▶ ESP Header
- ▶ MAC computation

Authentication Header Protocol AH

- ▶ Integrity Protection in the two modes
- ▶ ESP Header
- ▶ MAC computation
- ▶ Supported algorithms in AH and ESP
- ▶ Replay protection in AH and ESP

Authentication and Key Agreement with IKEv2

- ▶ The concept of security associations
- ▶ Overview on detailed discussion of IKEv2
- ▶ IP packet processing with IPsec
- ▶ Example use cases

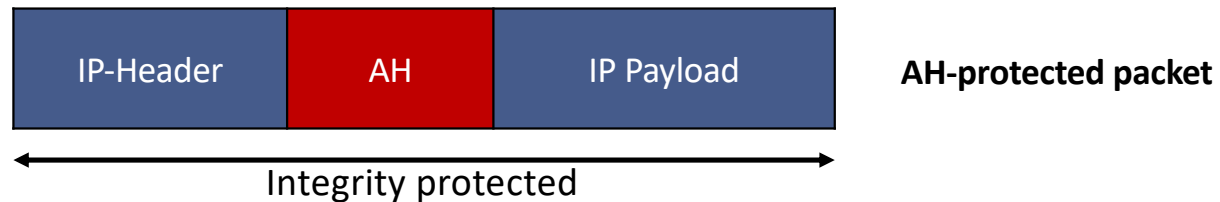
Integrity Protection offered by AH

- **Authentication Header Protocol**



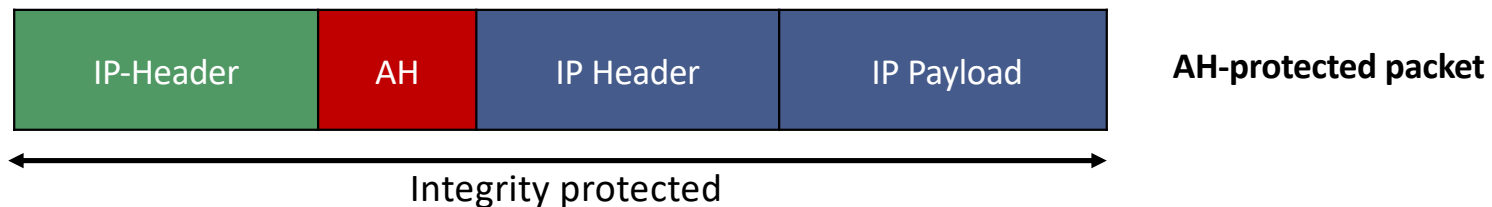
- ▶ **Transport mode** between two IPsec enabled hosts

- Integrity protection of the complete IP packet, including the header



- ▶ **Tunnel mode** between two IPsec-enabled gateway

- Integrity protection of the complete IP packet, including the new IP header



Authentication Header

Next header field

- ▶ 8-bit field, indicates type of header following the AH header
 - IP header in tunnel mode, first header in IP payload in transport mode

Payload length

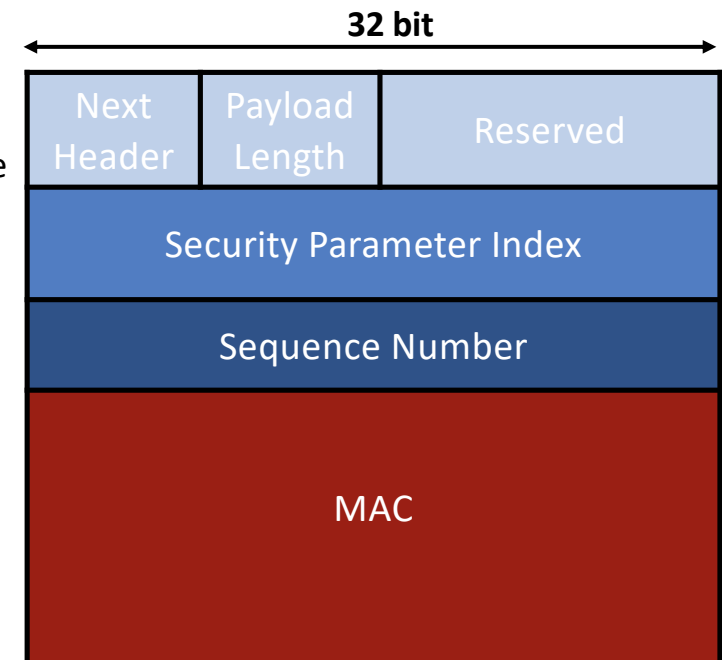
- ▶ 8-bit field defining length of authentication header
 - Depending on MAC algorithm, length of authentication data varies

Security Parameter Index (SPI)

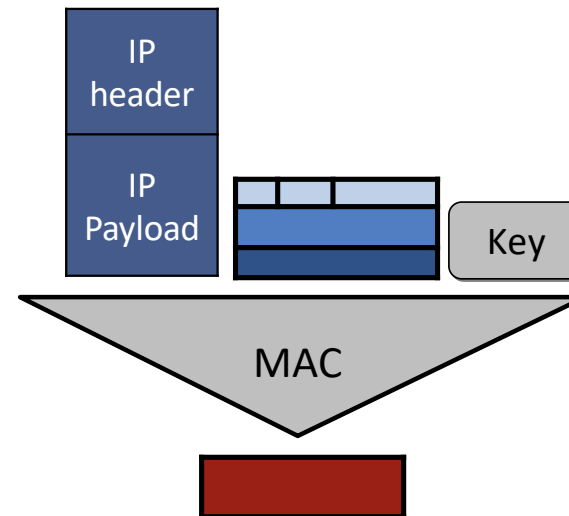
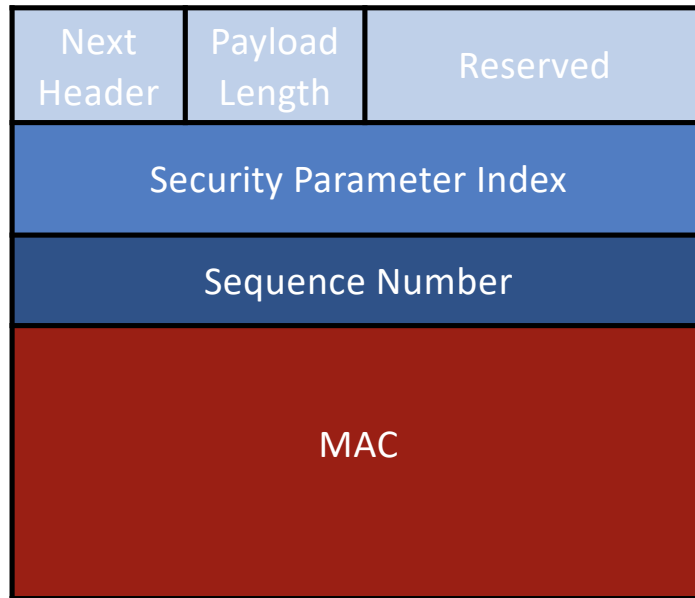
- ▶ 32-bit identifier of a security association (SA)
- ▶ Specified what keys and algorithms to use

Sequence number

- ▶ 32-bit sequence number incremented with each packet, used for replay protection

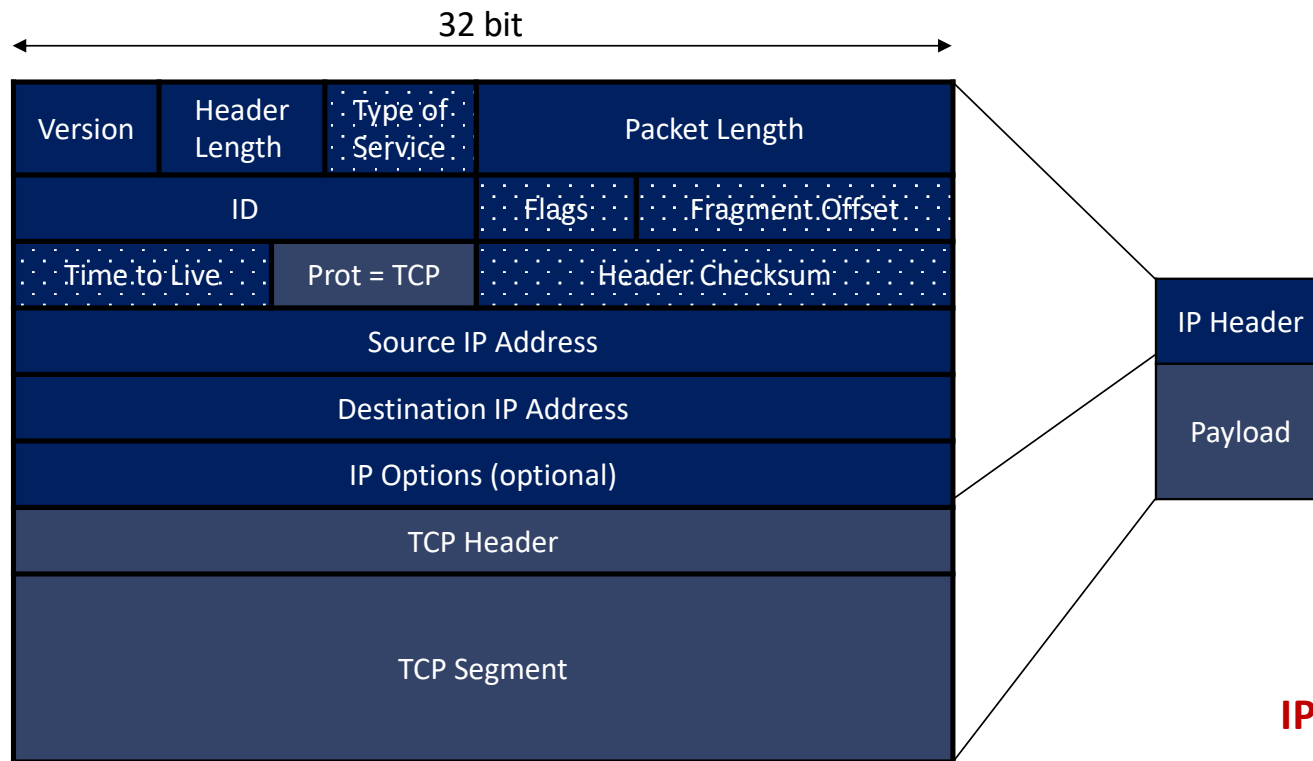


MAC Computation




- **Authentication header fields included in MAC-computation**
- **Non-mutable fields of outer IP header included in MAC-computation**
 - ▶ Mutable fields such as TTL, Header Checksum, Fragment Offset etc. can and should not be protected

Recap: Mutable Fields in the IPv4 Header



IPsec supports both IPv4 and IPv6!

Mutable fields  are set to zero for the MAC calculation in AH

Most recent MAC algorithm support for AH RFC 8221

Name	Status
AUTH_NONE	MUST NOT
AUTH_HMAC_MD5_96	MUST NOT
AUTH_HMAC_SHA1_96	MUST- (=expected to be phased out soon)
AUTH_DES_MAC	MUST NOT
AUTH_KPDK_MD5	MUST NOT
AUTH_AES_XCBC_96	SHOULD for IoT / MAY otherwise
AUTH_AES_128_GMAC	MAY
AUTH_AES_256_GMAC	MAY
AUTH_HMAC_SHA2_256_128	MUST
AUTH_HMAC_SHA2_512_256	SHOULD

Recommendations change over time, latest ones currently from 2017

XCBC is a predecessor of CMAC that differs in the generation of the masking keys

Most recent MAC algorithm support for ESP RFC 8221

Name	Status
AUTH_NONE	MUST (in comb. with combined enc/integ.) / MUST NOT
AUTH_HMAC_MD5_96	MUST NOT
AUTH_HMAC_SHA1_96	MUST- (=expected to be phased out soon)
AUTH_DES_MAC	MUST NOT
AUTH_KPDK_MD5	MUST NOT
AUTH_AES_XCBC_96	SHOULD for IoT / MAY otherwise
AUTH_AES_128_GMAC	MAY
AUTH_AES_256_GMAC	MAY
AUTH_HMAC_SHA2_256_128	MUST
AUTH_HMAC_SHA2_512_256	SHOULD

Recommendations change over time, latest ones currently from 2017

Same as for AH except for the first one

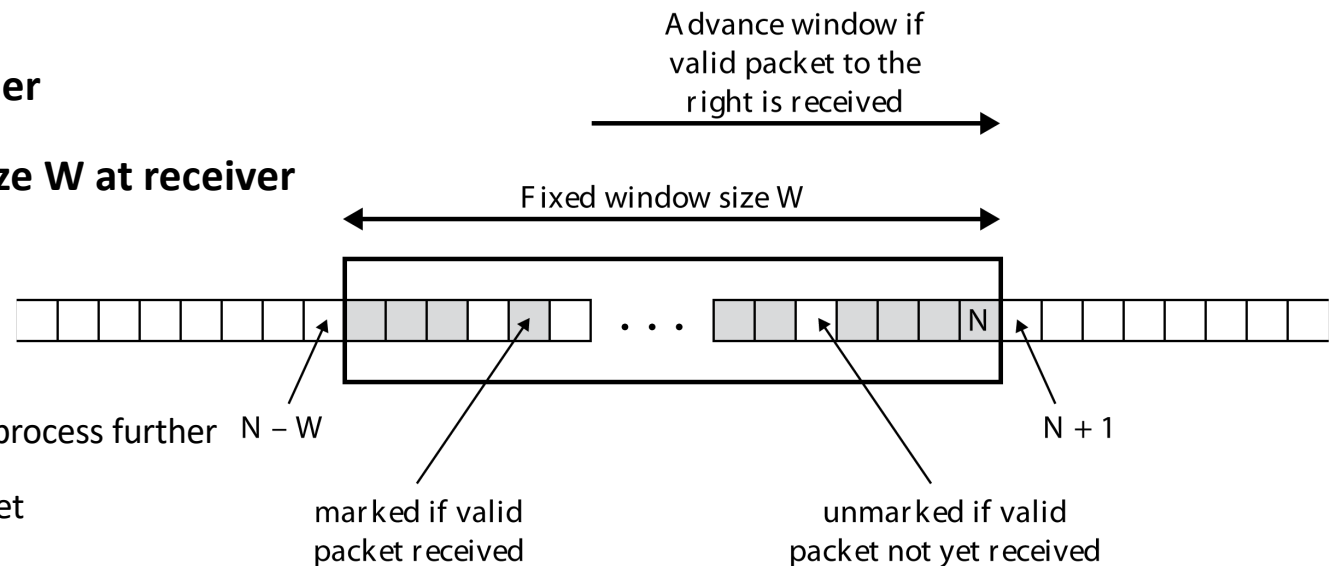
Most recent Encryption algorithm support for ESP RFC 8221

Name	Status
ENCR_DES_IV64	MUST NOT
ENCR_DES	MUST NOT
ENCR_3DES	SHOULD NOT
ENCR_BLOWFISH	MUST NOT
ENCR_3IDEA	MUST NOT
ENCR_DES_IV32	MUST NOT
ENCR_NULL	MUST
ENCR_AES_CBC	MUST
ENCR_AES_CCM	SHOULD (provides integrity as well)
ENCR_AES_GCM	MUST (provides integrity as well)
ENCR_CHACHA20_POLY1305	SHOULD (provides integrity as well)

Recommendations change over time, latest ones currently from 2017

Replay Protection in ESP and AH

- SQN included in ESP and AH header
- Window of acceptable SQNs of size W at receiver
- SQN checking at receiver
 - ▶ If SQN is in current window
 - **and not yet marked:** mark and process further
 - **and already marked:** drop packet
 - ▶ If SQN is lower than left boarder of window
 - drop packet, log event
 - ▶ If SQN is higher than current right boarder
 - mark as received, move window to include SQN as right boarder, process packet further



Recommended window size

- ▶ Should be ≥ 32
- ▶ Recommends default 64

Overview IPsec Part

Introduction

- ▶ Historical notes
- ▶ Security services offered by IPsec
- ▶ Transport Mode and Tunnel Mode
- ▶ Primary Use Cases

Encapsulating Security Payload Protocol ESP

- ▶ Encryption and Integrity Protection
- ▶ ESP Header
- ▶ MAC computation

Authentication Header Protocol AH

- ▶ Integrity Protection in the two modes
- ▶ ESP Header
- ▶ MAC computation
- ▶ Supported algorithms in AH and ESP
- ▶ Replay protection in AH and ESP

Authentication and Key Agreement with IKEv2

- ▶ The concept of security associations
- ▶ Overview on detailed discussion of IKEv2
- ▶ IP packet processing with IPsec
- ▶ Example use cases

Authentication and Key Agreement

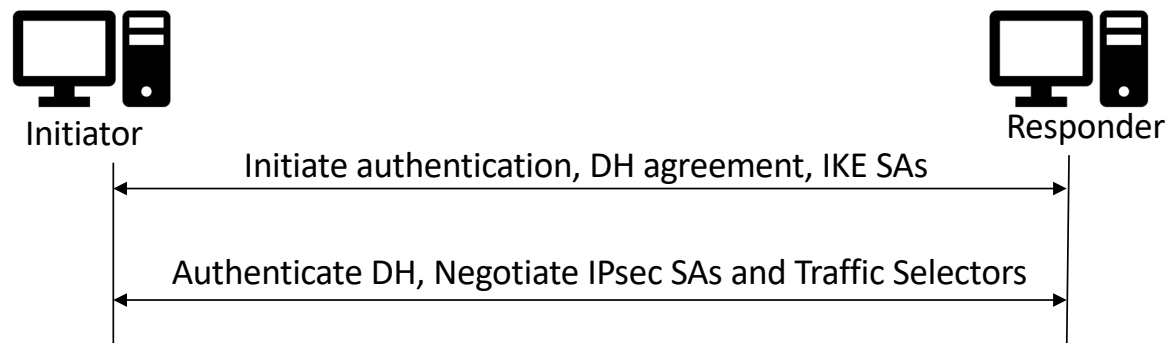
Security Association

- ▶ Identified uniquely by a 32-bit Security Parameter Index **SPI**
- ▶ **Security protocol type:** determines if the SA is for IKE, AH or ESP usage
- ▶ **Algorithm information:** Encryption and / or MAC algorithms, keys
- ▶ **Replay Window:** Current start point and size of replay window
- ▶ **SQN:** Current Value of the sequence number SQN
- ▶ **IPSec Mode:** Indicates if SA is usable for transport mode, tunnel mode or both
- ▶ **SA lifetime:** Lifetime of the security association
 - lifetime can be based on time, byte count, or both

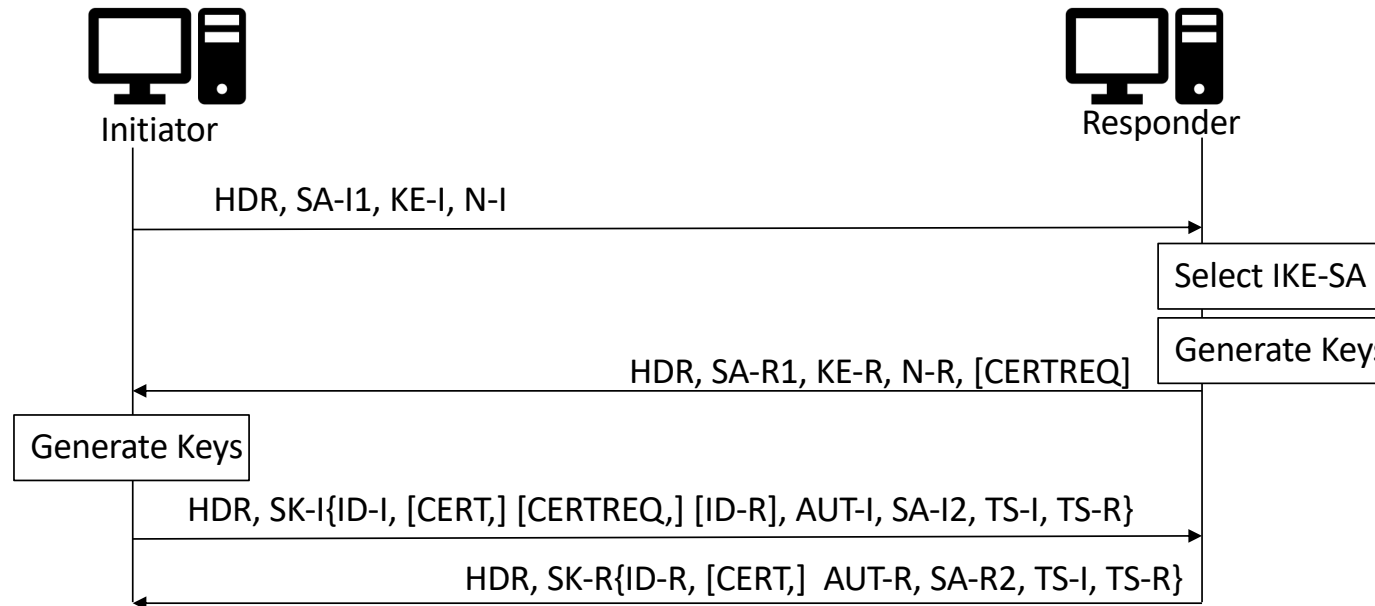
Authentication and Key Agreement

The Internet Key Management Protocol IKEv2

- ▶ Supports authentication and key agreement between two IPsec-enabled peers
 - Establishes at least two pairs of **security associations** (SAs) between the peers
 - One **IKE-SA** pair to protect the authentication and key agreement itself
 - One **IPSec-SA** pair to use with ESP and / or AH in tunnel or transport mode later
- ▶ Peer starting the protocol is called the **initiator**, the other peer is called **responder**



IKE v2 Exchange: Complete Overview



HDR: header, contains SPIs
 SA-I1: SAs offered for IKE
 SA-R2: SA selected
 KE-I, KE-R: public DH values
 N-I, N-R: nonces

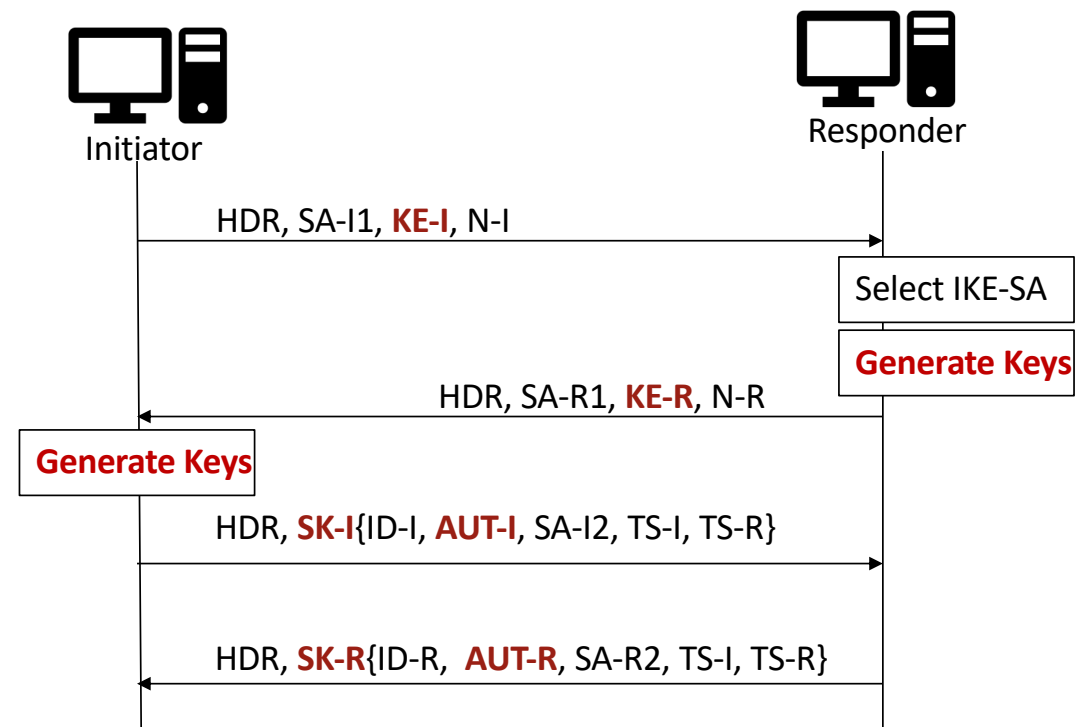
CERTREQ: certificate request
 CERT: certificate
 ID-I, ID-R: identifier
 AUT-I, AUT-R: sign. or MAC
 SA-I2: SA offered for IPsec

SA-R2: SA selected for IPsec
 TS-I, TS-R: traffic selectors
 SK-I, SK-R: encrypted with SKe
 and integrity protected with SK-a

IKE v2 Exchange: Authentication and Key Agreement

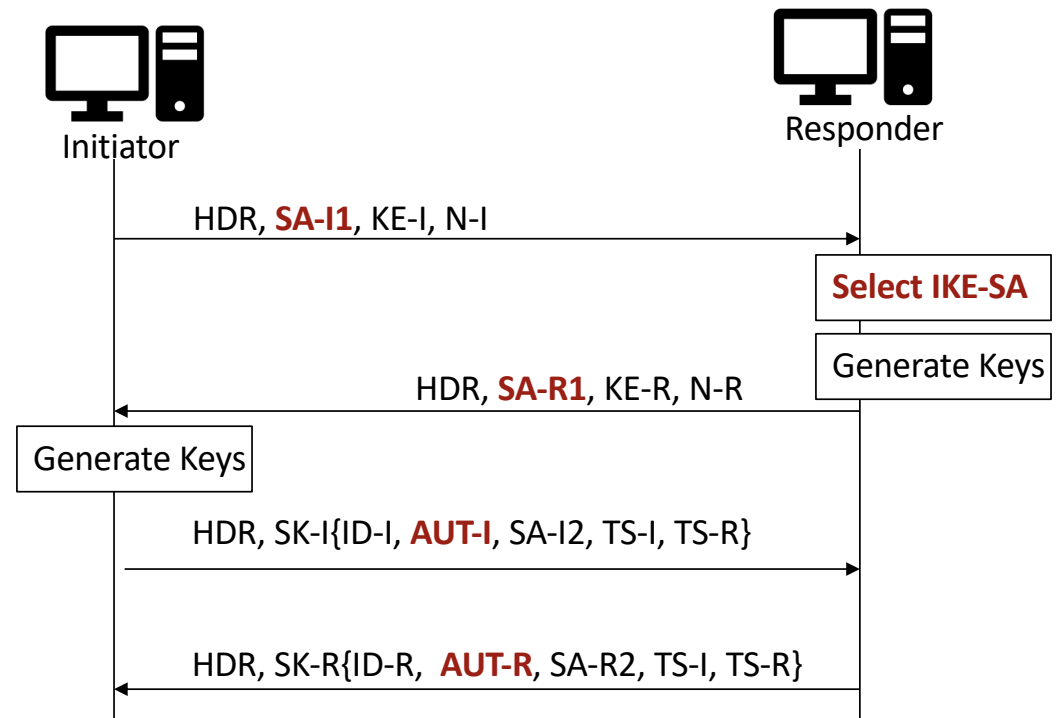
Variant of the secure authenticated DH

- DH values **KE-I** and **KE-R** exchanged in the clear
- Keys for encryption and integrity protection during IKE exchanged and further key derivation for IPSec SAs derived from nonces and DH values
- Authenticated by **AUT-I** and **AUT-R** using digital signatures or pre-shared keys
 - ▶ $AUT-I = \text{sign}(h(\text{message 1} \parallel N-R \parallel \text{MAC}_{SKp-i}(ID-I)))$ or
 $AUT-I = \text{MAC}(\text{message 1} \parallel N-R \parallel \text{MAC}_{SKp-i}(ID-I))$
 - ▶ $AUT-R = \text{sign}(h(\text{message 2} \parallel N-I \parallel \text{MAC}_{SKp-r}(ID-R)))$ or
 $AUT-R = \text{MAC}(\text{message 2} \parallel N-I \parallel \text{MAC}_{SKp-r}(ID-R))$
- Message 3 and 4 encrypted and integrity protected



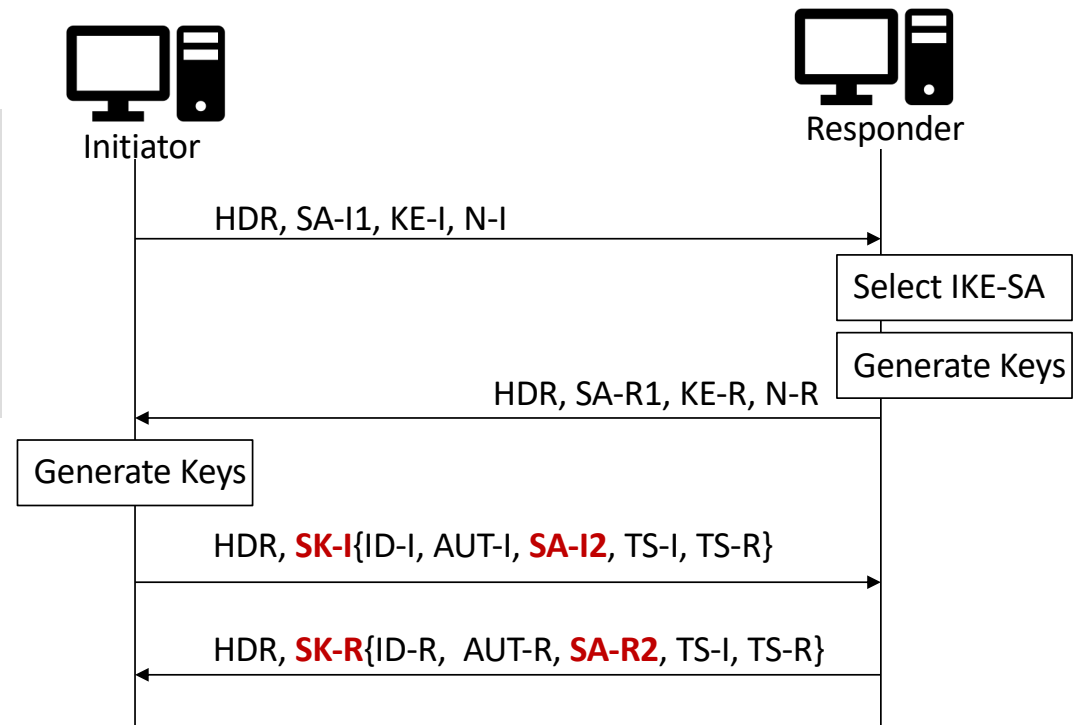
IKE v2 Exchange: IKE-SA Negotiation

- Initiator sends proposals for IKE-SAs in message 1
- Responder selects IKE-SA and includes selection in message 2
- Proposal and selection protected against manipulation with **AUT-I** and **AUT-R**



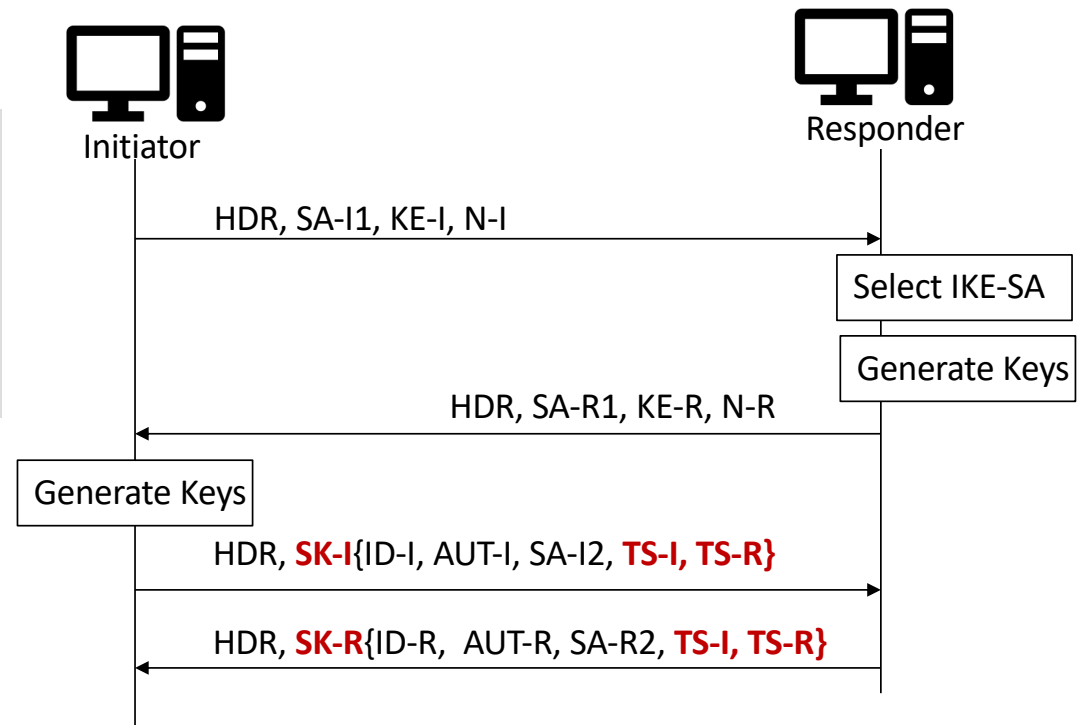
IKE v2 Exchange: IPSec-SA Negotiation

- Initiator sends proposals for IPSec-SAs in message 3
- Responder selects IPSec-SA, includes selection in message 4
- Proposal and selection protected against manipulation with integrity protection (and encryption) by **SK-I** and **SK-R**



IKE v2 Exchange: Negotiation of Traffic Selectors

- Initiator sends proposals for Traffic Selectors in message 3
- Responder includes selected Traffic Selectors in message 4
- Proposal and selection protected against manipulation with **SK-I** and **SK-R**



Traffic Selectors and Security Policy Database

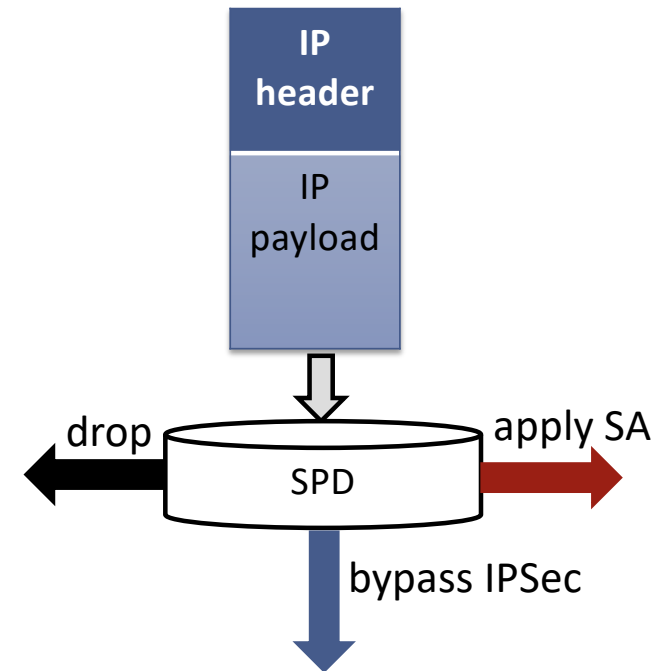
Traffic selectors are stored in a Security Policy Database

Traffic selectors specify

- ▶ Set of source IP addresses (one, list, range, wildcard)
- ▶ Set of destination IP addresses (one, list, range, wildcard)
- ▶ Transport layer protocol number (one, list, range, wildcard)
- ▶ Source and destination port (one, list, or wildcard)

Traffic selectors determine

- ▶ Whether inbound and outbound IP packets are protected, bypassed, or dropped
- ▶ If packet is to be protected, corresponding traffic selector points to the SA to use, if non exists yet, a new one is generated with IKE



Supported Algorithms

Encryption algorithms currently recommended for IKEv2 (RFC 8247)

- ▶ ENCR_AES_CBC MUST
- ▶ ENCR_AES_CCM SHOULD (supports integrity protection simultaneously)
- ▶ ENCR_AES_GCM SHOULD (supports integrity protection simultaneously)
- ▶ ENCR_CHACHA20_POLY1305 SHOULD (supports integrity protection simultaneously)

Integrity protection algorithms currently recommended for IKEv2 (RFC8247)

- ▶ AUTH_HMAC_SHA2_512_256 SHOULD
- ▶ AUTH_HMAC_SHA2_256_128 MUST

Recommendations change over time!

Examples of where else IPsec is used today

- **Many VPNs use IPsec between the VPN Client and Server**

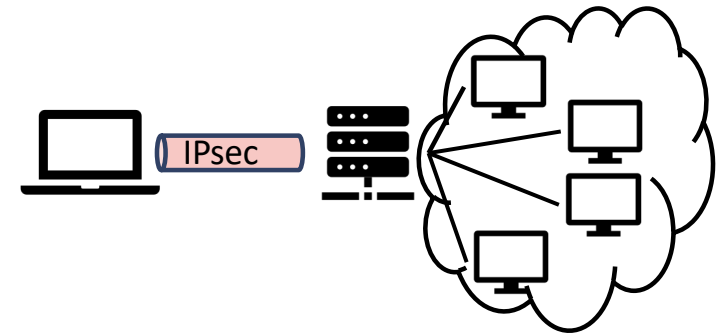
- ▶ Including the Cisco AnyConnect VPN Client used by RWTH

- **Connections between WLAN access points and authentication servers**

- ▶ E.g., in Eduroam IPsec is used to protect the transfer of session keys from the authentication server to the WLAN access point

- **Connections between backbone components in mobile systems**

- ▶ E.g., between base stations and backbone components or between backbone components that exchange subscriber information



Overview

IPSec

- ▶ Main use case
- ▶ Security services offered
- ▶ Authentication and key agreement
- ▶ Payload or packet protection

TLS 1.3

- ▶ Main use case
- ▶ Security services offered
- ▶ Handshake Protocol
- ▶ Payload protection with record protocol

Comparison of the protocols

- ▶ Differences
- ▶ Communalities in mechanisms used
- ▶ Overlaps in use cases

Transport Layer Security Protocols over the Years

- **Secure Socket Layer SSL**

- ▶ Predecessor of TLS, first version developed by Netscape in 1994

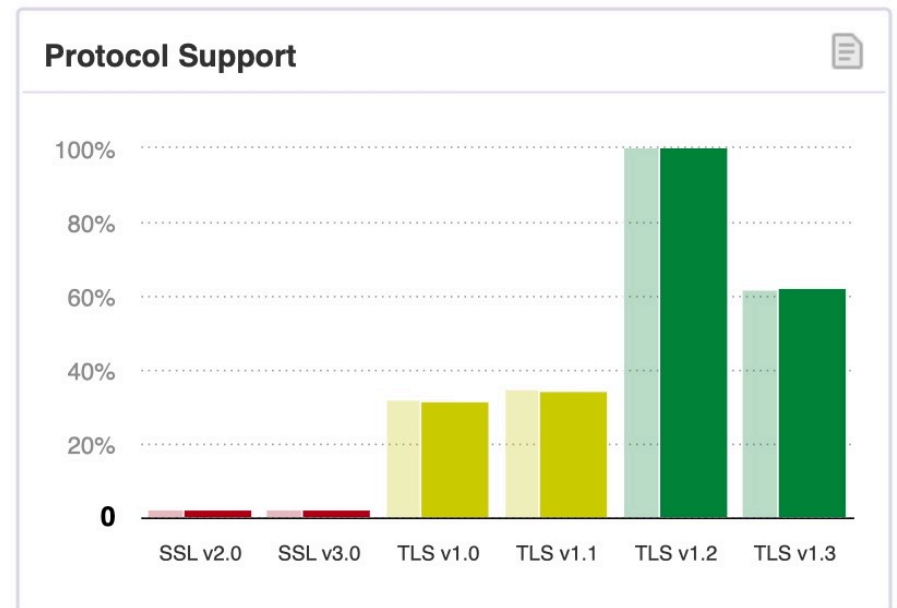
- **Transport Layer Security TLS**

- ▶ Standardized by the IETF
- ▶ TLS 1.0 and TLS 1.1 should not be used any more
- ▶ TLS 1.2 still in use but has many weakness and only very few unbroken configurations

- **TLS 1.3 standardized in RFC 8446 in 2018**

- **We focus on TLS 1.3**

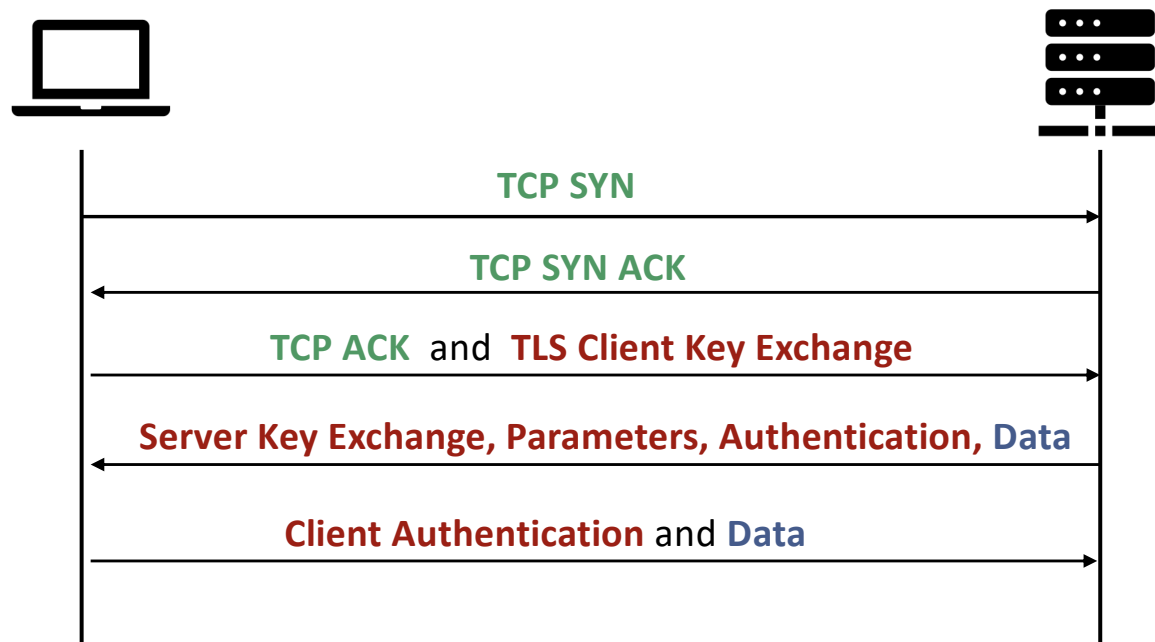
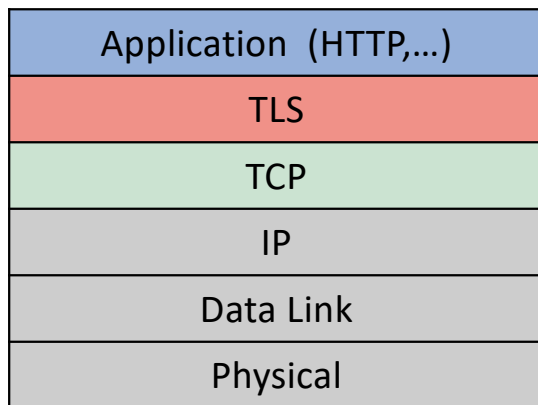
TLS version support of the top 150 000 visited websites according to the Alexa list (May 2023)



<https://www.ssllabs.com/ssl-pulse/>

Primary Use Case of TLS

- Transport layer protection of application traffic between a client and a server
- Most important use case
 - ▶ HTTP over TLS = HTTPS
- Other uses include
 - ▶ SMTP over TLS = SMTPS
 - ▶ DNS over TLS = DoT



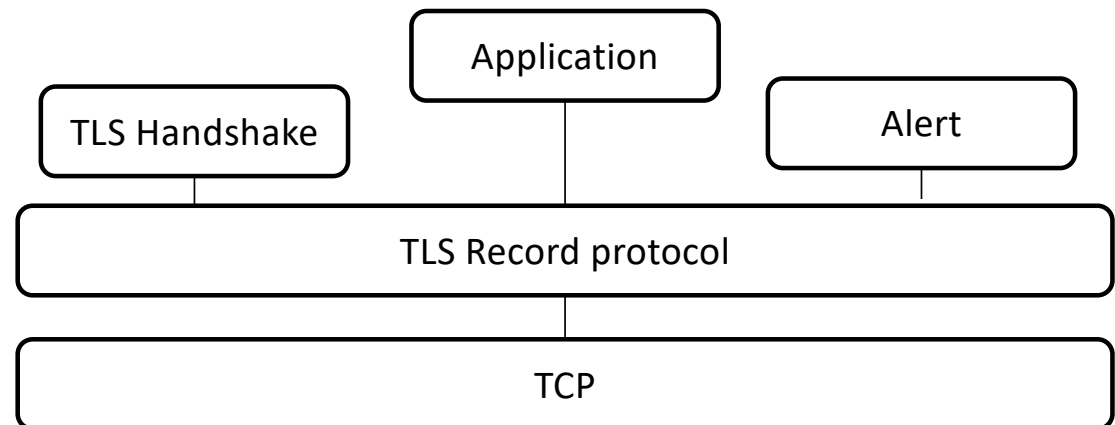
Security Services offered by TLS 1.3

- **Authenticated session key agreement**

- ▶ Using the **TLS Handshake protocol**
- ▶ Supports three key agreement methods
 - PSK-only
 - PSK-authenticated DH
 - Signature authenticated DH

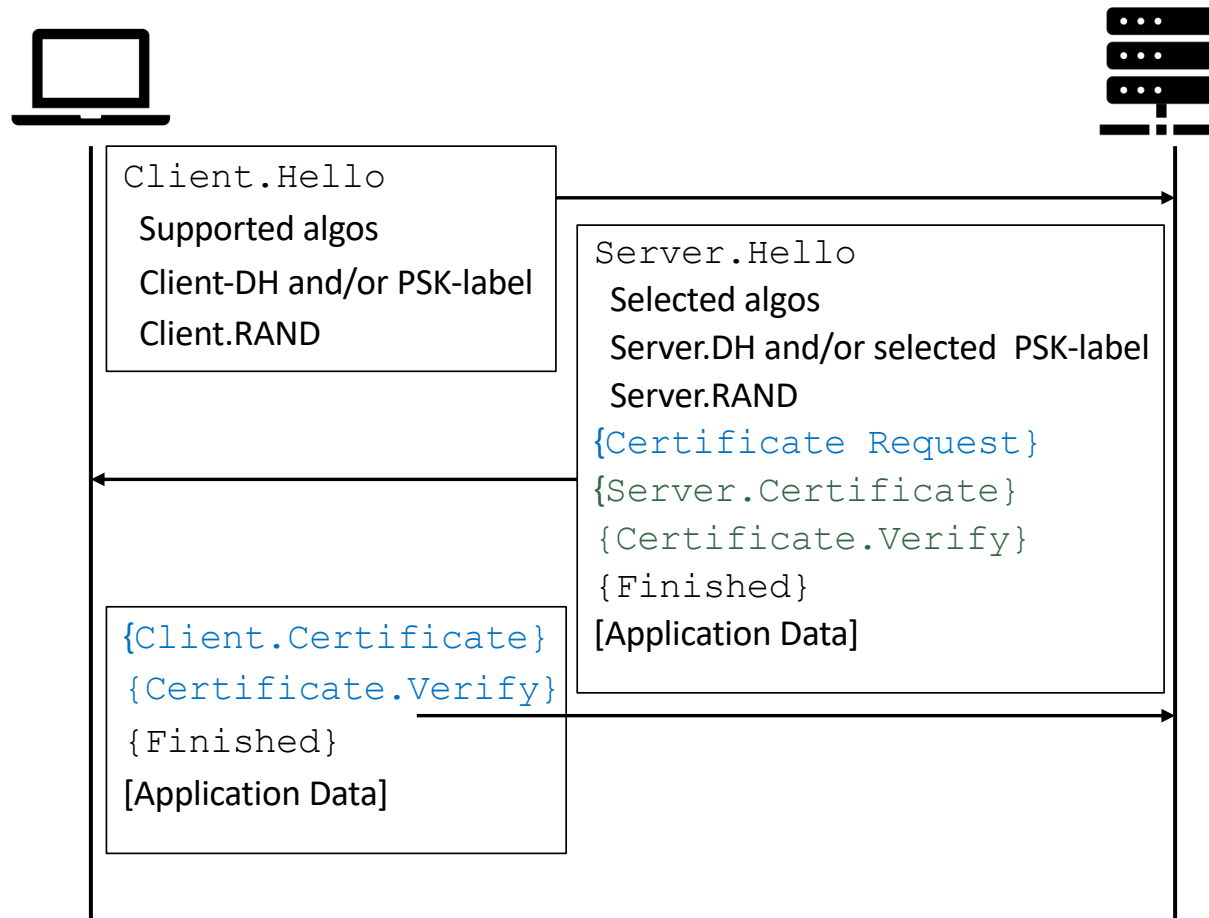
- **Encryption and integrity protection**

- ▶ Of application data, part of the handshake, alert and change cipher spec messages
- ▶ Using the **TLS Record Protocol**



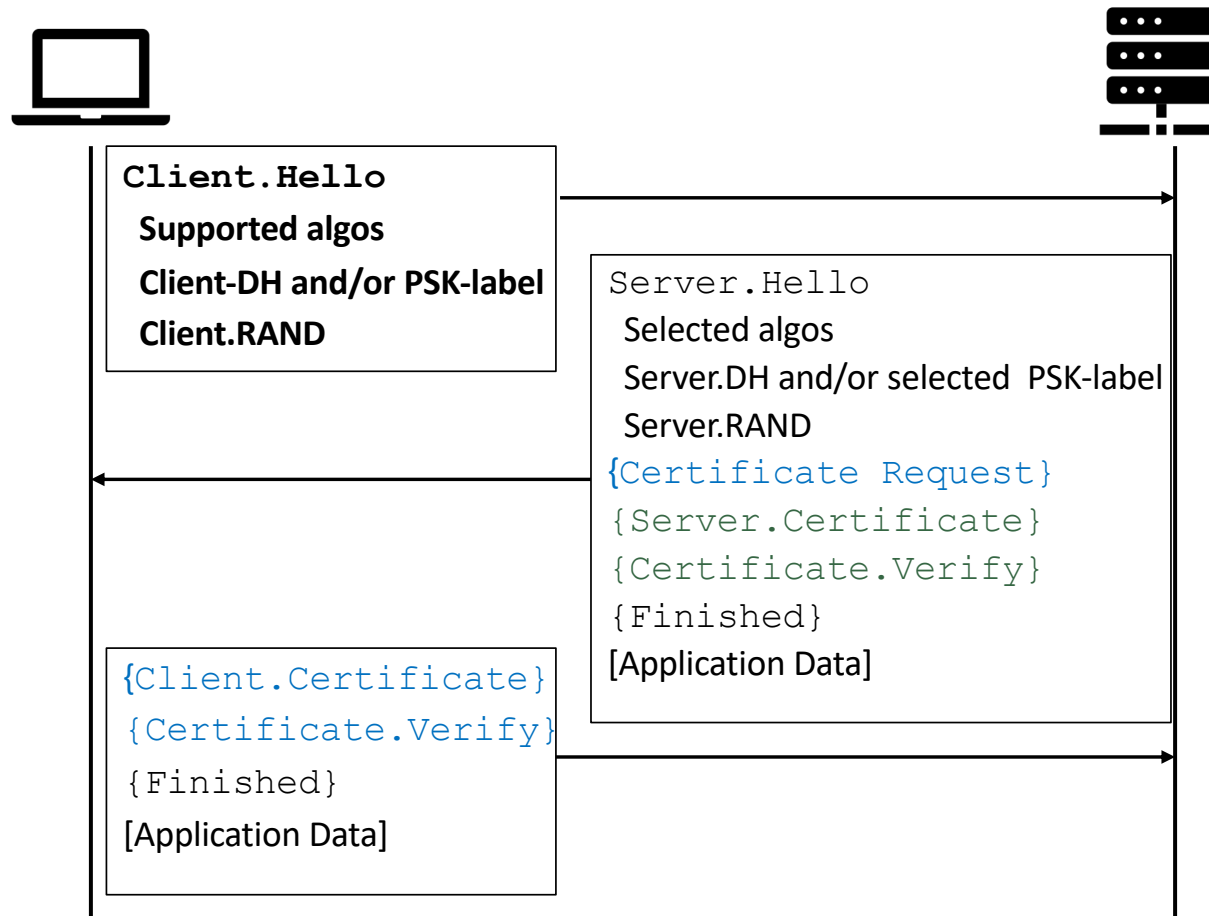
Note: we focus on **TLS 1.3** here
Most resources on the Web are still on TLS 1.2

Authentication and Key Agreement: TLS 1.3 Handshake Overview



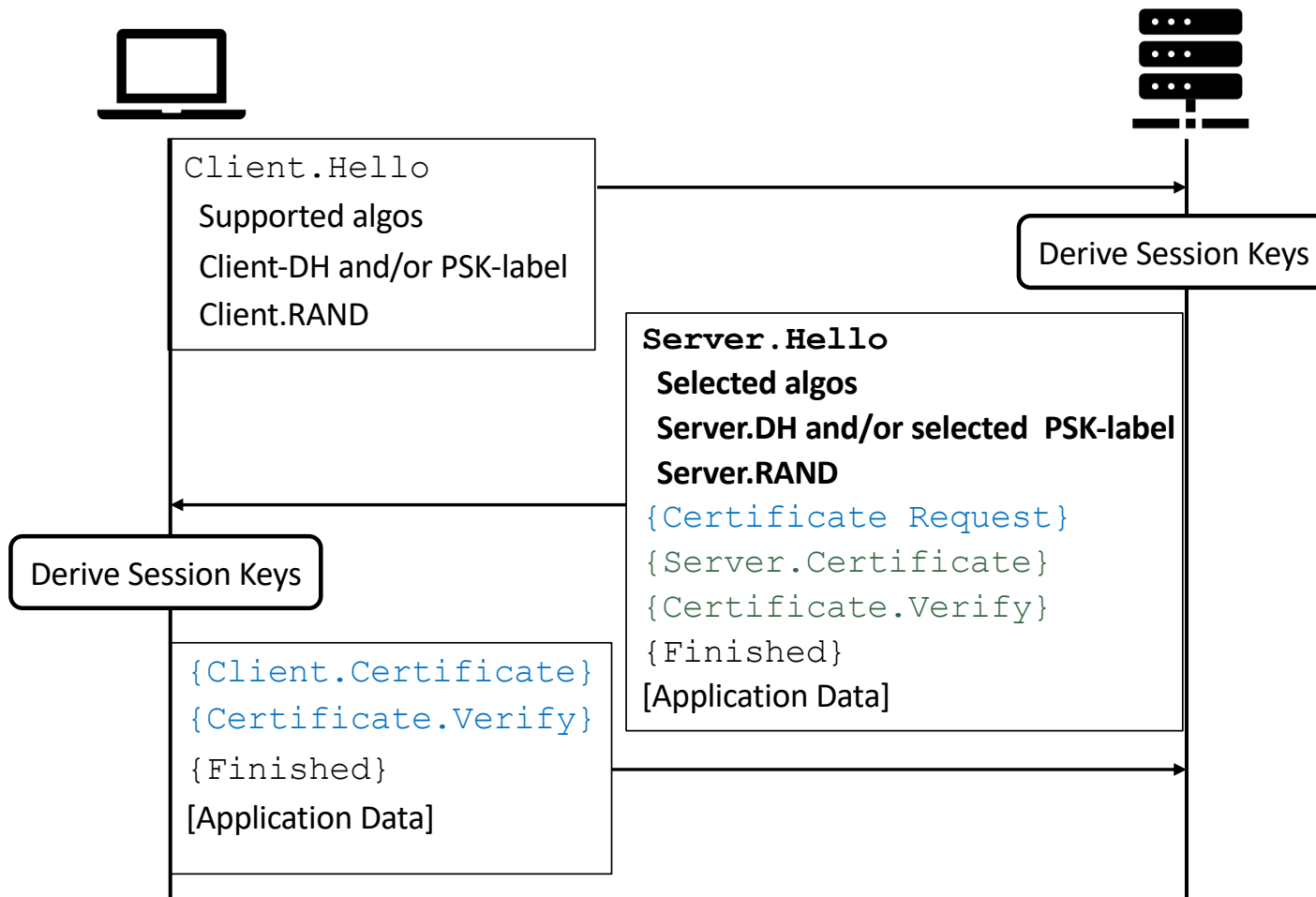
- **{}** encrypted and integrity protected handshake messages
- **[]** encrypted and integrity protected application data (different keys used)
- **Only sent if certificate-based client authentication required**
- **Only sent if DH is authenticated with server signature**

Authentication and Key Agreement: TLS Handshake Key Exchange Phase (1)



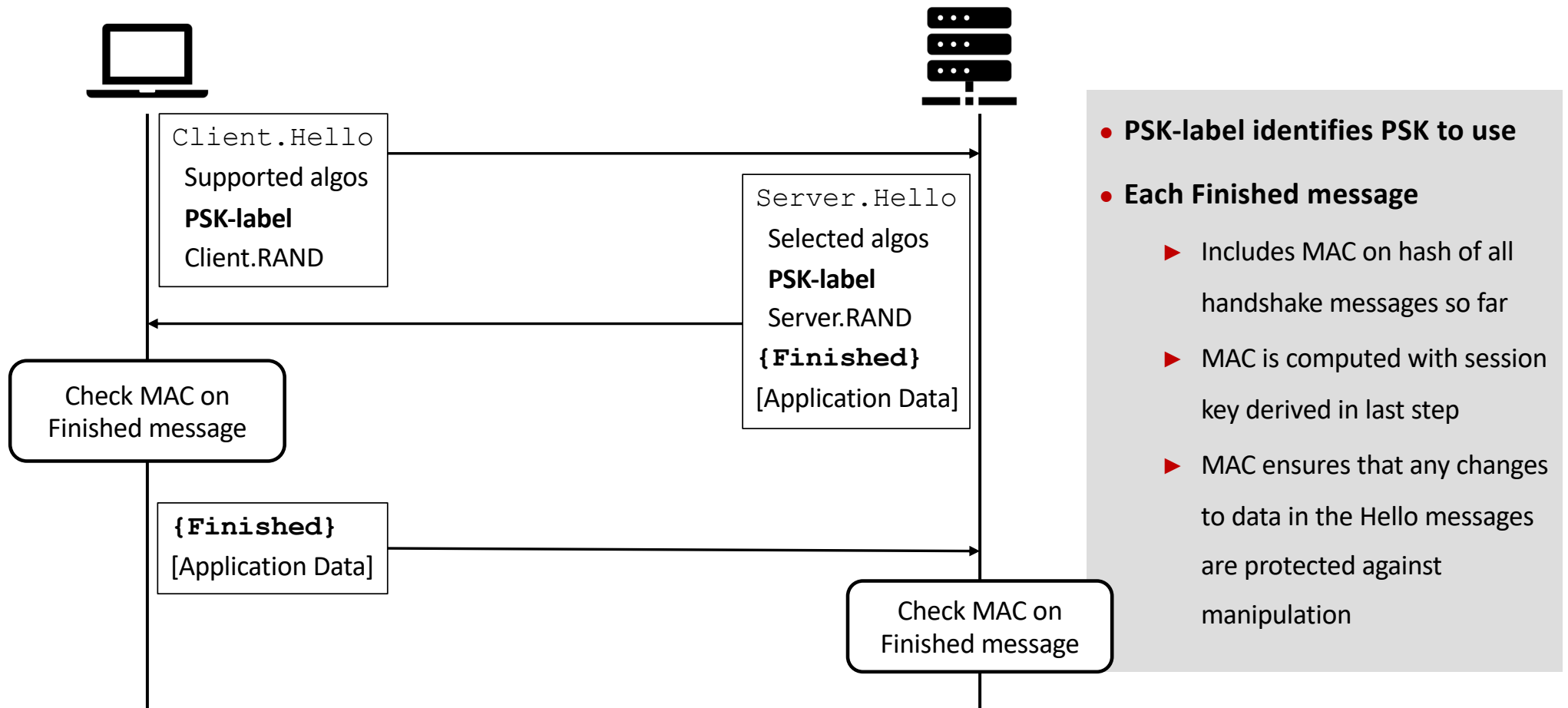
- **In Client.Hello, client offers**
 - ▶ several DH-values for several groups and/or
 - ▶ several PSK-labels identifying PSKs
 - ▶ Encryption and integrity protection algorithms it supports
- **and includes**
 - ▶ a fresh random number `Client.RAND`

Authentication and Key Agreement: TLS Handshake Key Exchange Phase (2)



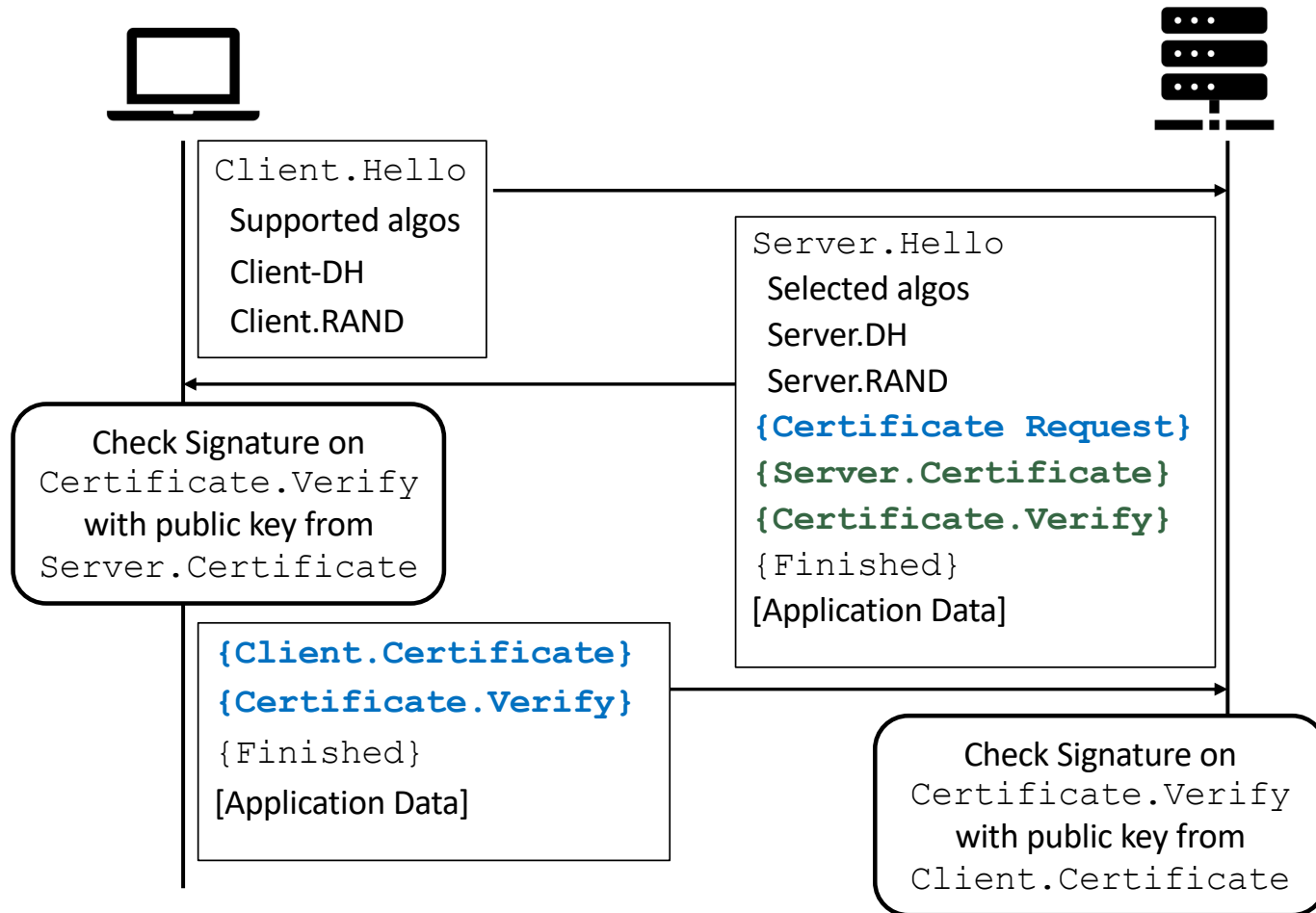
- **In Sever.Hello, server includes**
 - ▶ DH-values for the selected group and/or
 - ▶ PSK-label of selected PSK
 - ▶ Selected enc. and int. algos
- **Client and Server can now**
 - ▶ Compute the DH-Key and/or
 - ▶ Identify and retrieve the PSK to use
 - ▶ Derive session keys from DH-Key and / or PSK

TLS 1.3 Handshake PSK-Only Key Exchange



- **PSK-label identifies PSK to use**
- **Each Finished message**
 - ▶ Includes MAC on hash of all handshake messages so far
 - ▶ MAC is computed with session key derived in last step
 - ▶ MAC ensures that any changes to data in the Hello messages are protected against manipulation

TLS 1.3 Handshake DHE (with signatures) Key Exchange



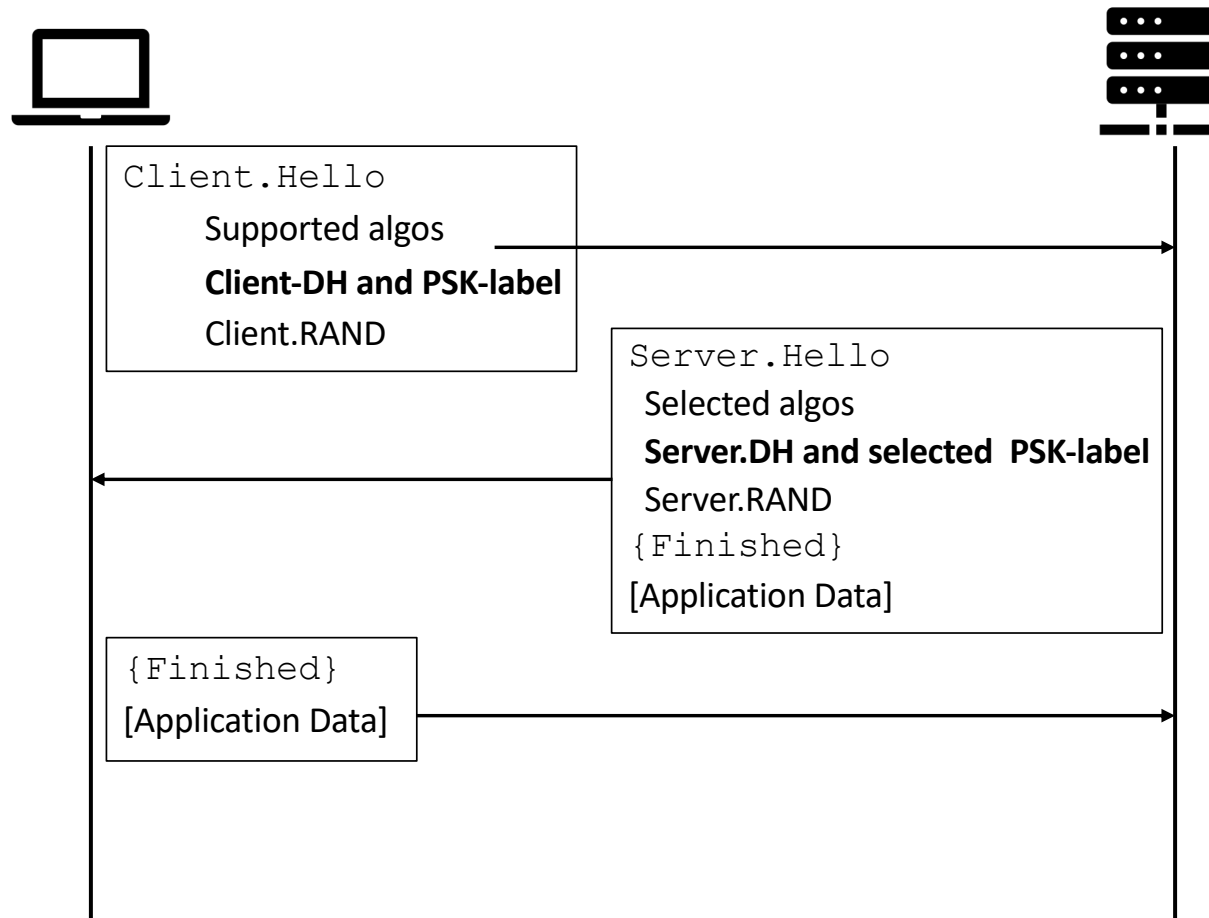
• Server sends

- ▶ Certificate request if client is to be authenticated with a certificate
- ▶ its own Server.Certificate including a chain of certificates
- ▶ Certificate.Verify message with a signature on the hash of all handshake messages with server's private key
- ▶ Finished message as before

• If requested Client sends

- ▶ Client.Certificate and Certificate.Verify

TLS1.3 Handshake PSK with DHE Key Exchange

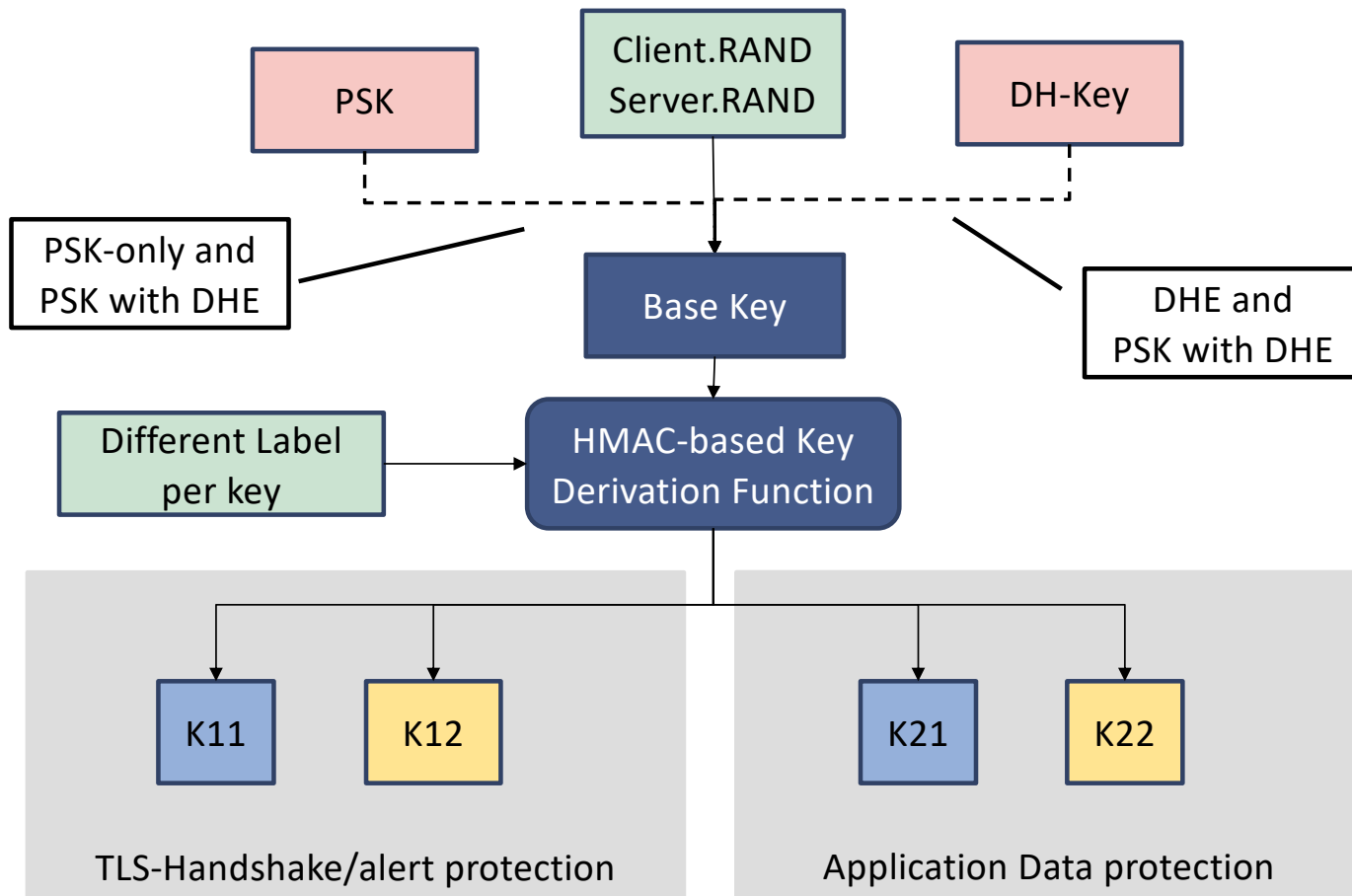


- Same as PSK-only
- But session keys derived from PSK and DH-key

PSK-only and PSK with DHE-key

- ▶ can also be used after a full handshake with DHE and signatures to resume
- ▶ In this case, the first message from the client may already contain data
 - Referred to as 0-RTT

Session Key Generation



Separate Keys for the different directions

- ▶ Counters, IV etc. can be selected independently

Client to Server Keys

Server to Client Keys

K_{xy} : Session key for AEAD ciphers

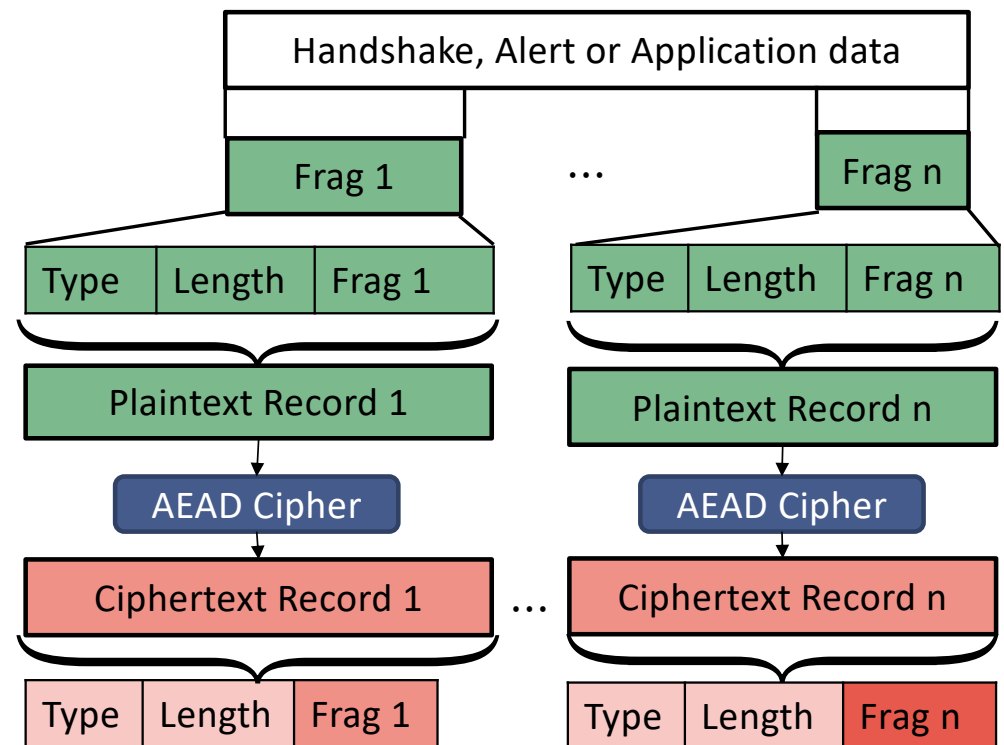
TCP Payload Protection with the TLS Record Protocol

- **The record protocol is responsible for**

- ▶ Taking messages to be transmitted and fragmenting data into blocks of 2^{14} bytes or less
 - Called TLS Plaintext records
- ▶ Protecting the records and transmitting them
- ▶ Verifying integrity protection on received data, decrypting received data
- ▶ Reassembling and delivering data to higher layers

- **Supports three main content types for the plaintext records**

- ▶ handshake , application-data, alert



Alert protocol

- **Specifies two different types of alerts**
 - ▶ Closure alerts
 - Closure-notify: Notifies receiver that sender will close connection now, receiver should ignore any traffic received after this message
 - user-canceled
 - ▶ Error alerts
 - unexpected-message
 - bad-record-mac: MAC on record layer did not check out correctly
 - handshake-failure: parameters could not be agreed upon
 -

TLS and Certificate Validation

- **The TLS RFC itself only specifies that**
 - ▶ TLS servers and clients need to check that the signature provided in the Certificate.Verify message can be verified with the public key in the certificate
- **Verifying the certificates received additionally requires the receiver to**
 - ▶ Check if the root CA is trusted in the context of the application invoking TLS
 - ▶ Check that the identity included in the certificate corresponds to the identity of the server
 - ▶ Verifying the signatures on all certificates in the provided chain up to a trusted root certificate
 - ▶ Verifying that each certificate in the chain is currently valid and has not been revoked

Supported Algorithms in Handshake and Data Protection

- All Ciphers supported by TLS 1.3 are AEAD ciphers

Supported AEAD Ciphers
TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_CCM_SHA256
TLS_AES_128_CCM_8_SHA256

Overview

IPSec

- ▶ Main use case
- ▶ Security services offered
- ▶ Authentication and key agreement
- ▶ Payload or packet protection

TLS

- ▶ Main use case
- ▶ Security services offered
- ▶ Authentication and key agreement
- ▶ Payload protection

Comparison of the protocols

- ▶ Differences
- ▶ Communalities in mechanisms used
- ▶ Overlaps in use cases

Comparison of IPsec and TLS

IPsec	TLS
IP-packet level protection	Protection of TCP Segments
Host-to-host protection of IP communication	Transport layer protection invoked by a specific application
Application independent protection of communication between individual hosts or complete networks	Communication between browser and web server and other client/server-style applications
Can be transparent to end users; no need to understand / configure IPsec	Requires end users to check if certificate has been issued to desired server
Highly configurable; Can be restricted to protect IP packets to / from individual host as well as complete networks	Invoked by a specific application running between client and server for all traffic of this application
Authentication and key agreement based on two-sided authenticated Diffie-Hellman	Authentication and key agreement based on a server-side only or mutually authenticated Diffie-Hellman
Authentication can be based on secret keys or public/private key pairs	Authentication based on public / private key pair of server and optional public / private key pair of client, alternatively a pre-shared secret key can be used since TLS 1.3

Base Specifications and References

IPSec

- ▶ Internet Key Exchange Protocol IKEv2
 - Specified in RFC RFC 7296
- ▶ Security Architecture for IP
 - Specified in RFC 4301
- ▶ Encapsulating Security Payload Protocol ESP
 - Specified in RFC 4303
- ▶ Authentication Header Protocol AH
 - Specified in RFC 4302

TLS 1.3

- ▶ TLS 1.3 RFC 8446
- ▶ Includes the handshake, record layer, and alert protocols

Book Chapter

- ▶ W. Stallings, Cryptography and Network Security: Principles and Practice, 8th edition, Pearson 2022
 - Chapter 17: Transport-Level Security
 - Chapter 20: IP Security

Summary

- **IPSec offers encryption and integrity protection for IP packets**
- **IPSec supports two modes**
 - ▶ Transport mode for IP-packet protection directly between packet origin and final destination
 - ▶ Tunnel mode for protection of IP-packets involving intermediate nodes such as security gateways
- **IPSec comprises**
 - ▶ The ESP protocol for encryption and integrity protection of the payload of the protected packet
 - ▶ The AH protocol for integrity protection of the entire protected packet (including the header)
- **IKEv2 offers authentication and key agreement for IPSec**
 - ▶ Based on a secure authenticated Diffie-Hellman key exchange (provides key confirmation)
 - ▶ Key exchange can be authenticated with the help of signatures or message authentication codes
 - ▶ Also negotiates which traffic is going to be protected with which protocols and algorithms

Summary

- **TLS 1.3 offers**

- ▶ Server-side or mutual authentication between client and server
- ▶ Session key establishment
- ▶ Encryption and integrity protection of TCP segments

- **Handshake protocol in TLS 1.3**

- ▶ Based on ephemeral DH exchange and signatures
- ▶ Based on a pre-shared key alone
- ▶ Based on ephemeral DH and pre-shared-key

- **Record protocol in TLS 1.3**

- ▶ Supports only AEAD ciphers



IT-Security

Chapter 7: Security of Selected Classical Applications

E-Mail, DNS, Remote Login

Prof. Dr.-Ing. Ulrike Meyer



Overall Lecture Context

- **Many applications can be protected using TLS**
 - ▶ Most prominent example HTTP over TLS = HTTPS
 - ▶ Also FTPs, SIPs, SRTP ,...
- **Some distributed applications, however, cannot easily use TLS end-to-end**
 - ▶ **Email**: asynchronous, no handshake between sender and receiver possible
 - ▶ **DNS**: connectionless, runs on UDP, caching necessary for performance reasons
 - ▶ ...
- **Secure versions of some applications have been developed in parallel to the first TLS version**
 - ▶ SSH: secures one of the oldest internet applications, namely **remote login**

Overview

Email Security

- ▶ Email Architecture
- ▶ Threats
- ▶ End-to-end protection
 - PGP and S/MIME
- ▶ Backbone protection
 - SMTPs
 - ...

DNS Security

- ▶ DNS System
- ▶ Threats
- ▶ DNSSec
- ▶ DoT / DoH

Remote Login with SSH

- ▶ Primary use case
- ▶ Security services offered
- ▶ TCP payload protection

Overview

Email Security

- ▶ Email Architecture
- ▶ Threats
- ▶ End-to-end protection
 - PGP and S/MIME
- ▶ Backbone protection
 - SMTPs
 - ...

DNS Security

- ▶ DNS System
- ▶ Threats
- ▶ DNSSec
- ▶ DoT / DoH

Remote Login with SSH

- ▶ Primary use case
- ▶ Security services offered
- ▶ TCP payload protection

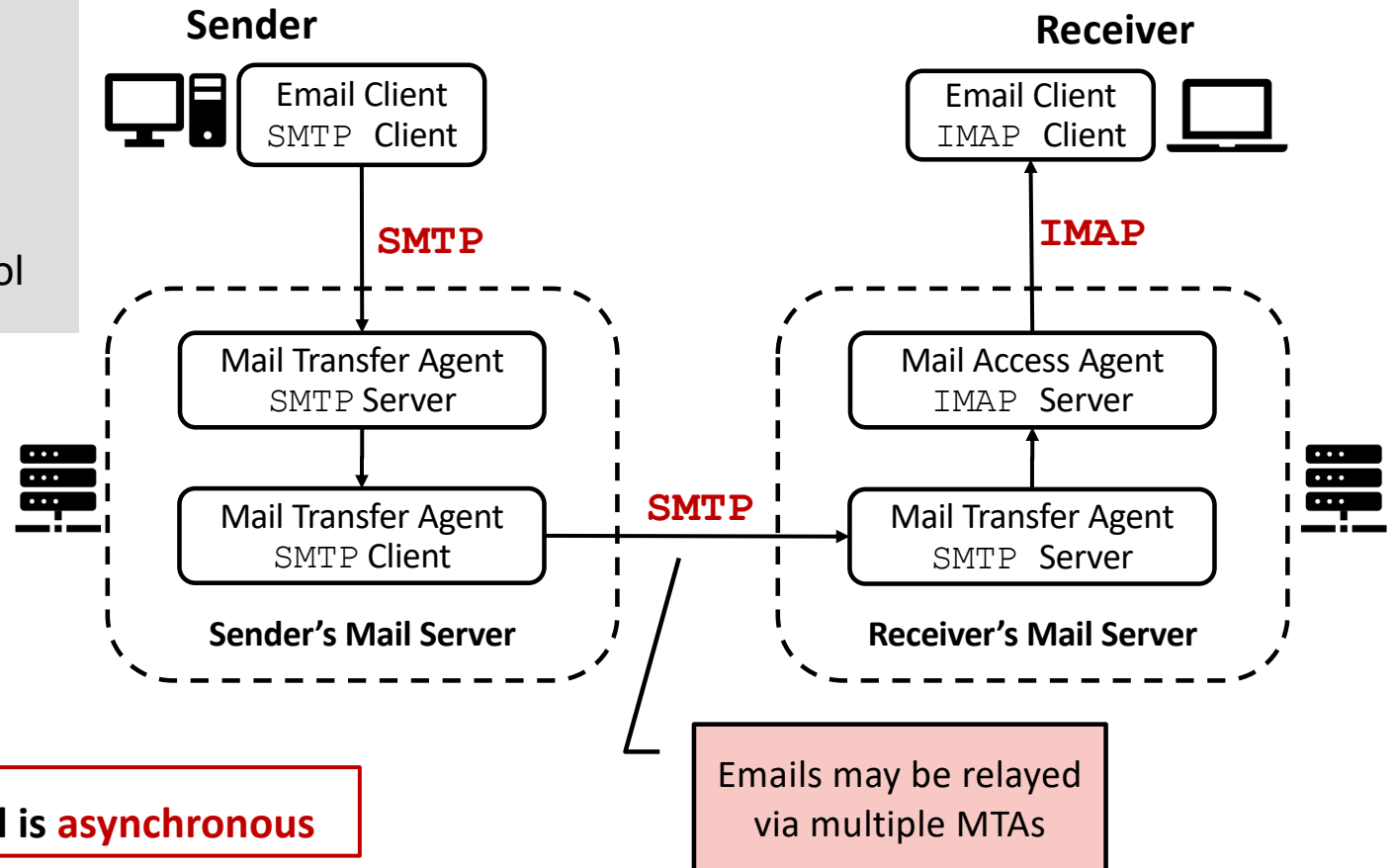
Classical Email Architecture (Simplified)

SMTP :

Simple Mail Transfer Protocol

IMAP :

Internet Message Access Protocol



Note: sending /receiving email is asynchronous

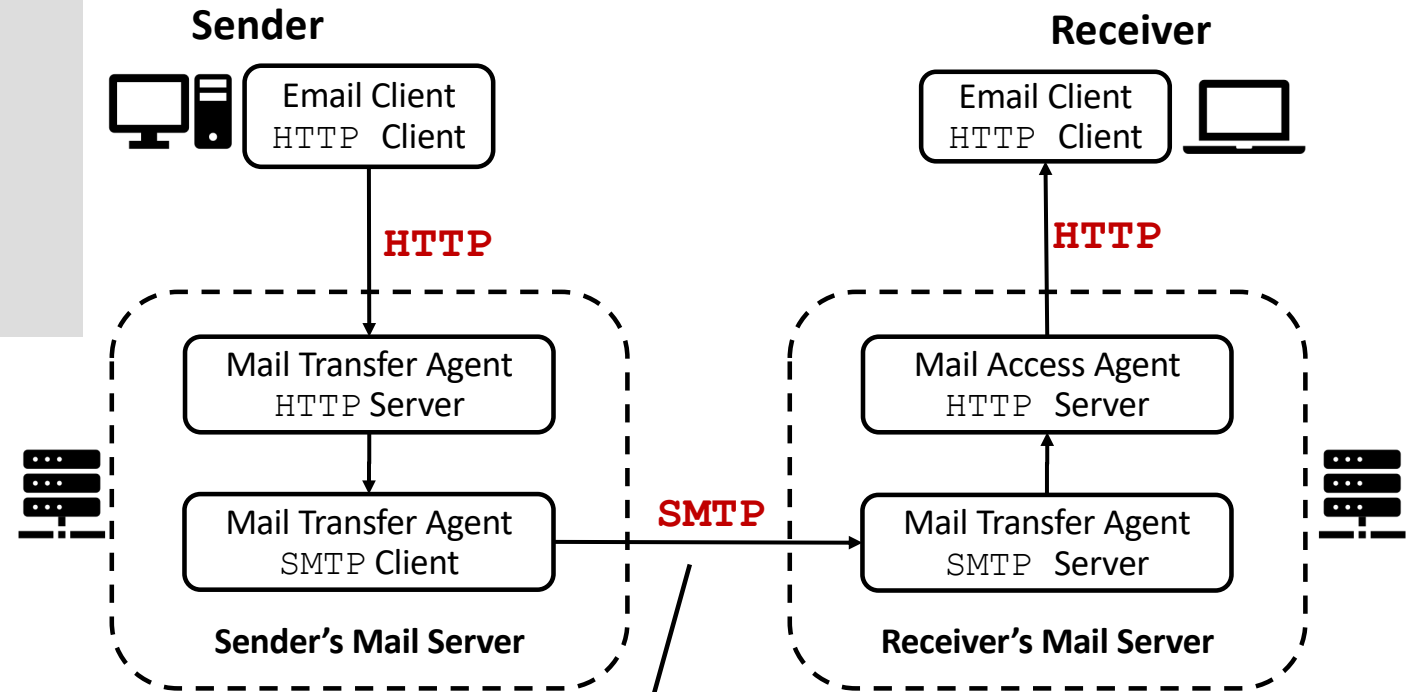
Alternative: Web Email Architecture (Simplified)

SMTP :

Simple Mail Transfer Protocol

HTTP :

Hypertext Transfer Protocol



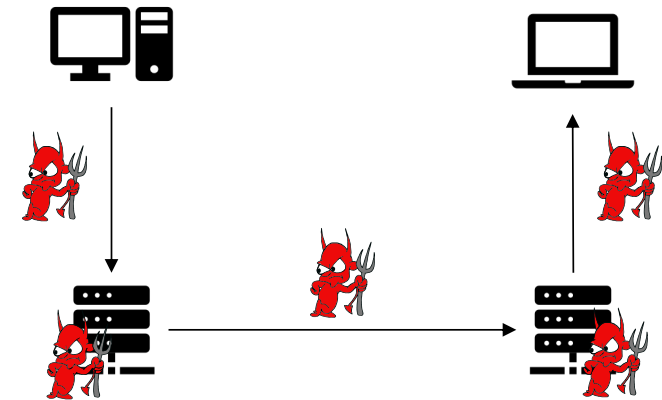
HTTP used to send emails to and receive emails from Mail servers

Emails may be relayed via multiple MTAs

Email Threats

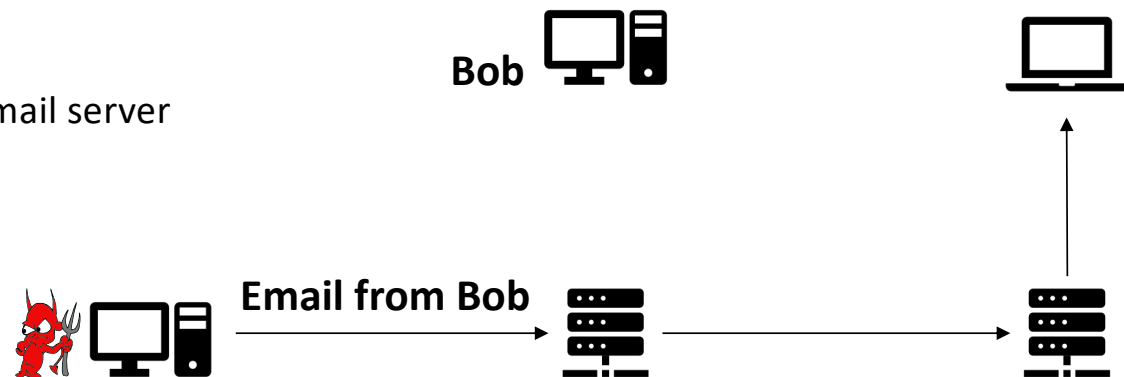
Eavesdropping and Manipulation

- ▶ During transfer
 - Between email clients and mail servers, between mail servers
- ▶ On storage at mail server
 - Emails stored in cleartext



Email Spoofing

- ▶ Attacker submits an email to some mail server
- ▶ Claims the email is from Bob



Protecting Emails with TLS

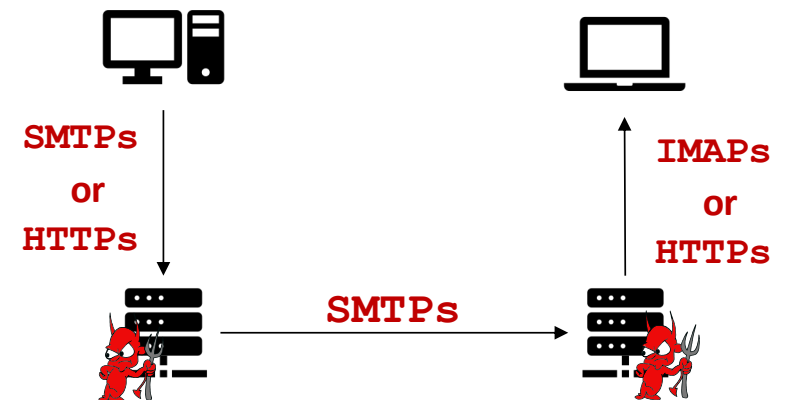
- **TLS allows us to protect TCP connections**

- ▶ **SMTPs**: SMTP over TLS
 - protects email transfer from sender to email server
 - protects email transfer between email servers
- ▶ **IMAPs**: IMAP over TLS
 - protects email transfer from email server to receiver
- ▶ **Alternatively: HTTPs** HTTP over TLS
 - protects email transfer from sender to email server
 - protects email transfer from email server to receiver

- **Hop-by-hop protection of confidentiality and integrity**

- **No non-repudiation**

- **Emails still stored in the clear on mail server**



End-to-end protection with TLS not possible

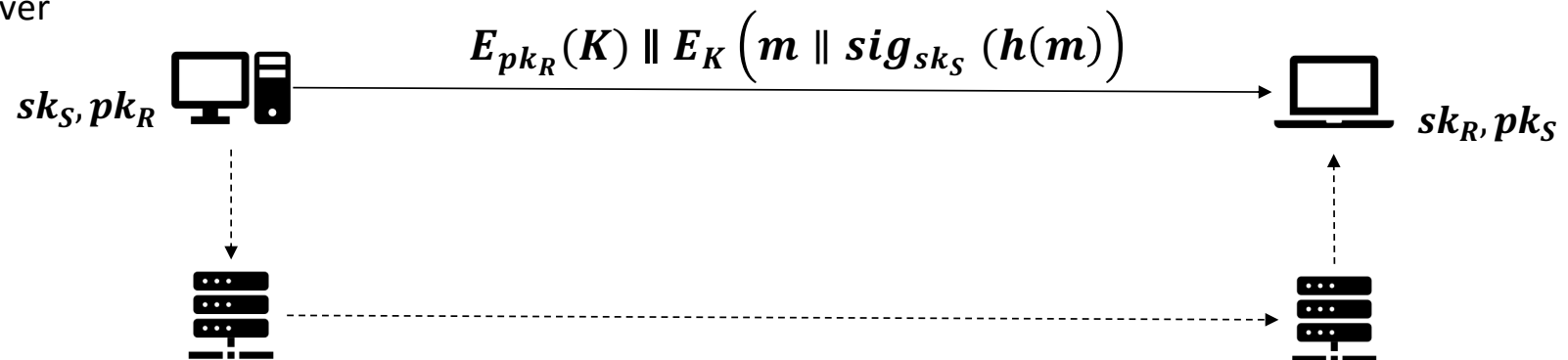
End-to-End Protection

Approach used by S/MIME and PGP

- ▶ Signs hash of message m with own private key sk_S
- ▶ Sender generates symmetric key K
- ▶ Encrypts mail and signature with K
- ▶ Encrypts K with receiver's public key pk_R
- ▶ Sends encrypted message and encrypted key as mail to its mail server

Threats covered

- ▶ **Eavesdropping** – symmetric encryption
- ▶ **Manipulation** – digital signature
- ▶ **Repudiation** – digital signature



Main Conceptual Difference between S/MIME and PGP

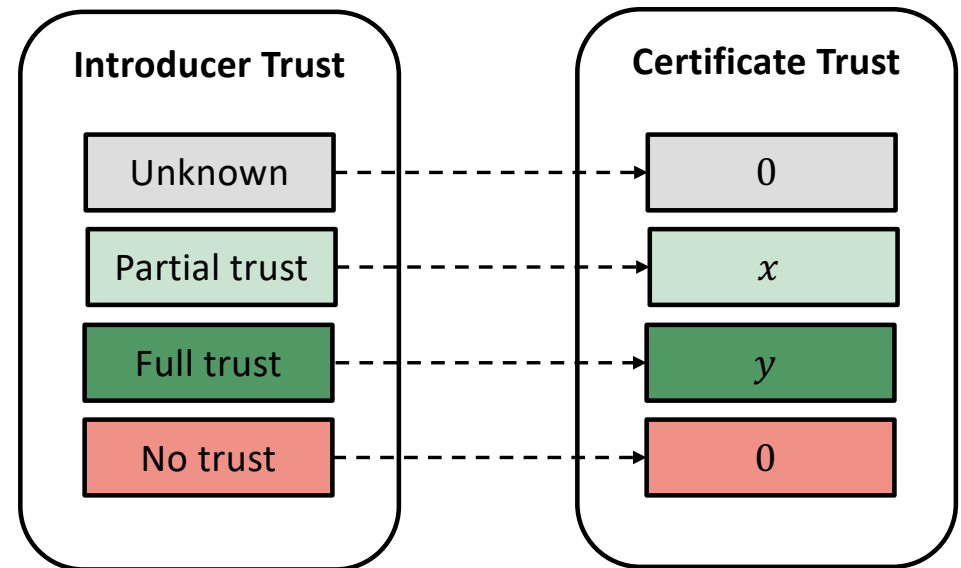
Distribution of public keys

- ▶ S/MIME: certificates signed by CAs, Trusted CAs configured in email client
- ▶ PGP: Web of trust
 - PGP users sign certificates for other PGP users
 - Each user decides which keys to trust

PGP Web of Trust

- Each PGP user

- ▶ assigns **introducer trust level** to other users
- ▶ assigns **certificate trust level** $0 \leq w \leq 1$ to each introducer trust level



User	Introducer Trust
Clare	Partial trust
Dave	No trust
Tom	Partial trust
Fred	Full trust

Certificate	Certificate Trust level
$Cert(pk_{Bob})_{Clare}$	x
$Cert(pk_{Bob})_{Dave}$	0
$Cert(pk_{Bob})_{Tom}$	x

Key Legitimacy

Key legitimacy is computed from certificate trust levels

- ▶ Let N_x (N_y) be the number of certificates of certificate trust value x (y)
- ▶ Then the key legitimacy is computed by

$$\text{key legitimacy} = \begin{cases} 1 & \text{if } x \cdot N_x + y \cdot N_y \geq 1 \\ 0 & \text{if } x \cdot N_x + y \cdot N_y < 1 \end{cases}$$

Example with $x = \frac{1}{2}$ and $y = 1$

User	Introducer Trust	Certificate	Certificate Trust level	public key	Key legitimacy
Clare	Partial trust	$Cert(pk_{Bob})_{Clare}$	1/2	pk_{Bob}	1
Dave	No trust	$Cert(pk_{Bob})_{Dave}$	0	pk_{Ted}	0
Tom	Partial trust	$Cert(pk_{Bob})_{Tom}$	1/2	pk_{Alf}	1
Fred	Full trust	$Cert(pk_{Ted})_{Dave}$	0		
		$Cert(pk_{Alf})_{Fred}$	1		

Threats covered

Hop-by-hop protection with TLS	End-to-end with S/MIME or PGP
Eavesdropping on transfer – Symmetric encryption	Eavesdropping on transfer and in storage – Symmetric encryption
Manipulation on transfer – MACs	Integrity protection on transfer and in storage – Digital Signature
	Non-repudiation – Digital Signature

But: mail servers and mail clients still accept unprotected messages

- Email spoofing still possible und extensively used e.g. in phishing attacks

Typical SMTP Exchange between Client and Server

- SMTP commands / responses

- Email message

- Email header lines

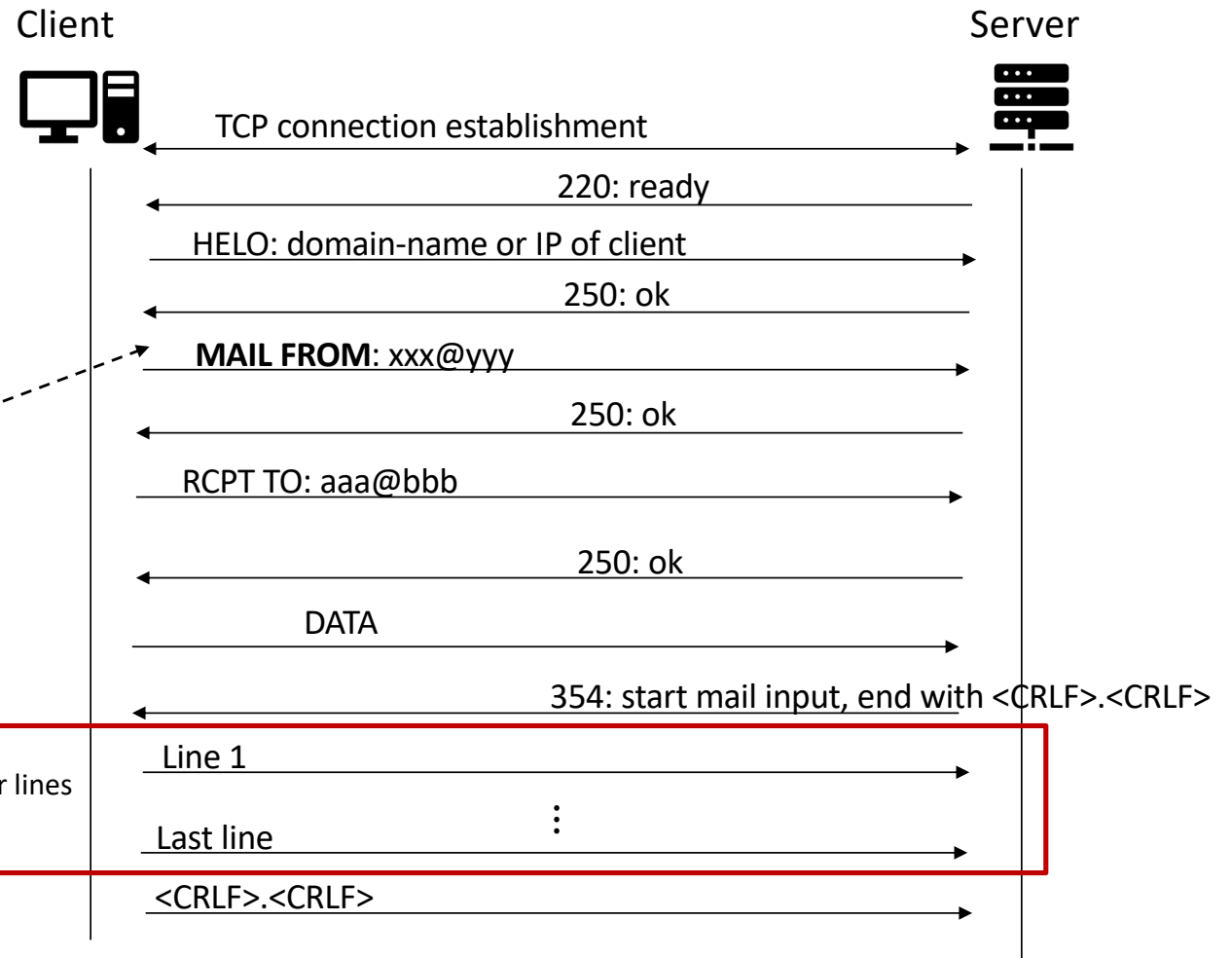
- From header line
 - To header line
 - ...

- Email Body

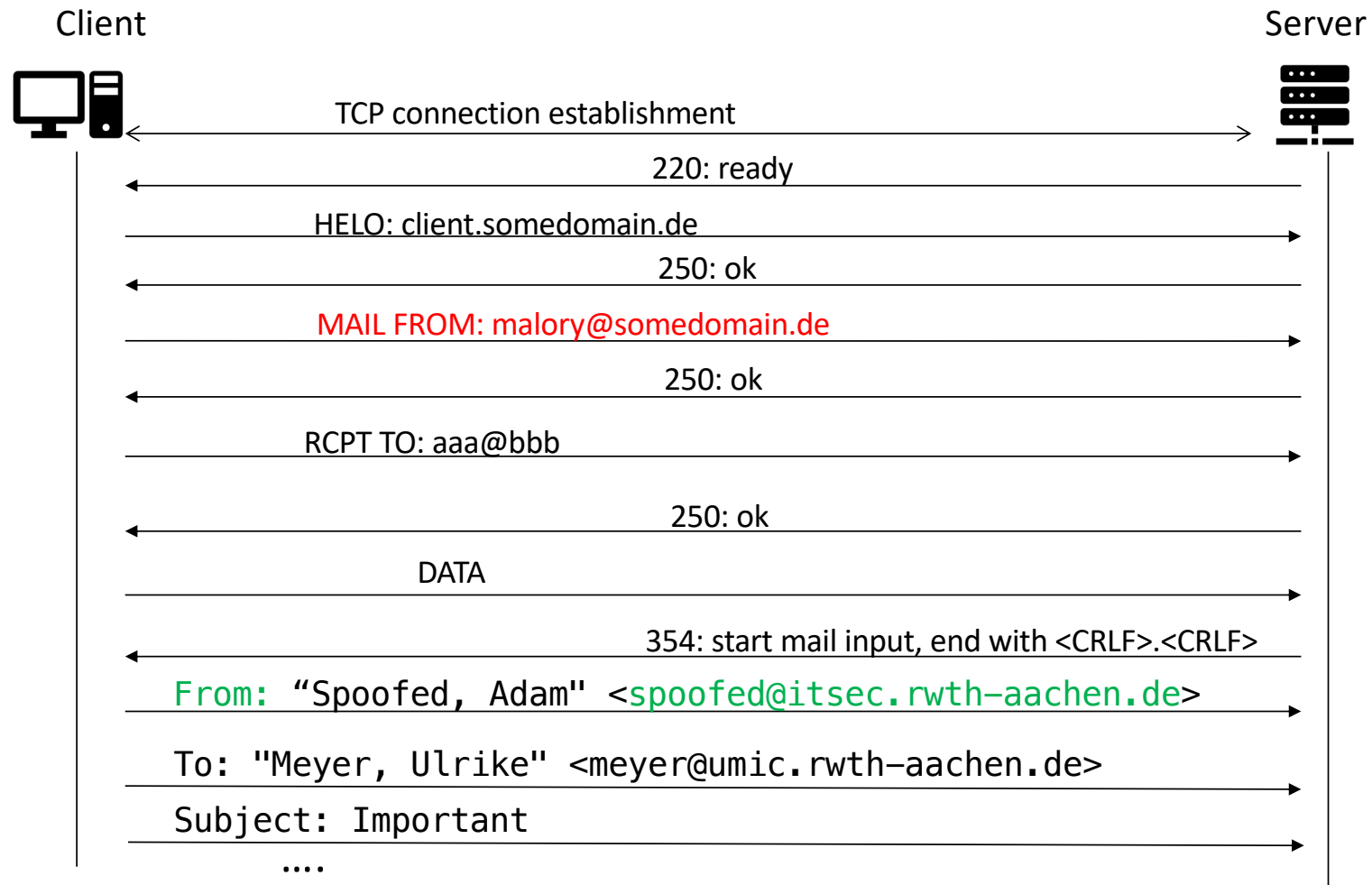
- MAIL FROM and From header

line may differ

Email
Starting with header lines
Including
From, To, CC...



Example: Spoofed From Header Line



Further Protocols between Mail Servers

Domain Key Identified Mail

- ▶ Mail server signs mail header lines and body
- ▶ Thereby "authenticates" from header line
- ▶ Signature checked by receiving mail server
- ▶ Allows receiving mail server to discard unsigned messages with from email address of domain that is known to be signing

Sender Policy Framework

- ▶ Allows to specify hosts that are allowed to send mail on behalf of the domain
 - In HELO and MAIL FROM SMTP commands
- ▶ Emails from other SMTP clients trying to send email on behalf of the domain can then be blocked

DMARC

- ▶ Complements SPF and DKIM by a DNS-based mechanism to distribute policies on
 - How emails claiming to come from a domain are to be handled
 - How to receive reports on domain abuse

Threat to Availability: SPAM

- **SPAM stands for “SPiced hAM”**

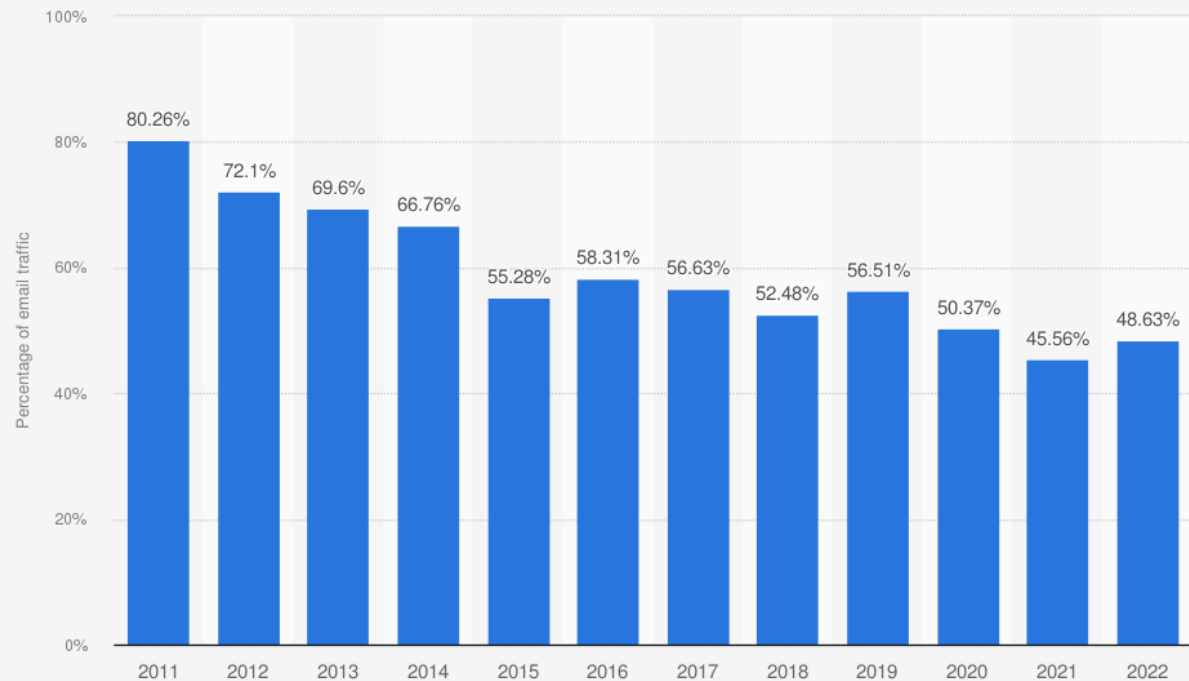
- ▶ Any unsolicited commercial email is SPAM



- **Sent by attackers by**

- ▶ Directly connecting to recipient’s email server
- ▶ Using open email relays
- ▶ Using malware-infected client machines

Global spam volume as percentage of total e-mail traffic from 2011 to 2022



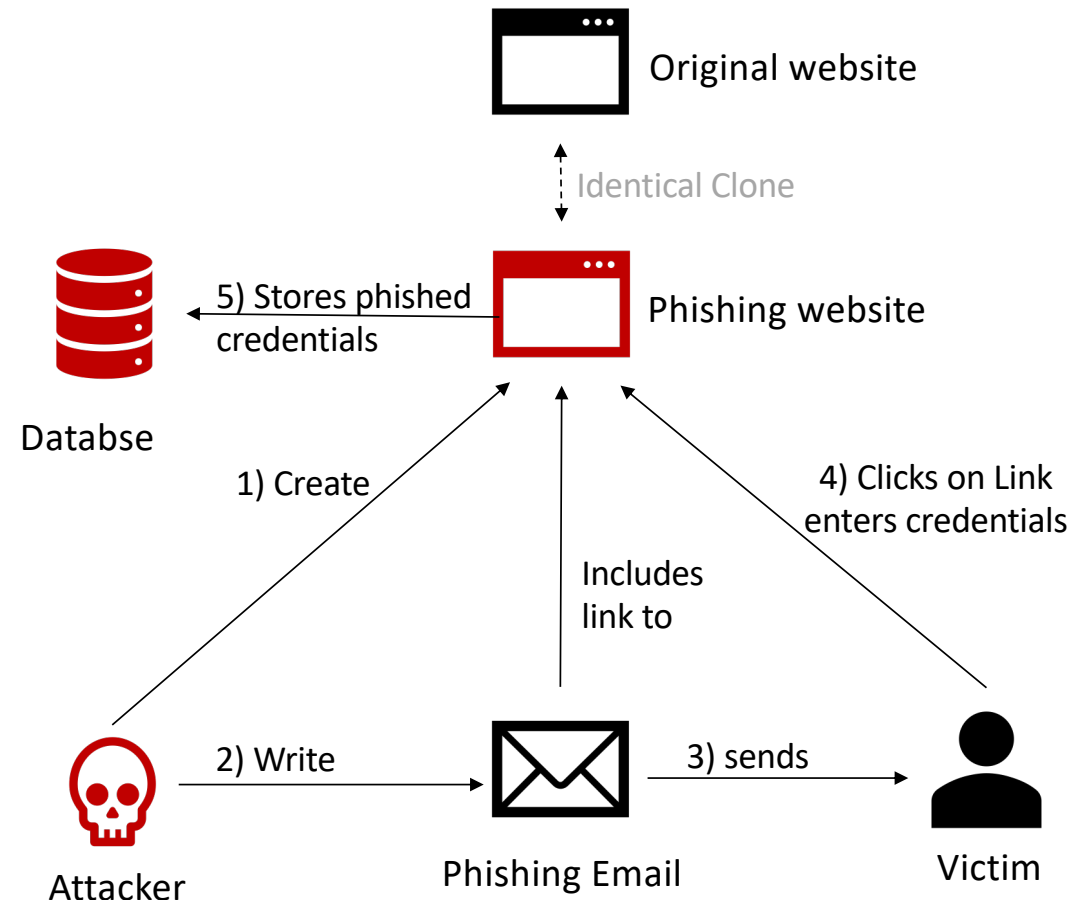
Source
Kaspersky Lab
© Statista 2023

Additional Information:
Worldwide; Kaspersky Lab; 2011 to 2022

<https://www.youtube.com/watch?v=duFierM1yDg>

Email Spoofing in Classical Phishing Attacks

- In a phishing attack and attacker tries to
 - ▶ lure a user into revealing private information
- Classical attack path
 - ▶ Attacker sends phishing email to victim, **often with spoofed sender address** to lure user into trusting it
 - ▶ Email includes link to phishing website
 - ▶ Phishing website is a clone of an original website
 - ▶ User enters private information into phishing website believing it's the original website
 - ▶ Phishing websites sends private information to a database controlled by the attacker
- What private information?
 - ▶ E.g. username / password, credit card number,...



Overview

Email Security

- ▶ Email Architecture
- ▶ Threats
- ▶ End-to-end protection
 - PGP and S/MIME
- ▶ Backbone protection
 - SMTPs
 - ...

DNS Security

- ▶ DNS System
- ▶ Threats
- ▶ DNSSec
- ▶ DoT / DoH

Remote Login with SSH

- ▶ Primary use case
- ▶ Security services offered
- ▶ TCP payload protection

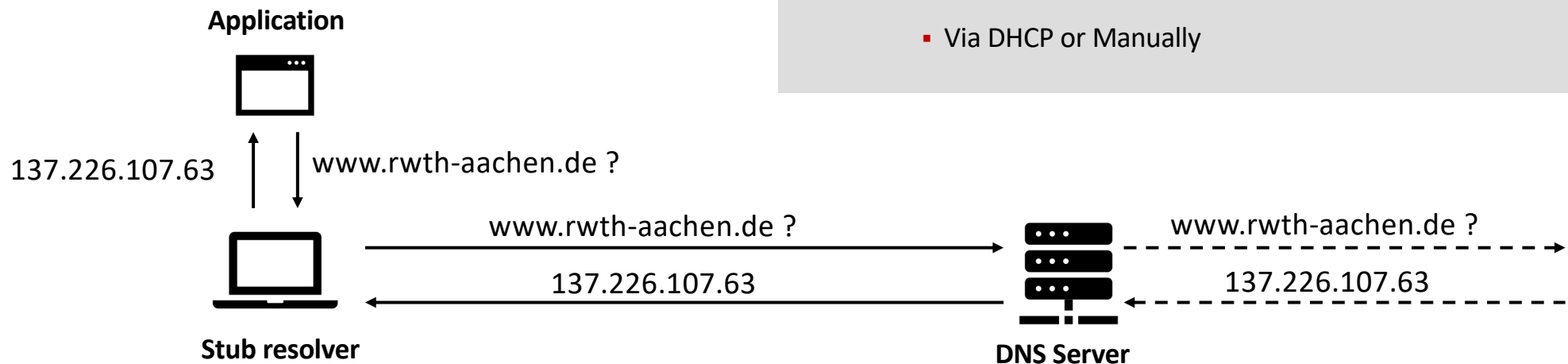
DNS System – Overview

Main purpose:

Map **domain names** to **IP addresses**

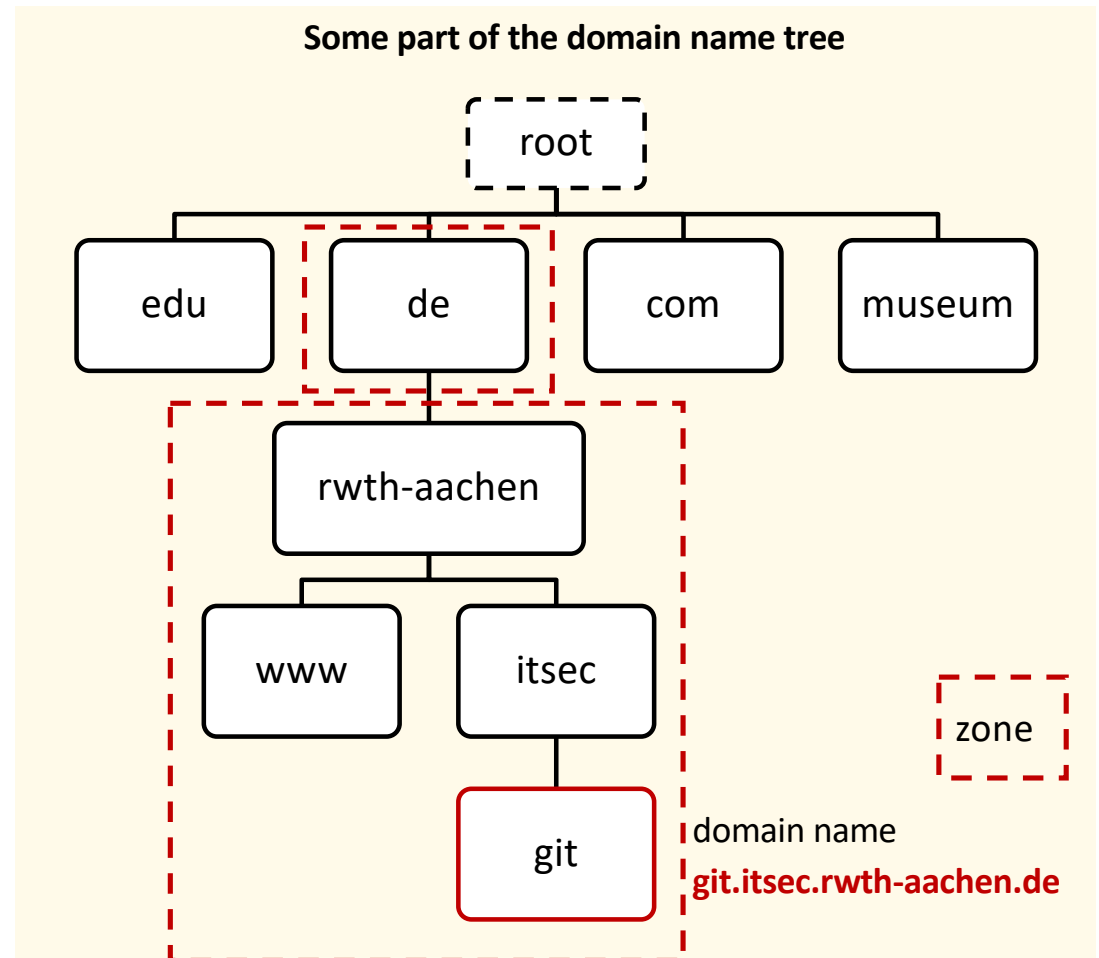
One of the oldest Internet applications (1987)

- ▶ Consists of thousands of DNS servers
- ▶ DNS servers can be queried by applications
 - via DNS client in the operating system: stub resolver
- ▶ Each host has a DNS server preconfigured
 - Via DHCP or Manually



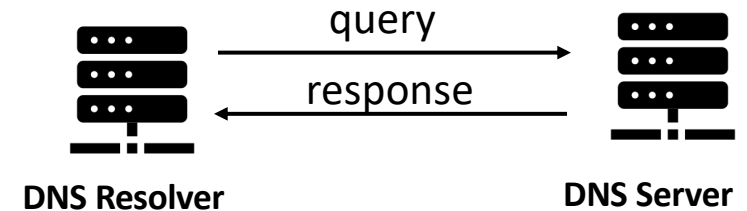
DNS System – Domain Name Space

- **Tree-structured name space**
 - ▶ starting from unnamed root domain
- **Over 1400 top-level domains: TLDs**
 - ▶ generic: e.g., .org, .edu, .mil, .com,...
 - ▶ country: e.g., .de, .uk, .es, .cn, ...
 - ▶ new TLDs: e.g., .tourism, .museum,...
 - ▶ Internationalized domain name, e.g. .mockba
- **Leaf domains may refer to single hosts or a collection of hosts**
- **Zone**
 - ▶ Connected part of the domain name space
 - ▶ child does not need to belong to zone of parent



DNS Queries and Responses

- A DNS resolver sends a DNS query to a DNS Server
- A DNS server answers with a DNS response
- Queries and responses use the same format



Query ID: 16 bit, same in query and response	Flags
# Questions	# RRs in Answer Section
# RRs in Authority Section	# RRs in Additional Information Section
Question(s): domain name and type of answer desired	
Answer section: RRs answering the question	
Authority section: RRs of name servers responsible for domain name in answer	
Additional information section: additional RRs, e.g., IP address of name servers	

In practice only 1 question per query is supported

```
dig www.tu-darmstadt.de
```

DNS clients are called resolvers!

RR stands for resource record

Resource records

- DNS distributes information on domain names as RRs

- Structure of RRs

domain name	time-to-live	class	type	value
-------------	--------------	-------	------	-------

- **domain name**

- ▶ to which the RR applies

- **TTL in seconds**

- ▶ indicates how long RR should be cached

- **class:** IN for Internet

Type	Meaning	Value
A	IPv4 address of a host	32 bit
AAAA	IPv6 address of a host	128 bit
MX	Mail exchange	Domain name of mail server accepting email for this domain
NS	Name Server	Domain name of an authoritative name server for this domain
CNAME	Canonical Name	Maps the domain name (alias) to an other domain name (canonical name)
PTR	Pointer	Used mainly for reverse lookups
SOA	Start of authority	Administrative information on a zone

Example

```
MacBook-Pro:~ uli$ dig www.tu-darmstadt.de

; <<>> DiG 9.10.6 <<>> www.tu-darmstadt.de
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14179
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.tu-darmstadt.de.          IN      A

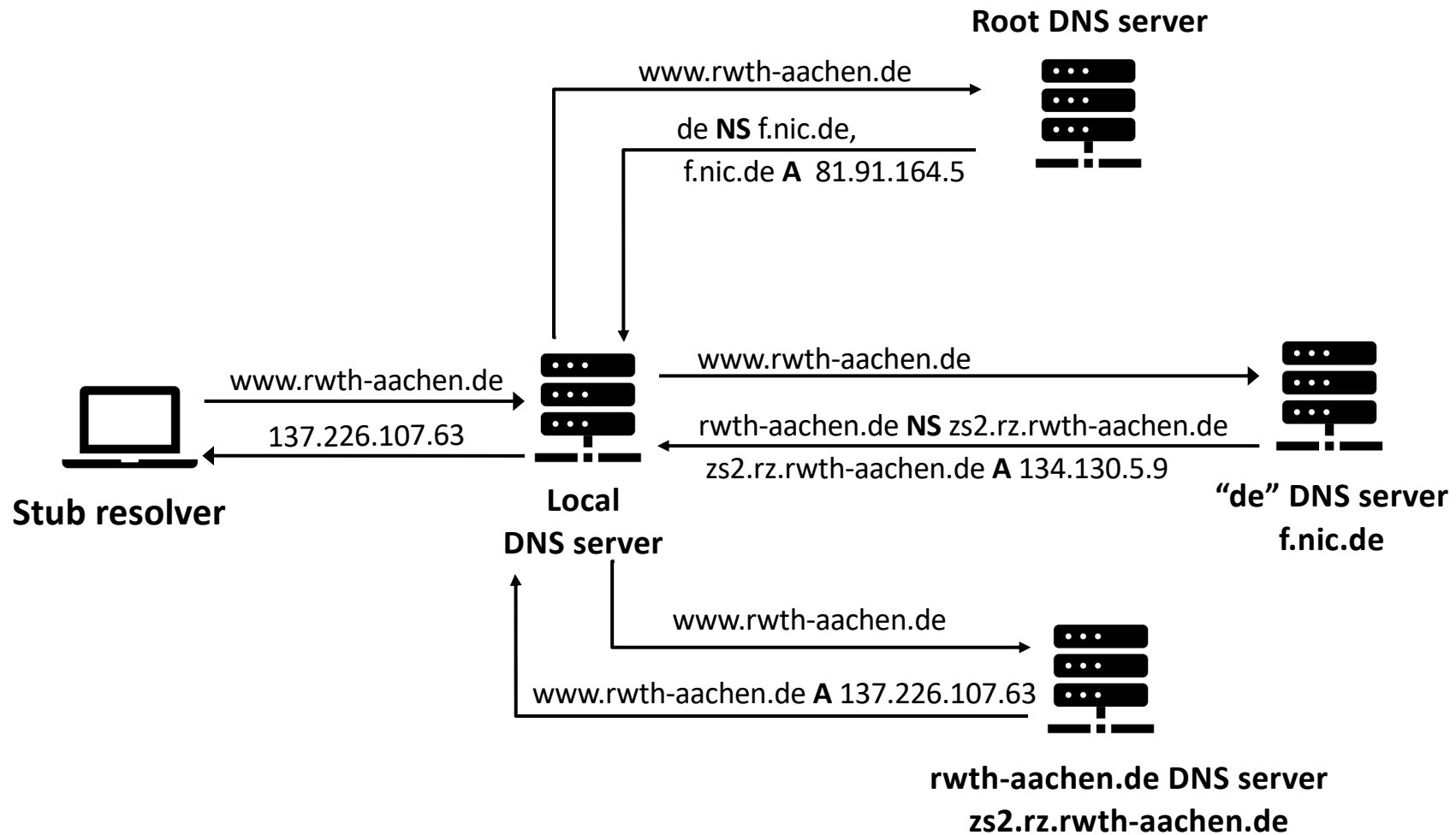
;; ANSWER SECTION:
www.tu-darmstadt.de.  3444   IN      CNAME   cms-sip02.hrz.tu-darmstadt.de.
cms-sip02.hrz.tu-darmstadt.de. 68489 IN A      130.83.47.181

;; AUTHORITY SECTION:
hrz.tu-darmstadt.de.  18179  IN      NS       ans1.net.hrz.tu-darmstadt.de.
hrz.tu-darmstadt.de.  18179  IN      NS       ans2.net.hrz.tu-darmstadt.de.

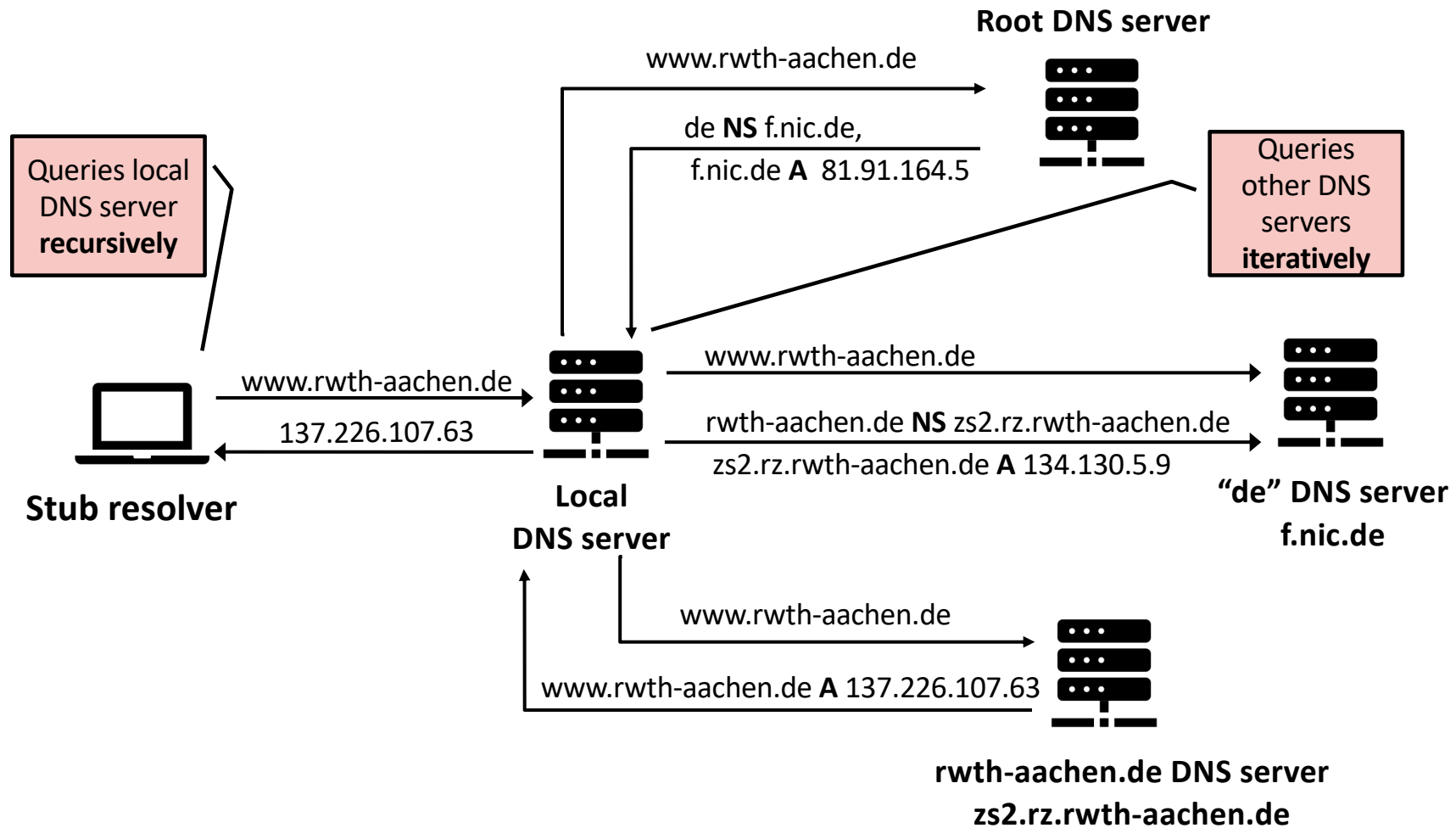
;; ADDITIONAL SECTION:
ans1.net.hrz.tu-darmstadt.de. 47911 IN A      130.83.22.61
ans2.net.hrz.tu-darmstadt.de. 47911 IN A      130.83.56.61
ans1.net.hrz.tu-darmstadt.de. 47911 IN AAAA   2001:41b8:83f:22::61
ans2.net.hrz.tu-darmstadt.de. 47911 IN AAAA   2001:41b8:83f:56::61

;; Query time: 56 msec
;; SERVER: 134.130.4.1#53(134.130.4.1)
;; WHEN: Thu Jun 01 12:16:29 CEST 2023
;; MSG SIZE rcvd: 222
```

Example: Typical Address Resolution

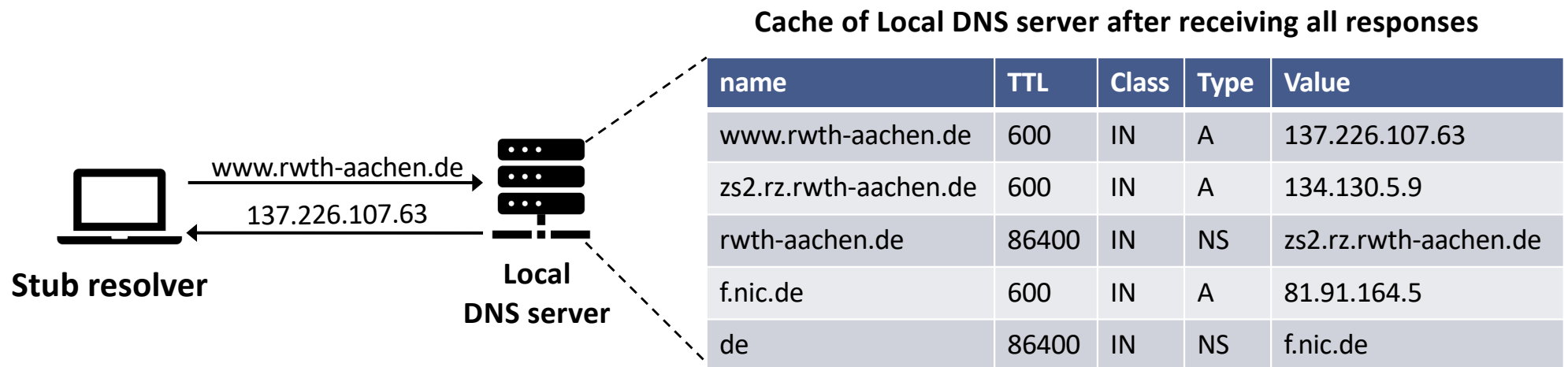


Example: Typical Address Resolution



Caching at DNS Servers

DNS servers cache all RRs they learn from responses for as TTL seconds



Caching accelerates future queries

- ▶ same queries as well as queries that may reuse part of the information obtained

DNS Threats Overview

- **Threats to availability**

- ▶ Flood DNS server with fake queries or responses
- ▶ Thereby make it unresponsive for legitimate queries



- **Threats to integrity such as**

- ▶ Provide incorrect RRs to resolvers
 - E.g., by making DNS servers cache fake RRs: **Cache Poisoning Attacks**
 - E.g., by making client machines connect to malicious DNS servers

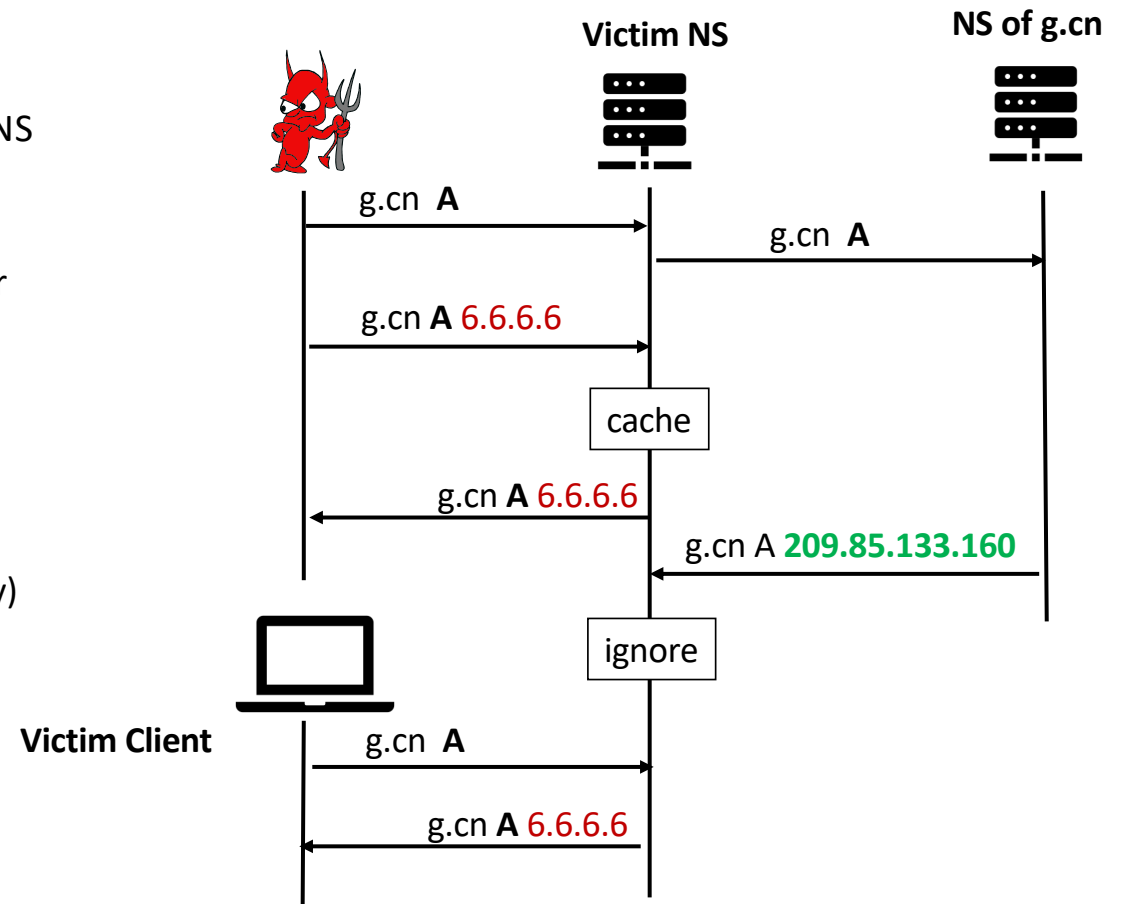
- **Threats to confidentiality**

- ▶ DNS queries are unencrypted
 - Anyone eavesdropping between client and local DNS server learns queries
 - Local DNS server learns queries

Example Attack Threatening Integrity: Cache Poisoning

Idea:

- ▶ Make victim DNS server query another DNS server for RR to be faked
- ▶ Send a fake response to victim DNS server
- ▶ Only successful if attacker can
 - spoof IP address of queried DNS server
 - guess correct query ID
 - (guess correct source port of victim's query)
 - be faster than real name server



DNSSec

- **Goal of DNSSec**

- ▶ Protect authenticity of resource records in an end-to-end fashion
- ▶ Enable distribution of authentic copies of public keys
- ▶ Still allow for caching
- ▶ Still allow DNS to run on top of UDP

- **Using TLS to protect authenticity of DNS RRs**

- ▶ Would require DNS to run on top of TCP
- ▶ Unnecesssary overhead
 - for one single round of query and response need TCP three-way handshake and TLS handshake
- ▶ Would make caching impossible as direct connection to authoritative name server required

DNSSEC New RRs and Keys

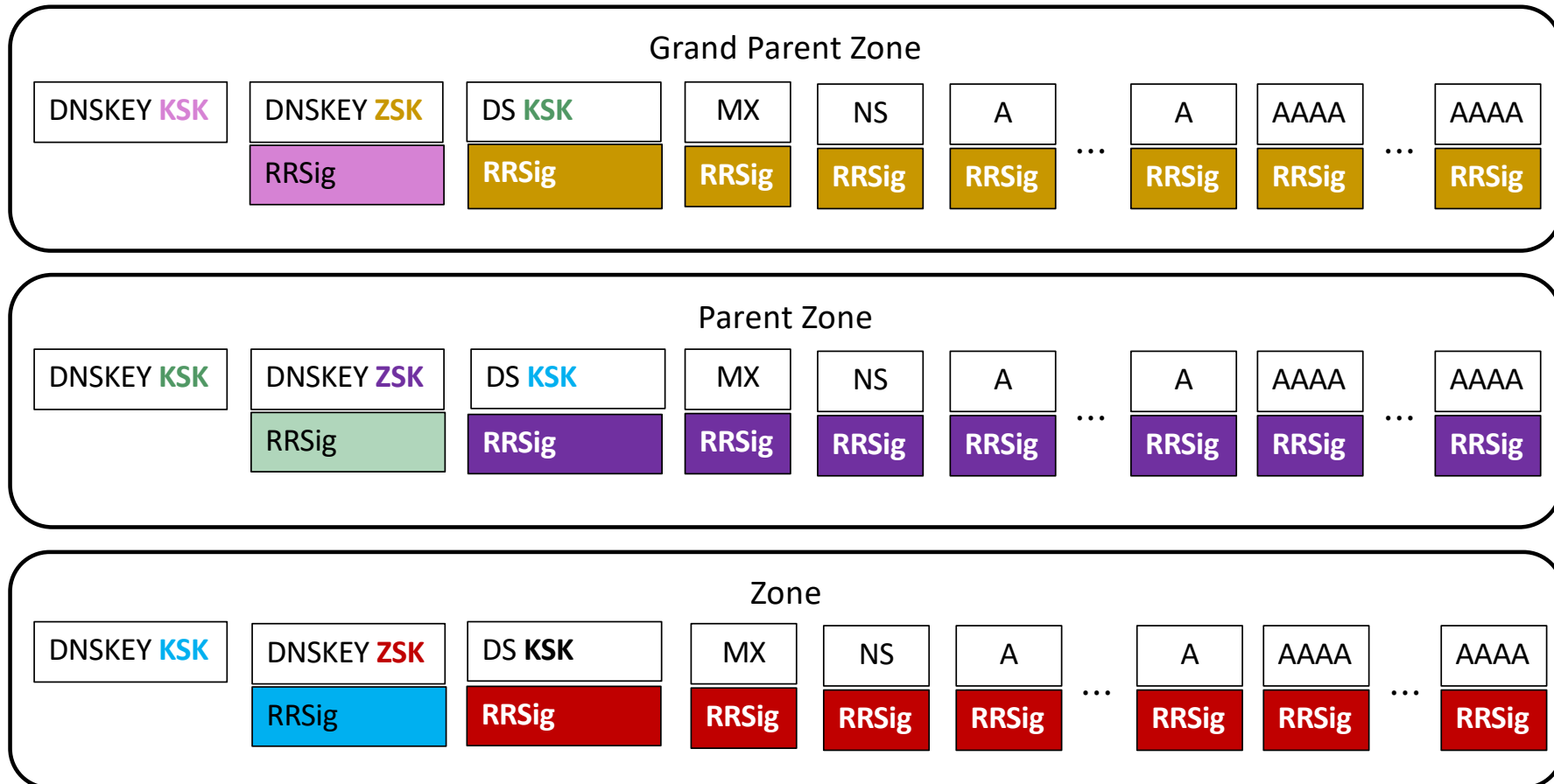
New RR types introduced by DNSSEC

- ▶ **RRSig RR:** used to carry signatures on a specific other RR referred to by domain name and type
- ▶ **DNSKEY RR:** used to carry public ZSKs and KSK of zones
- ▶ **DS RR:** (Delegation Signature RR): used to carry a hash of a KSK
- ▶ **NSEC RR:** used to help to authenticate negative DNS responses

Two types of keys used in DNSSEC

- ▶ **Zone Signing Key ZSK**
 - Used to sign resource records of a zone
 - Can be changed without involving parent zone
- ▶ **Key Signing Key KSK**
 - Used to sign DNSKEY RRs containing a ZSK
 - Distributed in DNSKEY RRs as well
 - Requires a DS RR in the parent domain and an RRSig RR that carries a signature on the DS RR signed with the zone's ZSK

Zone File with DNSSEC Resource Records



New RR Types (1)

- **DNSKEY RR contains**

- ▶ a public key of a zone
- ▶ Key flag that indicates if the key is a KSK (key flag 257) or a ZSK (key flag 256)
- ▶ **algorithm field** that indicates for which algorithm the key can be used

```
de.                4408      IN        DNSKEY    257 3 8
AwEAAbntyidABgdzt4jx+CVx8RgxEcJYdBFoihl3Ay87saAJsJXCVo6X
yGJWDHlgNFJrVzKL6ePIQ2vtNb/R4opICz1TTLB92MFiWJs6gKIBBHtx
z1+etiRAAWLgakExShzkmWmrFciMpTDIjNMEclpl4diuqgnnqiAtO4jw 97t/C69H ; key-id =39227
```

New RR Types (2)

- **DS RR (Delegation Signature RR) contains**

- ▶ **hash** of a KSK, indication of hash algorithm used
- ▶ A **key tag** that points to the KSK, an **algorithm field** that indicates the algo for which KSK can be used, algo identifier of **hash algo** used

```
de.           82597   IN      DS      39227 8 2  
AAB73083B9EF70E4A5E94769A418AC12E887FC3C0875EF206C3451DC 40B6C4FA
```

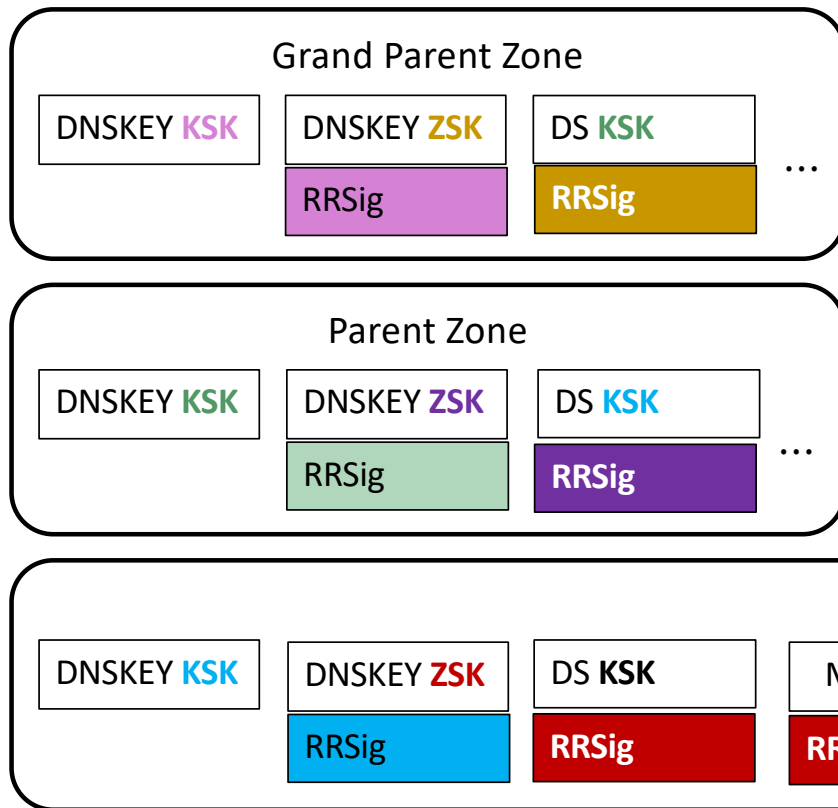
New RR Types (3)

- **RRSIG** RR contains

- ▶ a signature on a hash
- ▶ A **type covered** field that indicates the type of RR covered by the signature
- ▶ **Algorithm field**, that indicates the algorithm used
- ▶ A **key tag** that points to the key used to generate the signature
- ▶ Indication of the **validity period** of this signature, the **signer's name**
- ▶ **Original TTL** of signed RR

```
de.                5320    IN      RRSIG    DNSKEY 8 1 7200 20170601120000 20170511120000 39227 de.  
HMN5YPRBCKtSxWIR8/eW/3Kqy5AyVSbq/Zbx4fRKbSawf+v7rXHEqXnx  
CFS4DlaDWdOs0bOWLfMH748NW8YA1ZT6DSFKecuEkTHzLL4IbFzZcBgG  
KuJkp7wmiaamuW2PPGhutuqUcJ3CPy357OpuCJcT0qJh5nkkeURtZGj+  
gZCFNWjKIBvnLwDMGkFtdHsiW1DBUJA41KeF3Cbsvk9iJZnkxB+k6mlr  
a6A/G7+2fk6JuG4w7TBTfbBAa12VOHMzydwg2IMyweQOu3LssFR/WMF6  
h8k7SaOJhwE6WjhVuHhAUxsvN7TVPF9Ihb2FXMj1j0ckGwBZ4Y/SL7X /JLXhQ==
```

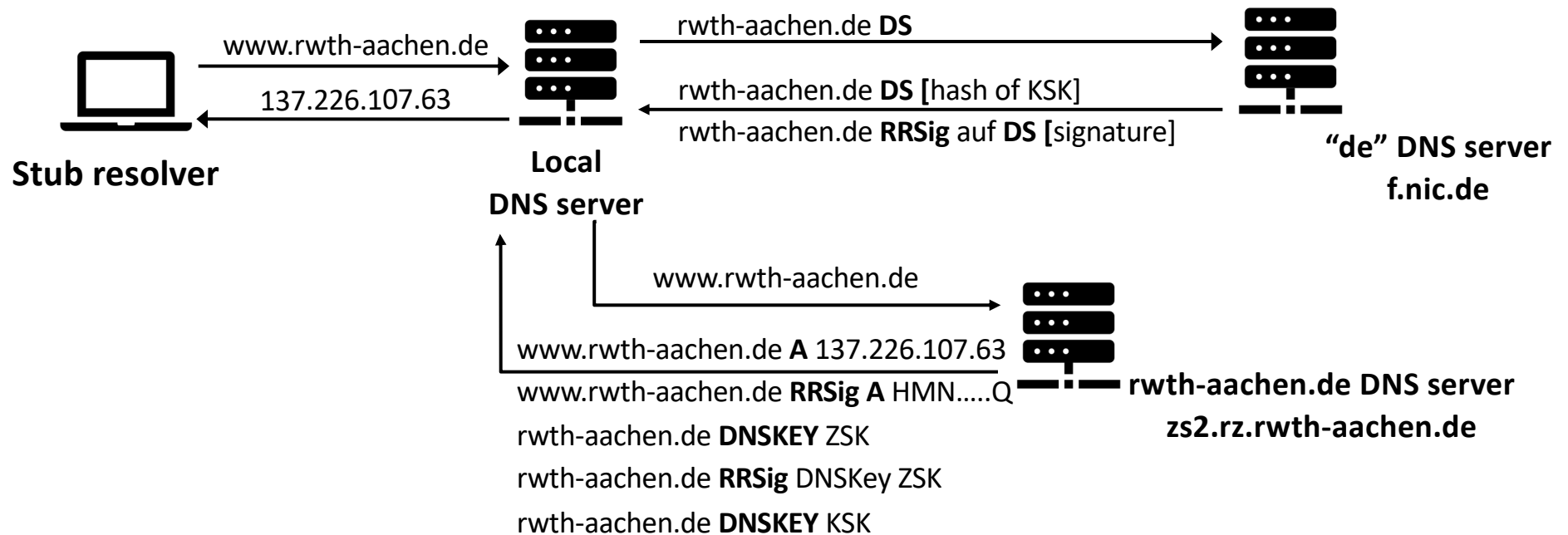
DNSSEC Signature Validation



- ▶ RR of zone trusted to be authentic if
- ▶ RRSig on RR with **ZSK** verifies correctly and **ZSK** trusted
- ▶ **ZSK** trusted if RRSig for **ZSK** verifies correctly and **KSK** is trusted
- ▶ **KSK** is trusted if RRSig for **KSK** verifies correctly and **ZSK** is trusted
- ▶ **ZSK** is trusted.... and **KSK** is trusted,
- ▶ **KSK** is trusted if **ZSK** is trusted ...
- ▶ ... until trusted root key is hit

Example: Typical Address Resolution with DNSec

Signatures are checked at the local DNS server; stub resolvers do often not support DNSec

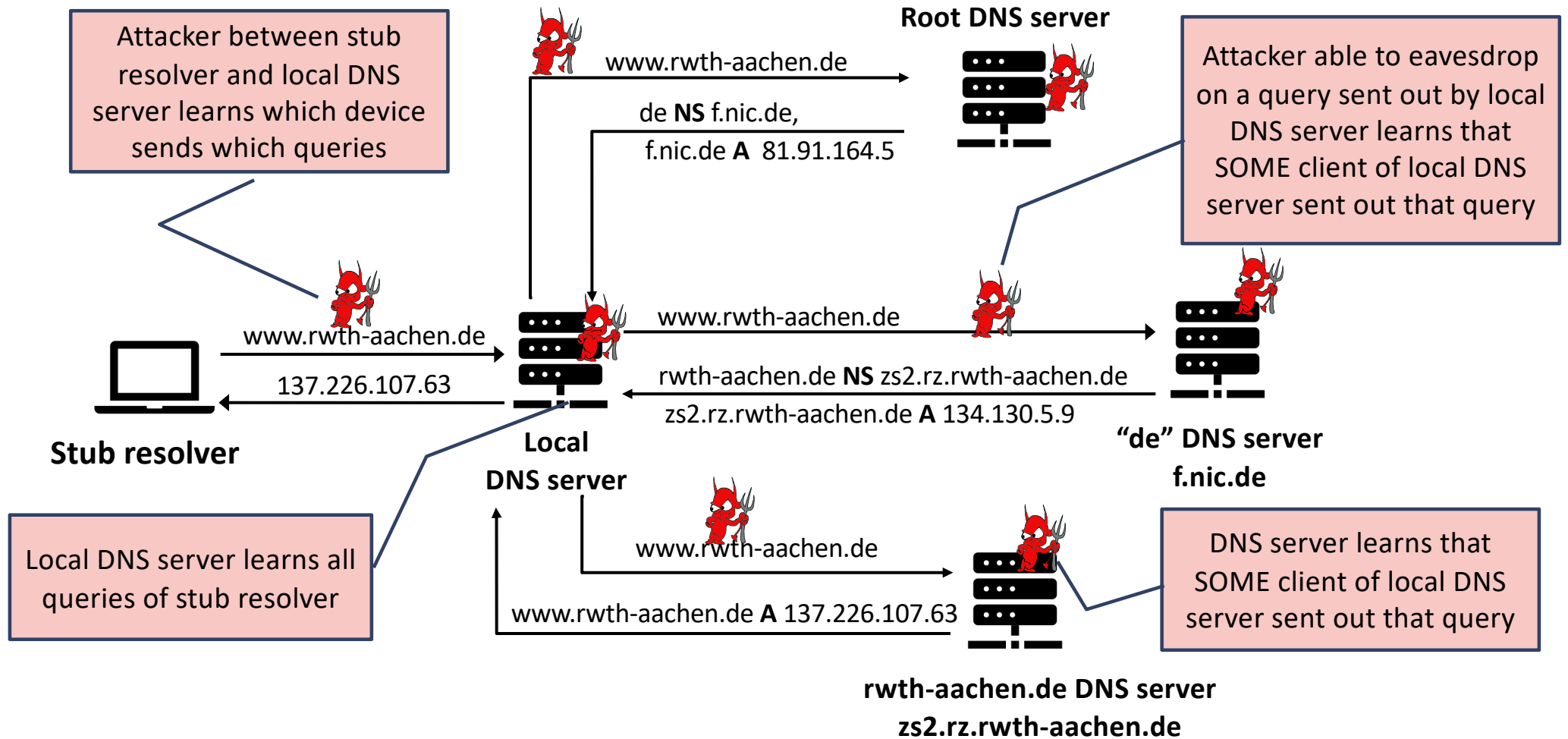


The DS record may also be cached by RWTH's name server and directly provided

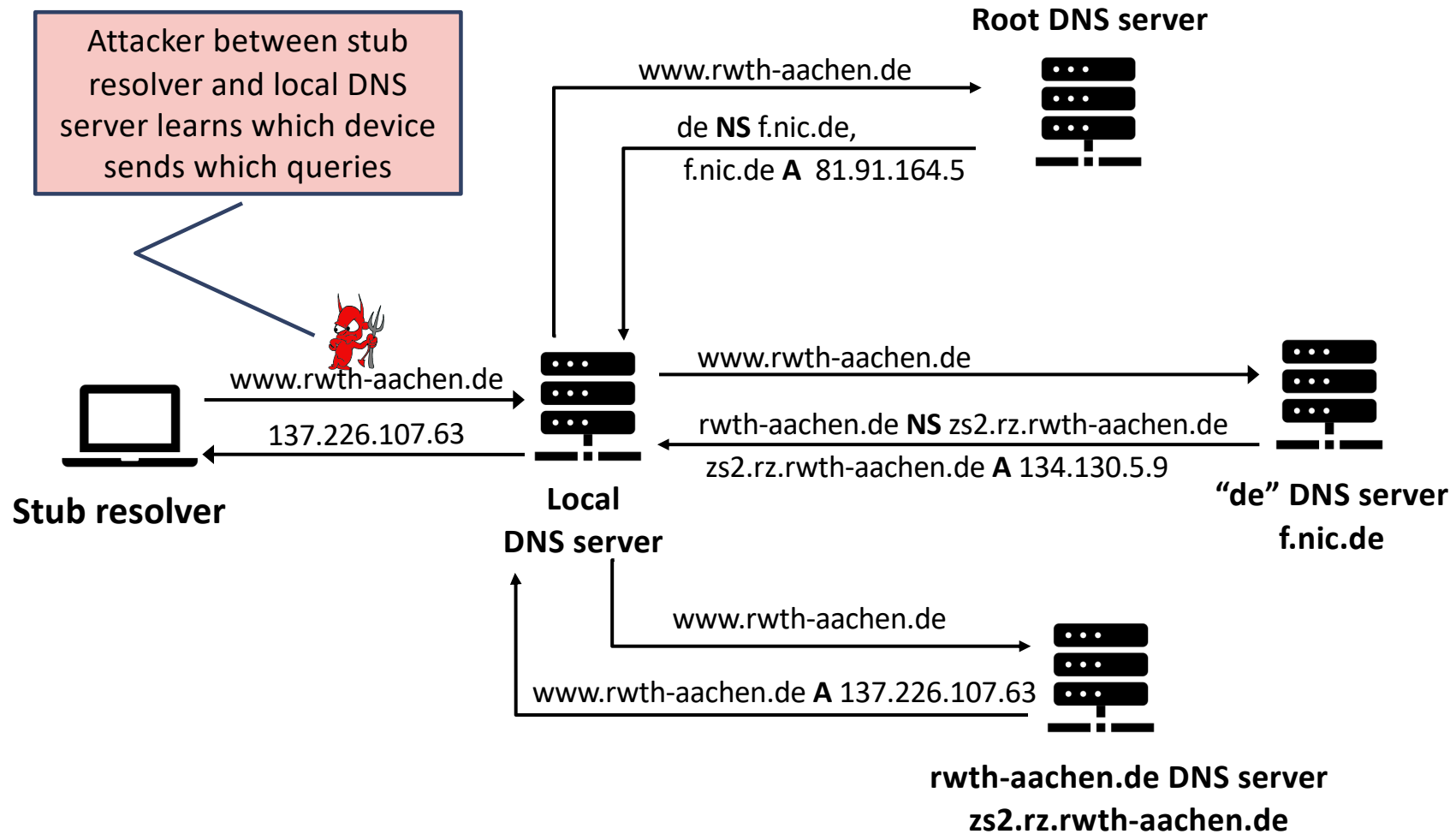
Confidentiality of DNS Queries

- **DNSSec only protects authenticity of DNS queries**
 - ▶ Implicitly assumes that DNS queries are not confidential
 - DNS responses only contain public information!
- **However, what someone queries and how reveals**
 - ▶ The websites he or she is interested in
 - ▶ Operating system used by the client
 - OS specific queries such as DNS queries to windows domains for updating
 - ▶ The anti-virus program you use
 - ▶ The productivity programs you use
 - Acrobat reader, all MS-office products etc. make specific DNS queries
 - ▶ ...

Example Threats against Confidentiality



Threat countered by DNS over TLS (DoT)



Overview

Email Security

- ▶ Email Architecture
- ▶ Threats
- ▶ End-to-end protection
 - PGP and S/MIME
- ▶ Backbone protection
 - SMTPs
 - ...

DNS Security

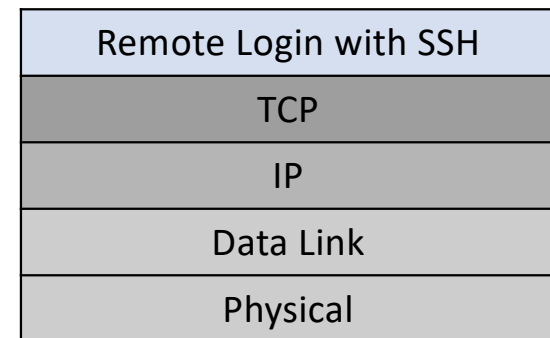
- ▶ DNS System
- ▶ Threats
- ▶ DNSSec
- ▶ DoT / DoH

Remote Login with SSH

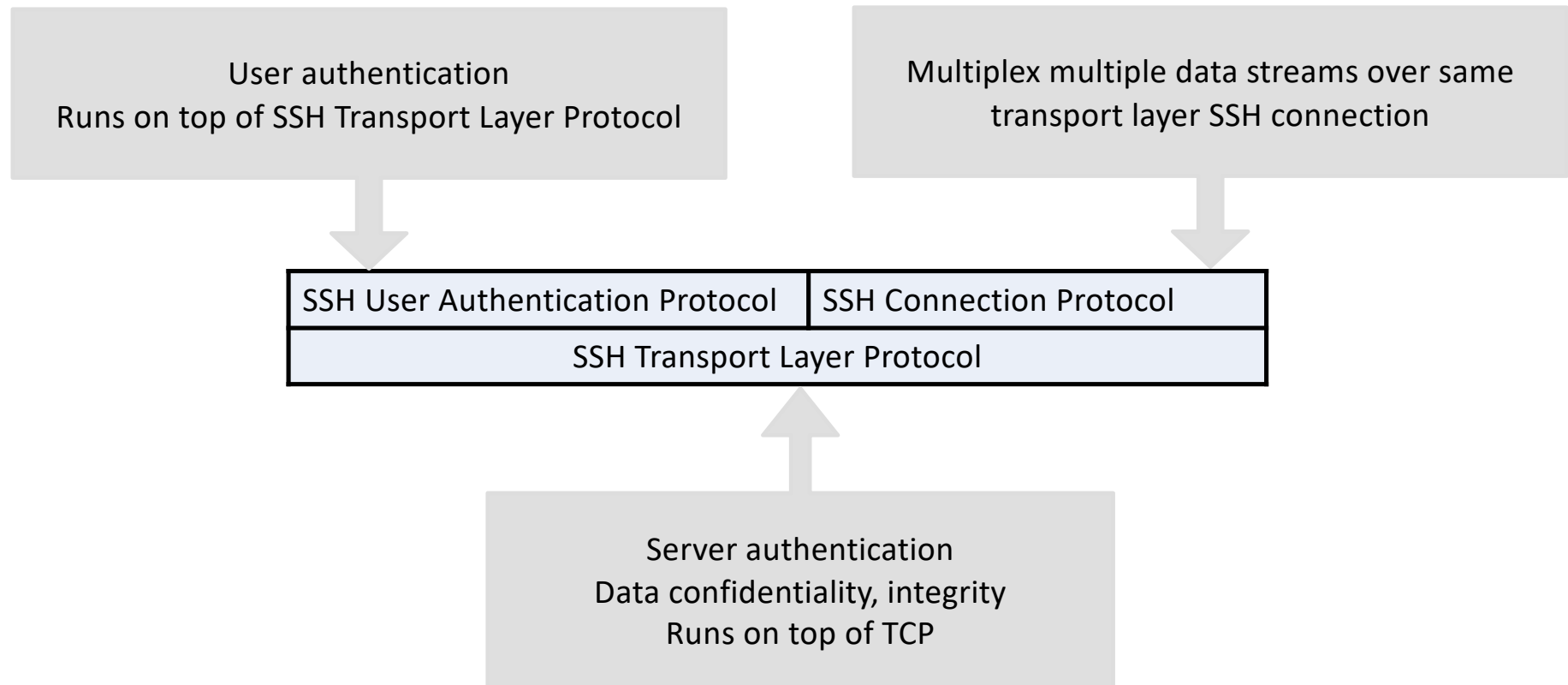
- ▶ Primary use case
- ▶ Security services offered
- ▶ TCP payload protection

SSH Main use Case and Operation

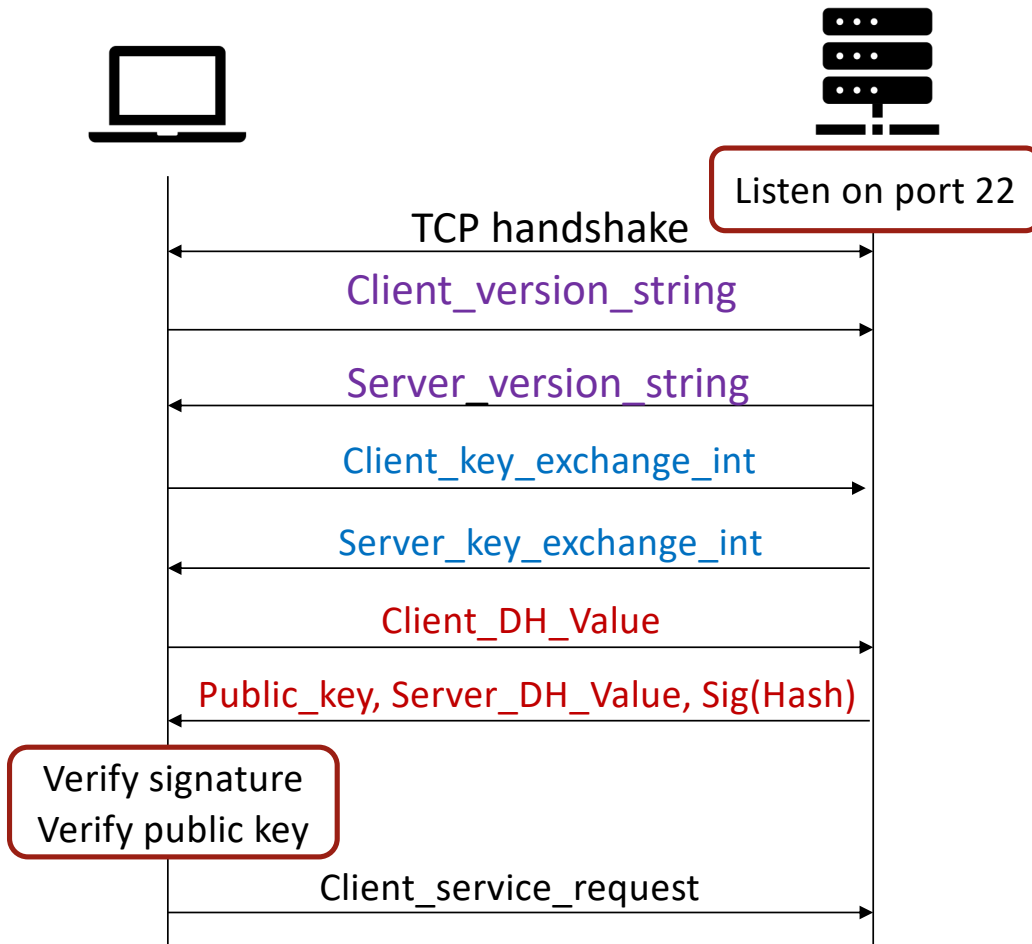
- **Originally designed to secure network services over an insecure network**
 - ▶ Specifically: remote login onto a machine for administrative purposes or in order to run applications on a remote host
- **Developed at the same time as the first SSL version**
 - ▶ Some overlap between SSL and TLS w.r.t use cases supported
 - ▶ But original main use cases quite different
- **Operates on the application layer**
 - ▶ Does not make use of TLS



SSH Protocol Family



SSH Transport Layer Protocol (1)



- **Version string**

- ▶ SSH version, SSH software version, optional comments

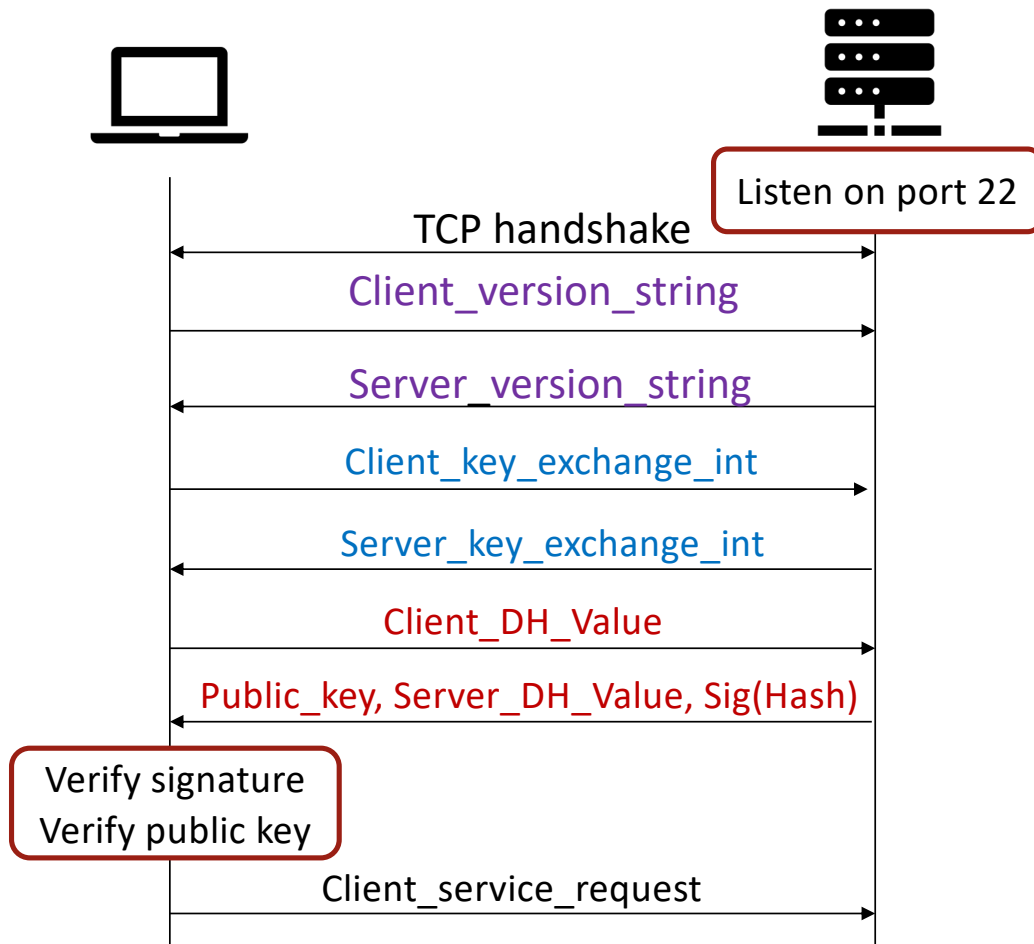
- **Key exchange init**

- ▶ RAND, list of supported key exchange methods, symmetric encryption algorithms, supported MACs, supported compression algos

- **Algorithm selection**

- ▶ Client list of algorithms ordered according to preference
 - Most preferred one first
- ▶ First algo in client list that is also on server's list is chosen

SSH Transport Layer Protocol (2)



- **DH-Value**

- ▶ Public DH value selected by client resp. server

- **Public key**

- ▶ Bare public key of server
- ▶ Alternative: public key certificate

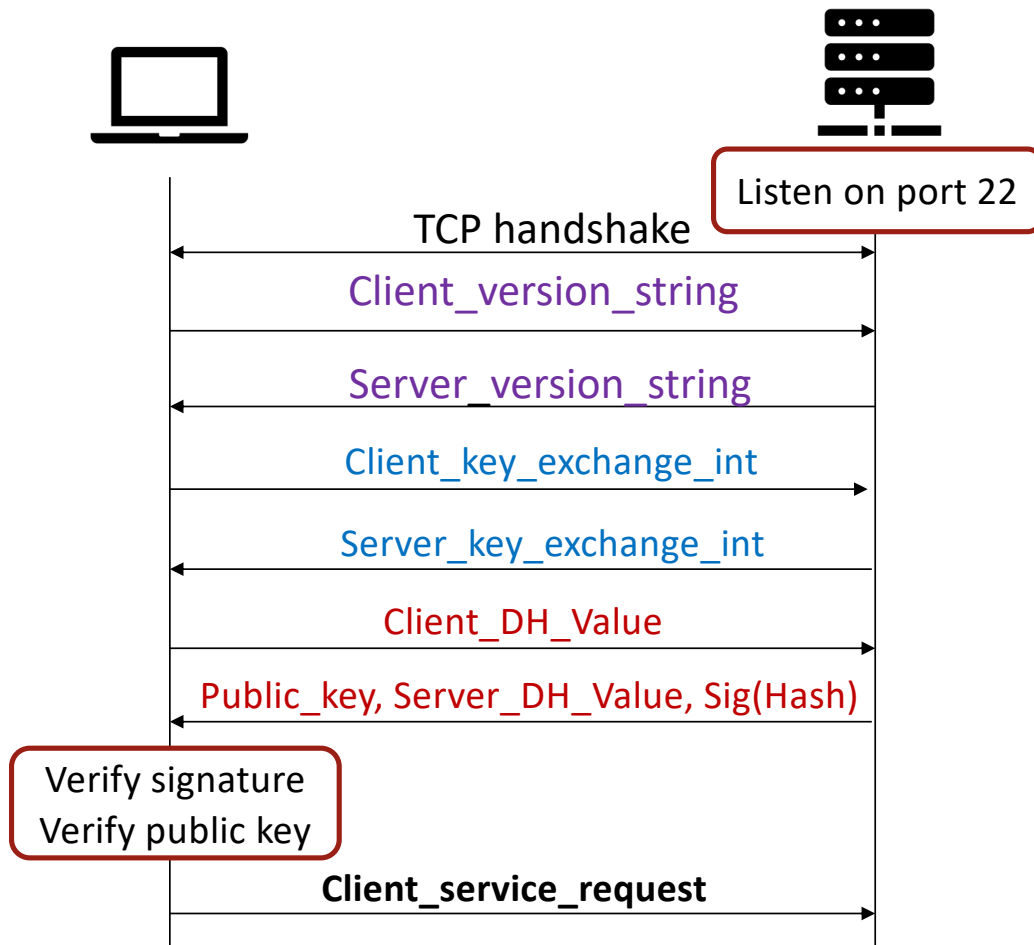
- **Hash**

- ▶ Computed on all information exchanged so far:
Client_version_string || ... || Server_DH_Value

- **Sig**

- ▶ Signature computed by server with private key corresponding to public key

SSH Transport Layer Protocol (3)



- **Session key derived from DH key**

- ▶ MAC keys, one per direction
- ▶ Encryption keys, one per direction
- ▶ IVs, one per direction if required for mode of encryption selected

- **Client_service_request**

- ▶ First message protected by the selected algorithms
- ▶ services
 - ssh-userauth
 - ssh-connection

Verifying Server's Public Key



Pre-stored

Client has local database that associates each host name with the corresponding public key of the host



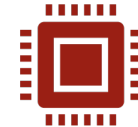
Certificate

The host name – key association is certified by a trusted CA and the server provides an appropriate certificate to the client instead of the public key alone



Fingerprint

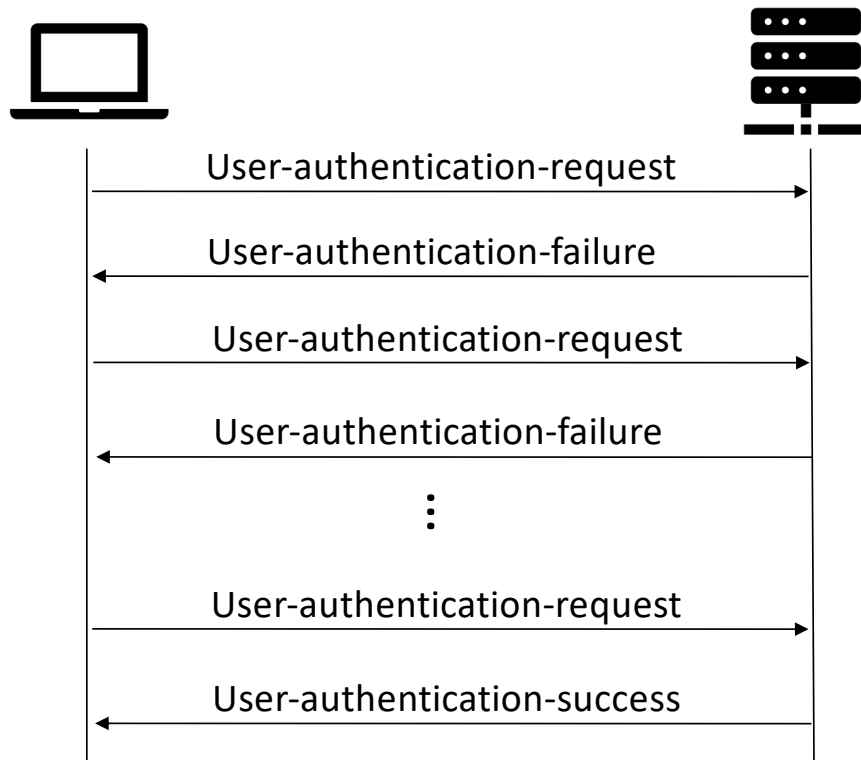
A fingerprint of the key is shown to the user and can be checked by the user over an external channel



Best effort

Accept host key without checking when first connecting to the server. Save the host key in a local database and use the pre-stored method from there on for this server

SSH User Authentication Protocol



- **User-auth-request**

- ▶ Username
- ▶ Service name
- ▶ Method name
- ▶ Method specific fields

- **User-auth-failure**

- ▶ List of auth. methods that can be used next
- ▶ partial success flag
 - True: request successful but additional auth. required
 - False: previous request not successful

- **User-auth-success**

- ▶ Server starts requested service

User Authentication Request per Method



Public key method

public key or public key certificate of user

Signature algorithm

signature on

- session ID (hash of messages exchanged during ssh transport establishment)
- all other data in the request

Password method

send user password over the ssh transport



Host based key method

public key or public key certificate of host

Signature algorithm

signature on

- session ID (hash of messages exchanged during ssh transport establishment)
- all other data in the request

SSH – Connection Protocol

- **Provides**
 - Interactive sessions, i.e., a remote execution of a program
 - E.g., a shell, an application, a system command
 - May involve forwarding of X11 connections
 - Forwarding TCP/IP connections aka port forwarding aka TCP/IP tunneling
 - Comes in different flavors explained on the following slides
- **All these applications can be multiplexed into a single encrypted and integrity protected tunnel**

Summary (1)

- **Many applications can be protected with the help of TLS**
 - ▶ HTTP, FTP, ...
- **Some applications nevertheless have special needs**
 - ▶ Email: asynchronous natures makes use of TLS end-to-end impossible
 - ▶ DNS: caching makes use of TLS end-to-end undesirable
- **TLS may help to protect data transfer in a hop-by-hop fashion**
 - ▶ E.g., Email from mail server to user's mail client
- **End-to-end protection of Email can be provided by PGP or S/MIME**
 - ▶ Both support encryption with symmetric key for end-to-end confidentiality
 - ▶ Digital signatures on hash of message for non-repudiation
 - ▶ Encryption of symmetric key with public key of receiver

Summary (2)

- **PGP or S/MIME mainly differ in how public keys are distributed**
 - ▶ S/MIME uses CA and classical certificates
 - ▶ PGP originally designed to use user-signed certificates
 - Supports use of CAs as well
 - Authenticity of keys “computed” from individually assigned trust levels
- **End-to-end protection does not prevent spoofing if it is optional to use**
 - ▶ We still see a lot of email spoofing especially in the context of phishing and SPAM
 - ▶ DKIM, SPF, and DMARC aim at making it harder
- **DNS system is mainly used to map domain names to IP addresses**
 - ▶ Also, to map domain names to name servers and mail servers responsible for the domain

Summary (3)

- **DNS is originally unprotected**

- ▶ DNS queries and responses are neither encrypted nor integrity-protected
- ▶ Consequently, DNS responses can be forged (see cache poisoning for example)
 - **DNSSEC** ensures that only authentic RRs are cached
 - Public keys required to verify authenticity of RRs are distributed via DNS
- ▶ DNS queries can be eavesdropped on
 - Addressed by **DoT** which protects connection between client and local DNS server with TLS

- **Remote login**

- ▶ SSH offers authentication, confidentiality, and integrity for remote login
- ▶ SSH was developed in parallel to the first SSL versions
- ▶ Telnet over TLS offers the similar security services as SSH but SSH offers more additional features

References

Book Chapters

- ▶ Stallings Chapter 16

RFCs related to Email Security

- ▶ PGP: RFC 4880 OpenPGP Message Format
- ▶ S/MIME: RFC 5751
- ▶ DKIM: RFC 6376
- ▶ SPF: RFC 7208
- ▶ DMARC: RFC 7489

Root server

- ▶ <https://root-servers.org>

Book

- ▶ Allan Liska and Geoffrey Stowe: “DNS Security: Defending the Domain Name System”, Elsevier 2016

RFCs related to DNS

- ▶ RFC 1034: Domain Names – Concepts and Facilities
- ▶ RFC 1035: Domain Names – Implementation and Specification
- ▶ RFC 4033: DNS Security Introduction and Requirements
- ▶ RFC 4034: Resource Records for the DNS Security Extensions
- ▶ RFC 4035: Protocol Modifications for the DNS Security Extensions

Key Signing ceremony for the root KSK

- ▶ <https://www.iana.org/dnssec/ceremonies>

References

- **W. Stallings, Cryptography and Network Security: Principles and Practice, 8th edition, Pearson 2022**
 - ▶ Chapter 19: Electronic Mail Security, DNS, DNSSec
 - ▶ Chapter 17.4: SSH

SSH Specification Details

- ▶ The secure shell transport layer protocol RFC 4253
- ▶ Latest Algorithm recommendations for SSH
 - RFC 9142



IT-Security

Chapter 8: Denial-of-Services Attacks

Prof. Dr.-Ing. Ulrike Meyer



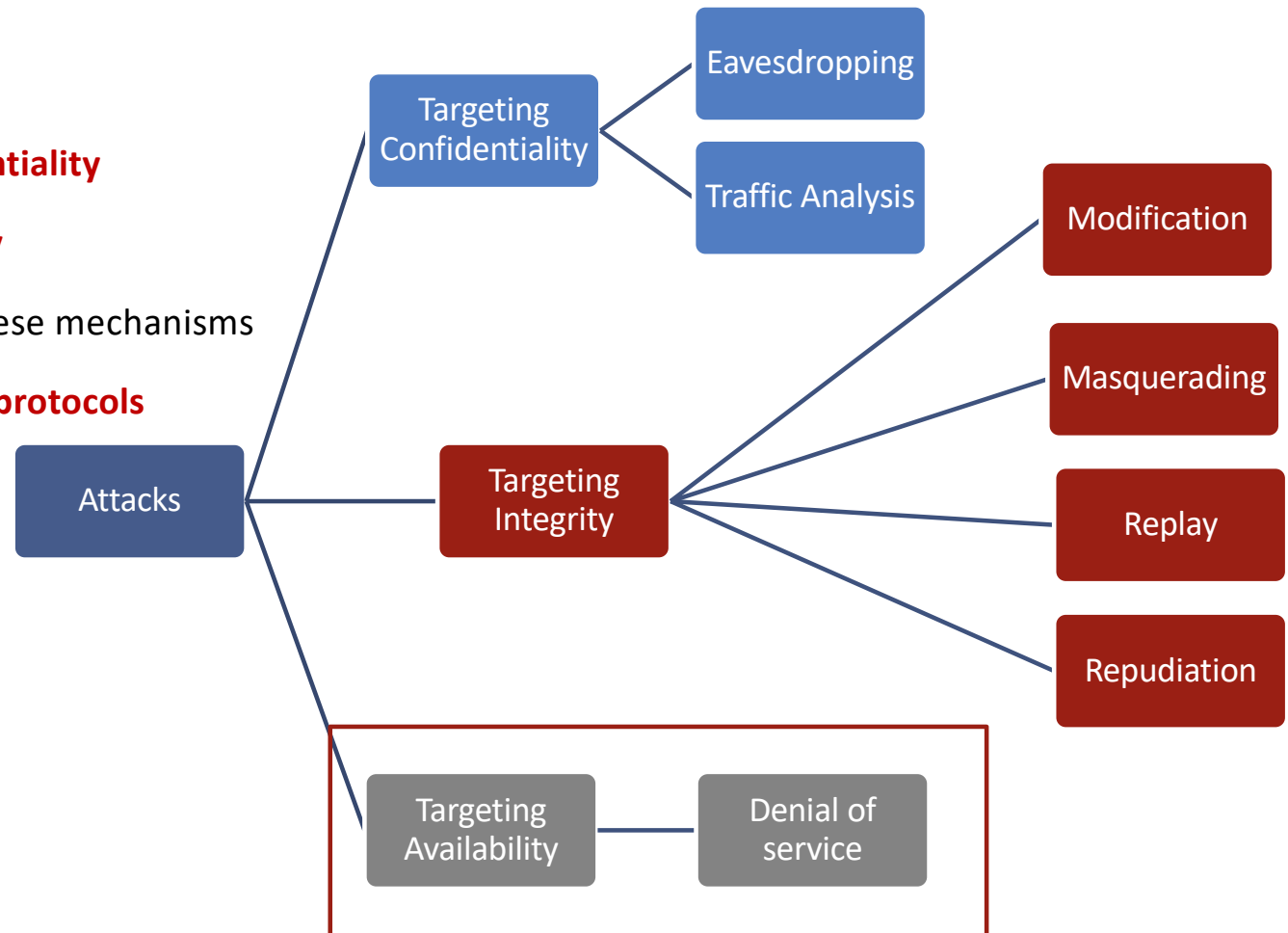
Overall Lecture Context

- So far, we focused on

- ▶ Mechanisms to **protect confidentiality**
- ▶ Mechanisms to **protect Integrity**
- ▶ **Key distribution** methods for these mechanisms
- ▶ Using them to **protect network protocols**
 - on IP, TCP, and application layer

- Not covered yet

- ▶ Availability



Overview

- **Definition of Denial-of-Service Attacks**

- **Types of attacks and simple examples targeting**

- ▶ Network bandwidth
- ▶ System resources
- ▶ Application resources

- **DoS Defenses**

- ▶ Incident response cycle
- ▶ Examples for preventive measures

- **More Advanced Techniques**

- ▶ Source address spoofing
- ▶ DDoS with compromised machines
- ▶ Reflection Attacks
 - Basic principle
 - Amplification attacks as subtype of reflection attacks

Definition and Classification

Definition

A **denial of service** (DoS) is an action that prevents or impairs the authorized use of networks, systems, or applications by **exhausting resources** such as central processing units, bandwidth, and disk space.

Classification according to type of resources targeted

Network Bandwidth

- Targets **capacity** of network link connecting victim server to the Internet

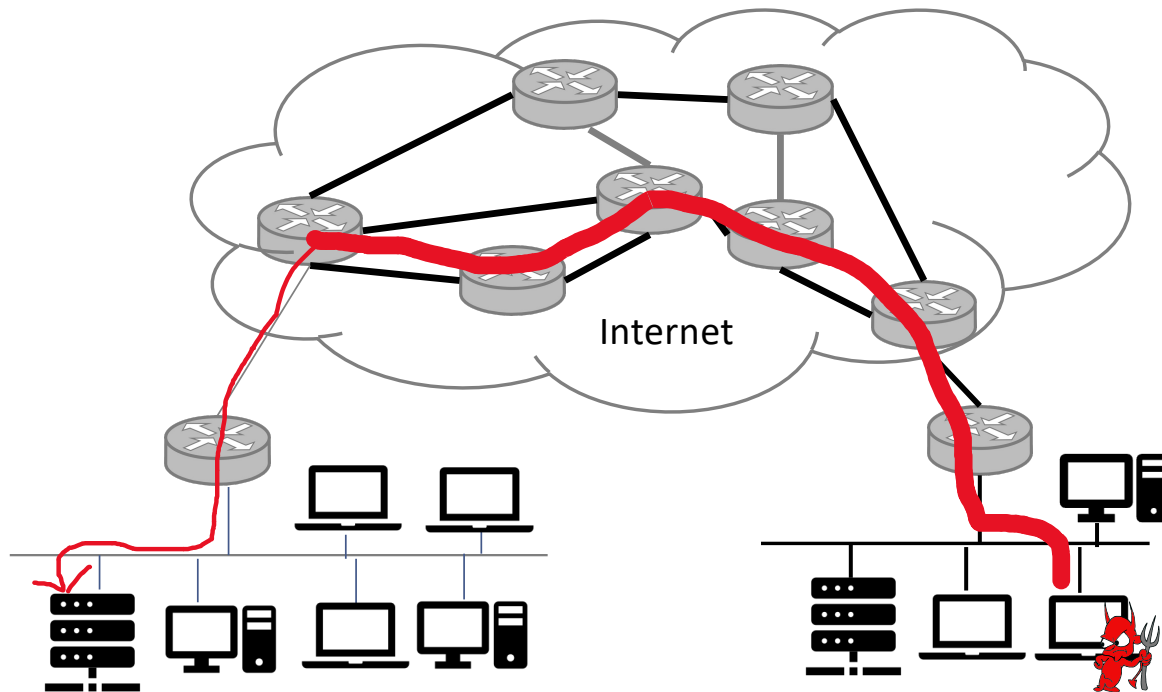
System Resources

- Targets **overloading** or **crashing** network handling software of the OS installed on the victim machine

Application Resources

- Targets a **specific application** such as a Web server or a DNS Server and overloads it with many resource consuming valid-looking requests

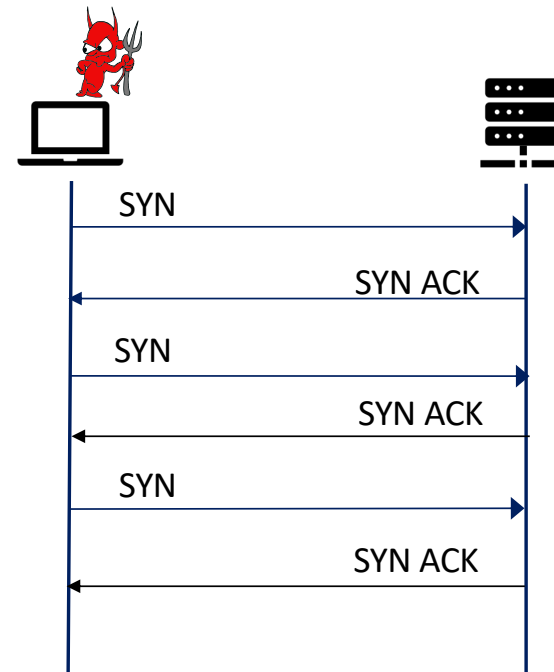
Example: Classic Flooding Attack targeting Network Capacity



- Attacker floods target network with requests
 - ▶ E.g., ICMP echo requests aka "ping"
- Router connecting target network to the ISP starts dropping IP packets
- Consequently, legitimate packets are dropped as well
- Works well if attacker's connection has higher bandwidth than target's

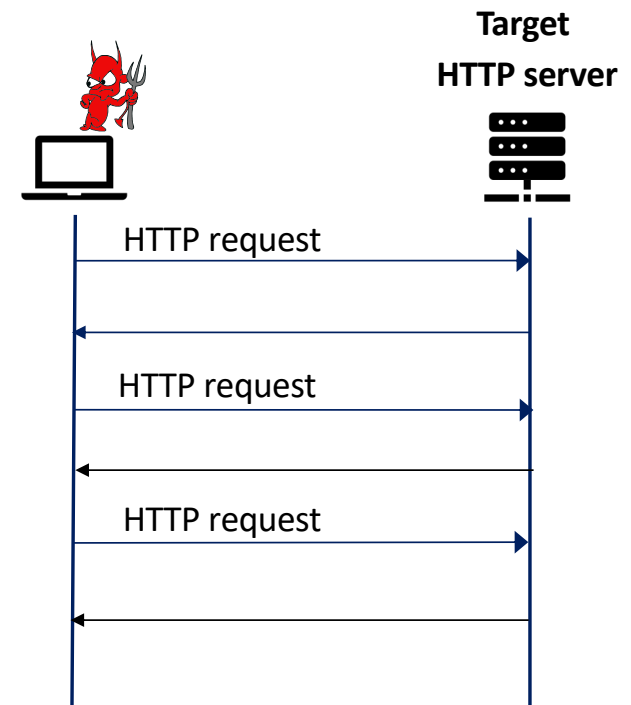
Example: Classic SYN Flooding Attack targeting System Resources

- **Targets system resources of a target server**
 - ▶ Namely, table of open TCP connections
- **Flood server with TCP SYN messages**
 - ▶ Fills up table of open TCP connections
- **Future request from legitimate users fail**
 - ▶ Server unavailable for legitimate requests



Example: HTTP Flood targeting Application Resources

- Bombard web server with HTTP requests
- Request crafted to consume considerable resources
 - ▶ E.g., request to download a large file from the target
 - Causes the web server to read the file from hard disk
 - Store it in memory
 - Convert it into a packet stream
 - Transmit the packets
 - Thus, consumes memory, processing, and transmission resources
 - ▶ Another example: recursive HTTP flood
 - Attacker starts from a given HTTP link to the server, then follows all links on the provided website recursively
 - Also called **spidering**



Overview

- **Definition of Denial-of-Service Attacks**

- **Types of attacks and simple examples targeting**

- ▶ Network bandwidth
- ▶ System resources
- ▶ Application resources

- **DoS Defenses**

- ▶ Incident response cycle
- ▶ Examples for preventive measures

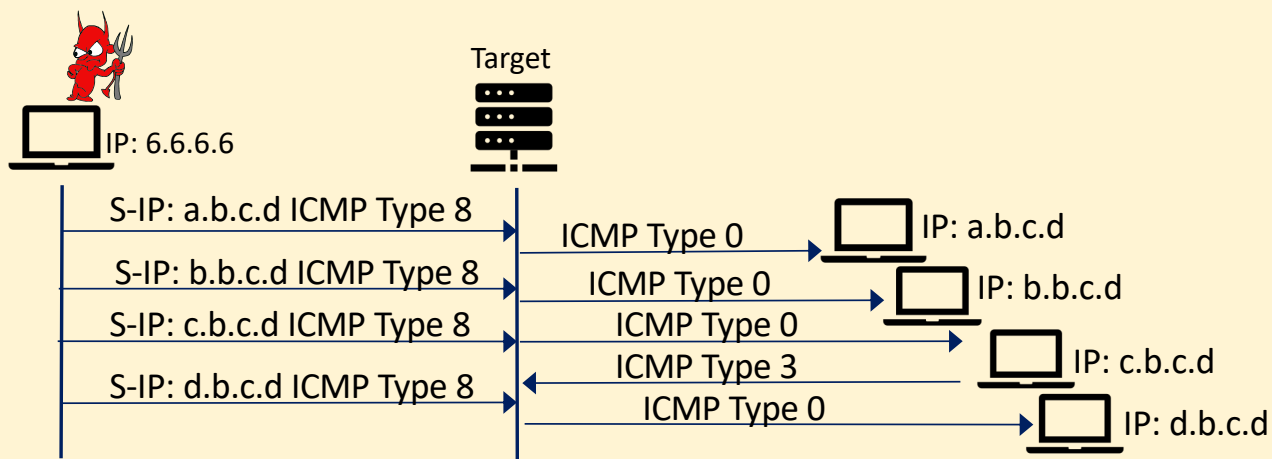
- **More Advanced Techniques**

- ▶ Source address spoofing
- ▶ DDoS with compromised machines
- ▶ Reflection Attacks
 - Basic principle
 - Amplification attacks as subtype of reflection attacks

Source Address Spoofing Directly

- Attack from single IP can easily be blocked
- Attackers often use spoofed IP addresses
 - ▶ Attacker will not be hit by the responses!

ICMP-food with IP address spoofing

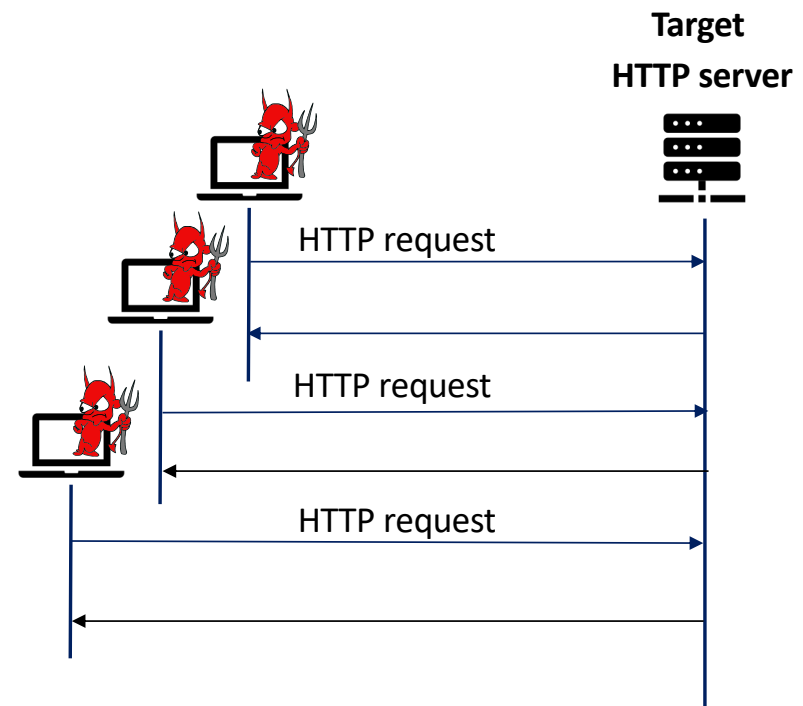


• Thwarting spoofing

- ▶ Block packets with topologically invalid IP
- ▶ Needs to be applied close to the on on the subnet the attacker acts from
- ▶ Unfortunately, there are still ISPs that do not implement such filtering
 - Too costly?

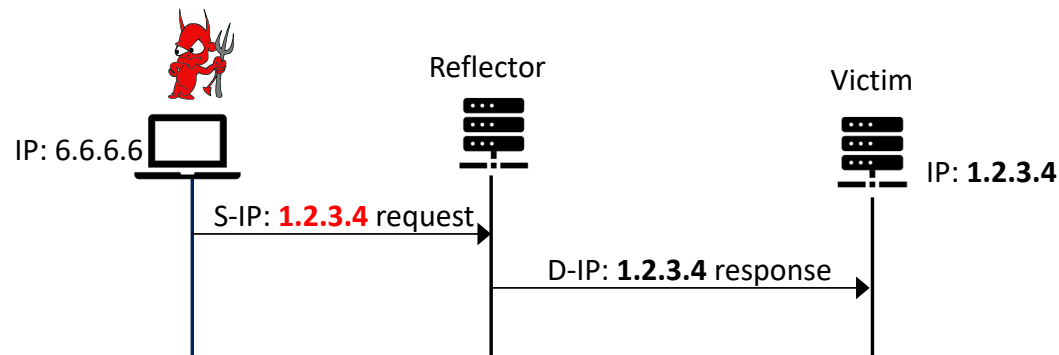
Distributed Denial of Service Attacks

- Also known as **DDoS** attacks
- Attacker makes many compromised devices send requests to the target
 - ▶ Compromised machines often infected with a bot malware
 - ▶ Remotely controllable by the attacker
 - ▶ May attack multiple targets over time
 - ▶ Owners of compromised devices unaware of the fact that their devices participate in attacks



Principle of Reflection Attacks

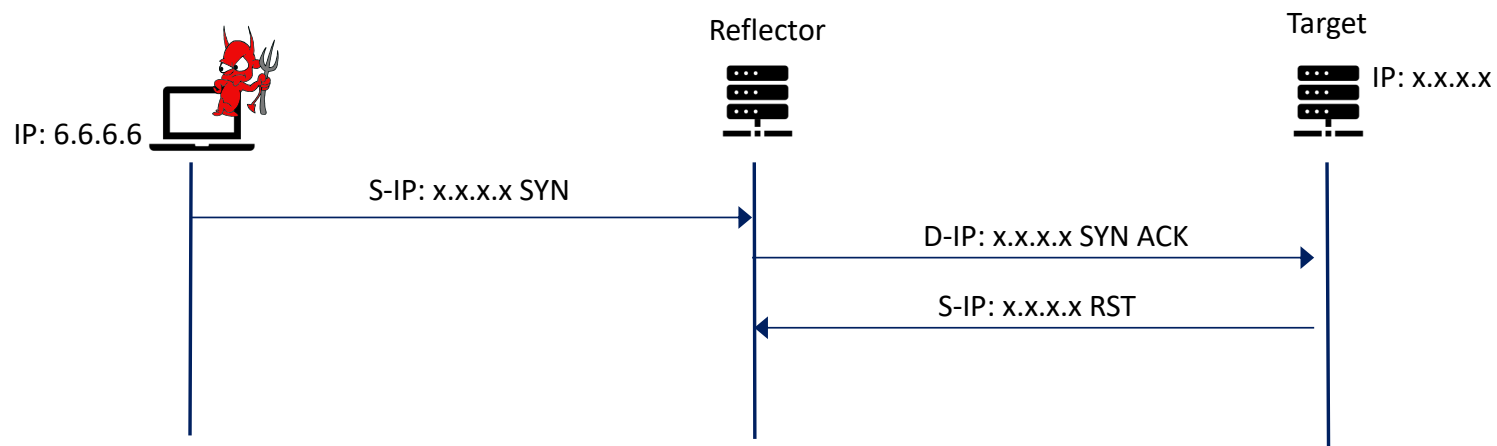
- Spoof source address to victim's address in requests sent to multiple reflectors
- Overwhelm victim with replies sent out by the reflectors to these faked requests
- Reflectors attack the victim, not the attacker himself
 - ▶ Each reflector may see only one request -> not suspicious
 - ▶ Victim is hit by lots of unsolicited responses



Reflection Attack

- **Simple example: SYN/ACK flooding attack using reflection**

- ▶ Attacker sends SYN to reflectors using the target's IP address as source address
- ▶ Reflectors respond to target with SYN ACK
- ▶ Target is flooded with unsolicited SYN ACKs sent by many reflectors



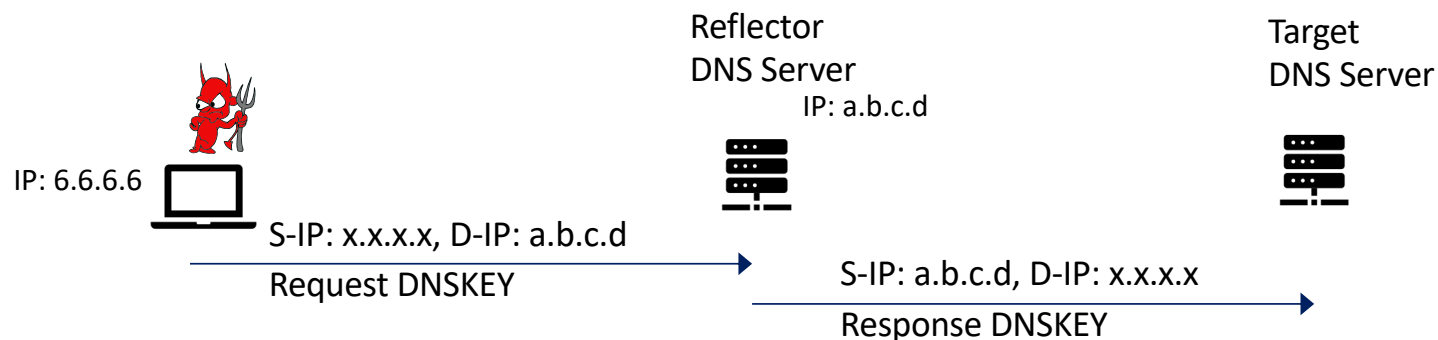
Amplification Attacks

- **Variant of reflection attacks**

- ▶ Each original packet sent by the attacker generates multiple or large response packets sent to the target

- **Example DNS amplification attack**

- ▶ Uses DNS servers as reflectors
- ▶ Attacker sends fake DNS requests to reflectors spoofing the target DNS servers IP
- ▶ Small DNS requests may lead to huge responses especially due to DNSSEC
 - DNSKEY RRs were particularly large



Overview

- **Definition of Denial-of-Service Attacks**

- **Types of attacks and simple examples targeting**

- ▶ Network bandwidth
- ▶ System resources
- ▶ Application resources

- **DoS Defenses**

- ▶ Incident response cycle
- ▶ Examples for preventive measures

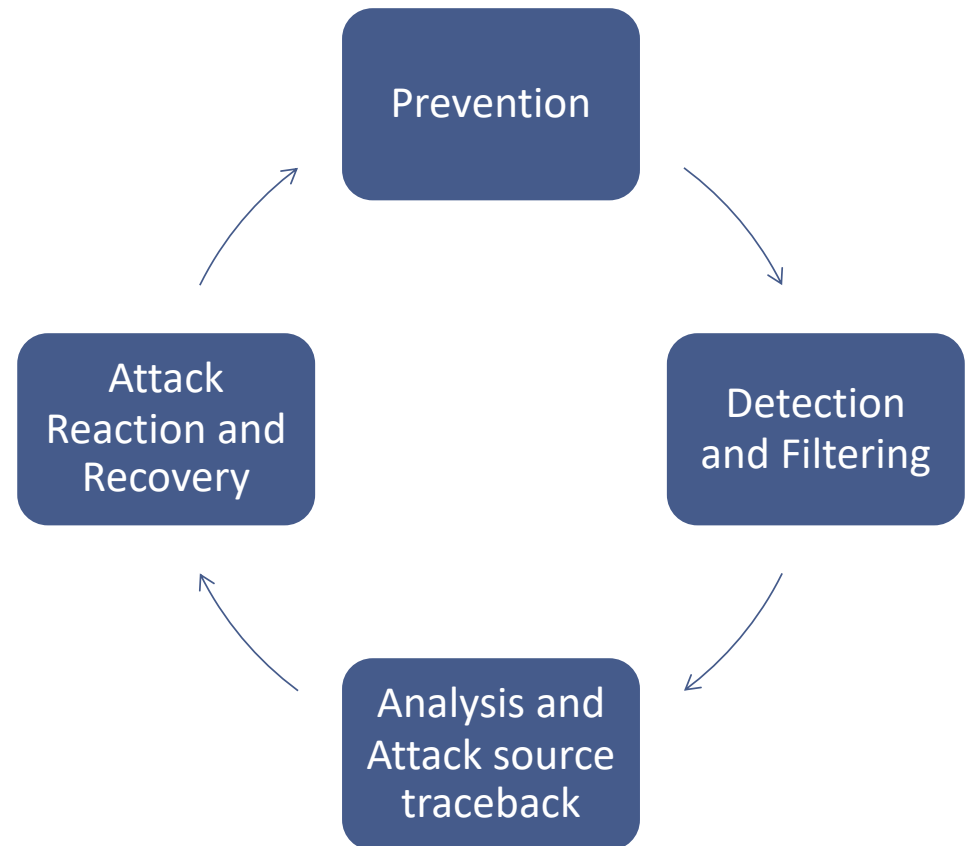
- **More Advanced Techniques**

- ▶ Source address spoofing
- ▶ DDoS with compromised machines
- ▶ Reflection Attacks
 - Basic principle
 - Amplification attacks as subtype of reflection attacks

Defense Against DoS Attacks

Incident response cycle for DoS attacks

- ▶ Take measures to prevent DoS attacks
- ▶ Take measures to detect and filter DoS attacks
 - **Intrusion detection systems**
- ▶ Log traffic to analyze after or during attack
 - Can enable attack attribution
 - Can be used to derive new preventive measures
 - Can be used to generate new detection rules
- ▶ Take measures to react to and recover from attack



Examples for Preventive Measures

Protocol Design

- Modify protocols to minimize DoS potential
- E.g., use cookies stored on client side instead of state kept on server side

Disable IP address spoofing

- Does not prevent attack against own infrastructure but helps others
- Filter IP packets with source addresses that do not belong to subnet they egress from

Throttle specific Packets

- Throttle IP packets known to be used as part of flooding attacks
- E.g., ICMP messages of type 8 (echo requests)

Lift resource limitations

- Increase the size of TCP connection tables
- Modify time-out behavior of server

Summary

- **Denial of Service Attacks can target**

- ▶ Network resources like the network bandwidth
- ▶ System resources of the operating system of hosts
- ▶ Application specific resources

- **Attackers try to hide their location by**

- ▶ using spoofed source IP addresses in attack packets
- ▶ using compromised machines of unsuspecting users
- ▶ Also shield the attacker from response traffic

- **Reflection attacks allow an attacker to indirectly attack a target**

- ▶ Attacker sends requests to reflectors with target's IP address as source
- ▶ Reflectors then flood target with reply messages

Summary

- **An amplification attack is a special form of a reflection attack**
 - ▶ Small requests sent out by attacker on behalf of target
 - ▶ Each lead to multiple response or a single large response sent by the reflectors
 - ▶ Attack is thus amplified
- **Defenses against DoS attacks try to**
 - ▶ Prevent DoS attacks in the first place
 - ▶ At least detect DoS attack if prevention is not possible
 - ▶ Filter and block attack-related traffic
 - ▶ Log attack traffic to derive future preventive and detection measures

References

- **W. Stallings, Cryptography and Network Security: Principles and Practice, 8th edition, Pearson 2022**
 - ▶ Chapter 21: Network Endpoint Security
 - 21.4 Denial of Service Attacks



IT-Security

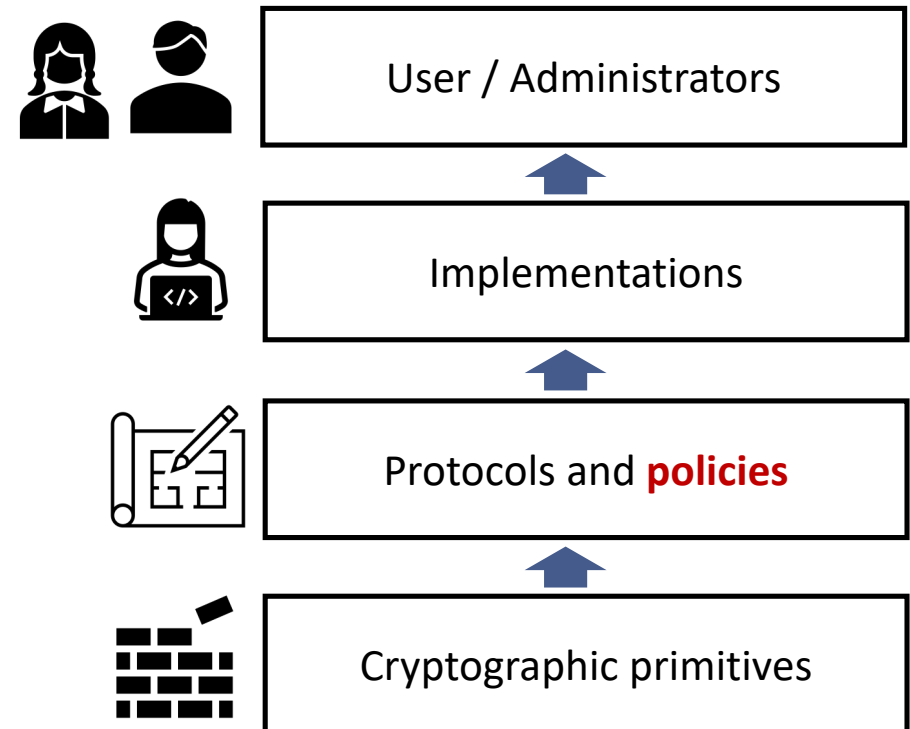
Chapter 9: Access Control, Firewalls, Intrusion Detection

Prof. Dr.-Ing. Ulrike Meyer



Overall Lecture Context

- So far, we mainly looked at **cryptographic protection** of data in transit or storage
 - ▶ IPSec, TLS, SSH, PGP, S/MIME, DNSSec
- Now we look at
 - ▶ **Access Control**: Blocking unauthorized access
 - Specifying a **policy** of who should be allowed to access what and how
 - ▶ **Firewalls**: Blocking unwanted network traffic
 - Specifying a **policy** of which traffic to allow and which to deny
 - ▶ How to at least **detect intrusions in general** if other security mechanisms fail



Overview

• Access Control

- ▶ Access control matrices and lists
- ▶ Discretionary access control
- ▶ Access control on UNIX-based systems
- ▶ Role-based access control
- ▶ Attribute-based access control

• Firewalls

- ▶ Firewalls policy
- ▶ Firewall types
- ▶ Placement of firewalls

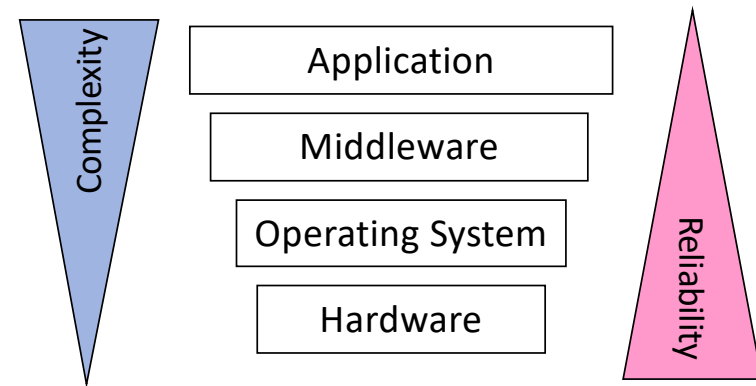
• Intrusion Detection Systems

- ▶ Components of an IDS
- ▶ Performance and Base-rate fallacy
- ▶ Anomaly vs. Misuse-based detection
- ▶ Host-based vs. network-based
- ▶ Example: SNORT

Access Control

Access Control IETF RFC 4949

Process by which use of system resources is regulated according to a security policy and is permitted only by authorized entities (users, programs, processes, or other systems) according to that policy



Authentication determines who a subject IS

Access Control determines what a subject is AUTHORIZED to DO

Access Control

Access Control IETF RFC 4949

Process by which use of system resources is regulated according to a security policy and is permitted only by authorized entities (users, programs, processes, or other systems) according to that policy

Policy expressed by **Access Control Matrix**

	Object 1	Object 2	Object 3
Subject 1	rights	rights	rights
Subject 2	rights	rights	rights

Examples for rights: read, write, execute, append,...

Authentication determines who a subject IS

Access Control determines what a subject is AUTHORIZED to DO

Access Control Lists

- Many cells in an access control matrix are empty
 - ▶ E.g., private files of a subject
- Access control lists abbreviate matrices by
 - ▶ Storing rows of matrices alongside objects

Access Control Matrix

	File 1	File 2	File 3
Subject 1	all		all
Subject 2		all	read
Subject 3			
Subject 4			
Subject 5			



File 1	
Subject 1	all
Others	

ACL of File 1

Basic Types of Access Controls

Discretionary Access Control

- ▶ Each object has an owner
- ▶ Owner decides who may access an object how
 - May include deciding how gets a special grant access right

Mandatory Access Control

- ▶ A system-wide security policy decrees access to objects
- ▶ Compares security labels of objects to security clearances of subjects

Often occur in combination in modern implementations

Discretionary Access Control Implementations Differ in

- **Which subjects can modify an objects ACL**
 - ▶ Creator of object
 - ▶ Specific right that allows changes (revocation difficult)
- **Privileged user and how ACLs apply to that user**
- **Support of groups or wildcards**
 - ▶ Allow to abbreviate ACLs
- **Handling of contradictory permissions**
 - ▶ Allow if any permission allows it
 - ▶ Deny if any permission denies it
 - ▶ Apply first matching entry
- **Application of default settings**

Classical Example for Discretionary Access Control: UNIX File System

- Each object is associated with three classes of subjects
 - ▶ owner group others
- Three rights available
 - ▶ r:read w:write x:execute
- ACL for an object indicates
 - ▶ If object is directory or not
 - ▶ Rights assigned to each subject class
- Rights are initially set to default value
- Rights can be changed by owner with `chmod`

Example ACLs

Directory	owner	group	others
d	rwX	r--	---
-	rw-	rw-	r--

Meaning of `chmod abc`

r = 4	w = 2	x = 1
7 = rwx	6 = rw-	5 = r-x
4 = r--	3 = -wx	2 = -w-
1 = --x		

```
chmod 715 file sets rwX --x r-x
```

Meaning of Rights for Directories in UNIX-based Systems

- The Unix permissions have the following effect on **directories**
 - ▶ **r** allows listing the content of a directory
 - ▶ **w** allows adding or deleting objects to/from a directory
 - ▶ **x** allows opening / executing files, cd to subdirectory if x is also set on it
- **Examples**
 - ▶ **d rwx r-- r--**
 - Allows group members to list the content of the directory but does not allow them to access any subdirectory
 - ▶ **d rwx --x r--**
 - Allows group members to change to subdirectories, open all files in the directory, execute all executables in there, but not to list them, delete them, or add files or subdirectories or executables,...

Unix: Access Decisions and User IDs and Group IDs

- **Each process or subject is associated with a user ID and at least one group ID**
 - ▶ Can be a member of more groups as well
- **Access decisions to objects are based on user IDs and group IDs**
- **When a file is created it is owned by a particular user and marked with that user's user ID as owner**
 - ▶ It also belongs to a specific group
 - Initially the primary group of its creator or the group of its parent directory if that has the setGID bit set
- **If the userid is 0 (root) then the access control decision is 'yes'**
 - ▶ I.e. root can do whatever it likes, some things can only be done by root

Unix: User IDs, Group IDs

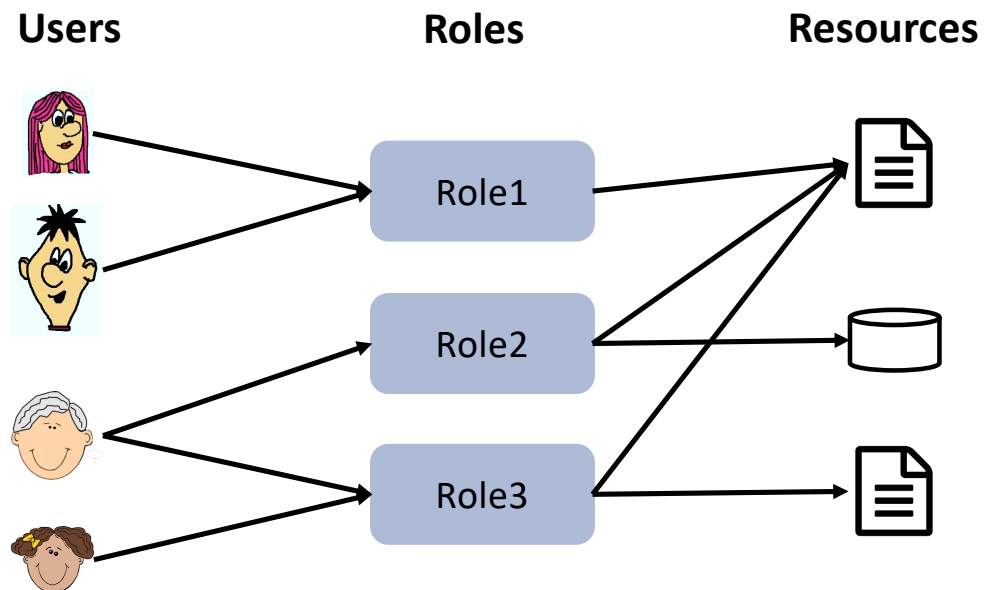
- **Each process has three user IDs and three group IDs**
 - ▶ real uid, effective uid, saved uid
 - ▶ real gid, effective gid, saved gid
- **Real user ID (ruid)**
 - ▶ identifies the owner of the process
- **Effective user ID (euid)**
 - ▶ used in most access control decisions
 - ▶ can be assigned to a process by a system call (e.g. setuid)
- **Saved user ID (suid)**
 - ▶ stores a previous user ID such that it can be restored later
- **Similar: group IDs**

Special Types of Permissions: setuid, setgid, sticky bit

- When **setuid** permission is set on an executable file, then
 - ▶ a process that runs this file is granted access based on the userID of the owner of the file
 - ▶ The effectiveUID on which the access decision depends is set to the UID of the owner of the file
 - ▶ This special permission allows allows the process running the file to access files and directories that are normally available only to the owner.
- Similar **setgid** permission
- The **setuid** and **setgid** permissions are indicated as **s** instead of **x**
 - ▶ `chmod 2000` sets the setuid bit, `chmod 4000` sets the setgid bit
- The **sticky bit** protects the files within a directory
 - ▶ If a directory has the sticky bit set, a file within it can be deleted only by file owner, directory owner, or root

Role-based Access Control Models (RBAC)

- Define roles of subjects, e.g. as job functions within an organization
- Assigns access rights to roles instead of individual users
- User are assigned roles according to their responsibilities



Access Matrix Representation of RBAC

		Role			
		R ₁	R ₂	R _n
User	U ₁	X			
	U ₂	X			
	U ₃		X		X
	U ₄				X
	U ₅				X
	U ₆				X
	⋮				
U _m	X				

		Object						
		R ₁	R ₂	F ₁	F ₂	P ₁	D ₂	...
Role	R ₁	control		Owner, r		wake	search	
	R ₂		c	w	x			
	R ₃							
	⋮							
	R _n				w	stop		

- **Assignment of users to roles according to their responsibilities**
- **Access control matrix maps Roles to Objects**
 - ▶ Allowing for roles as objects allows for hierarchy

Attribute-based Access Control

- **Defines authorizations expressed as conditions on properties of the subject, object, and the environment**
 - ▶ E.g. attribute of object: creator of the object
 - ▶ Then a single access rule can specify the ownership privilege for any creator of an object
- **Advantage of ABAC**
 - ▶ Flexibility and expressiveness
- **Main concern with ABAC**
 - ▶ Performance impact of evaluating predicates on objects and user properties for each access
- **Proposed uses include cooperating web services and cloud computing**

Entities and Attributes in the ABAC Model

• Subject

- ▶ An active entity that causes information to flow amount objects or changes system state
 - E.g. , user, application, process, device

• Subject attributes

- ▶ Associated with a subject
- ▶ Define the identity and characteristics of the subject
 - E.g., subject identifier, organization, job title,...

• Object

- ▶ Passive system-related entity
 - in the context of a specific request information
 - E.g., device, file, record, table, process, program, network, domain,...

• Object attributes

- ▶ Associated with an object
 - E.g., title, creator, date, author,...

• Environment attributes

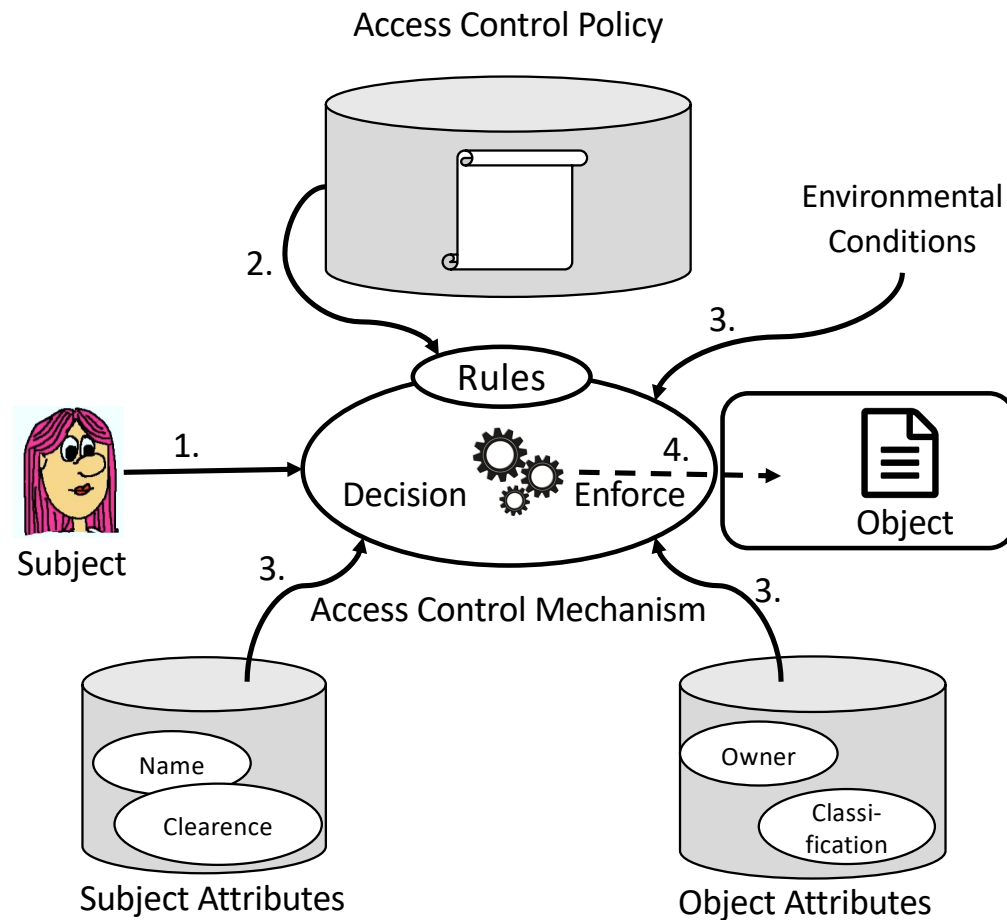
- ▶ Operational, technical, or situations environement or context in which the access occurs
 - E.g. Current date and time, current malware activities,,...

Attribute Evaluation

- **ABAC relies on**
 - ▶ Evaluation of attributes of subjects and objects
 - ▶ A formal access control rule defining allowable operations for subject/object attribute combinations in a given environment
- **ABAC systems are able to enforce DAC, RBAC, and MAC concepts**
- **ABAC enables fine-grained access control**

ABAC Logical Architecture

1. Subject request access to object
2. Access control mechanism governed by set of rules defined by access control policy
3. Based on these rules assesses attributes
4. Grants access to object if access is authorized, denies otherwise



Example: RBAC vs ABAC (1)

- **Assume store must enforce access rule to movies based on user age and movie rating (no environment here)**
- **In RBAC**
 - ▶ Users would be assigned one of the three roles adult, juvenile, child
 - ▶ Three rights: can view R-rated movies, can view PG-13-rated movies, can view G-rated movies
 - ▶ The adult role obtains all three rights, the juvenile role only the last two, child role only the last one

Movie Rating	Users Allowed Access
R	Age 17 and older
PG-13	Age 13 and older
G	Everyone

Example: RBAC vs ABAC (2)

- In ABAC there is no need for roles, instead whether a user u can access a movie m given environment e is determined by a rule

```
R1: can_access(u, m, e) ←  
    (Age(u) ≥ 17 ∧ Rating(m) ∈ {R, PG-13, G}) ∨  
    (Age(u) ≥ 13 ∧ Rating(m) ∈ {PG-13, G}) ∨  
    (Age(u) < 13 ∧ Rating(m) ∈ {G})
```

Movie Rating	Users Allowed Access
R	Age 17 and older
PG-13	Age 13 and older
G	Everyone

- No user to role assignment, no role to rights assignment necessary

Example: RBAC vs ABAC (3)

- **Advantage of ABAC becomes clearer if we add more attributes**
- **Assume that objects have an additional release date**
 - ▶ Divides movies into `new release` or `old release`
- **Users have the attribute `premium user` or `regular user`**
 - ▶ Only `premium users` are allowed to access `new release` movies
- **To capture this new situation in RBAC we would have to**
 - ▶ Double the number of roles and double the number of rights
- **In ABAC we just need two new rules in addition to R1 on last slide:**

```
R2: can_access(u,m,e) ←  
    ((MembershipType(u) = Premium) ∨  
    ((MembershipType(u) = Regular ∧ MovieType(m) = OldRelease)
```

```
R3: can_access(u,m,e) ← R1 ∧ R2
```

Overview

• Access Control

- ▶ Access control matrices and lists
- ▶ Discretionary access control
- ▶ Access control on UNIX-based systems
- ▶ Role-based access control
- ▶ Attribute-based access control

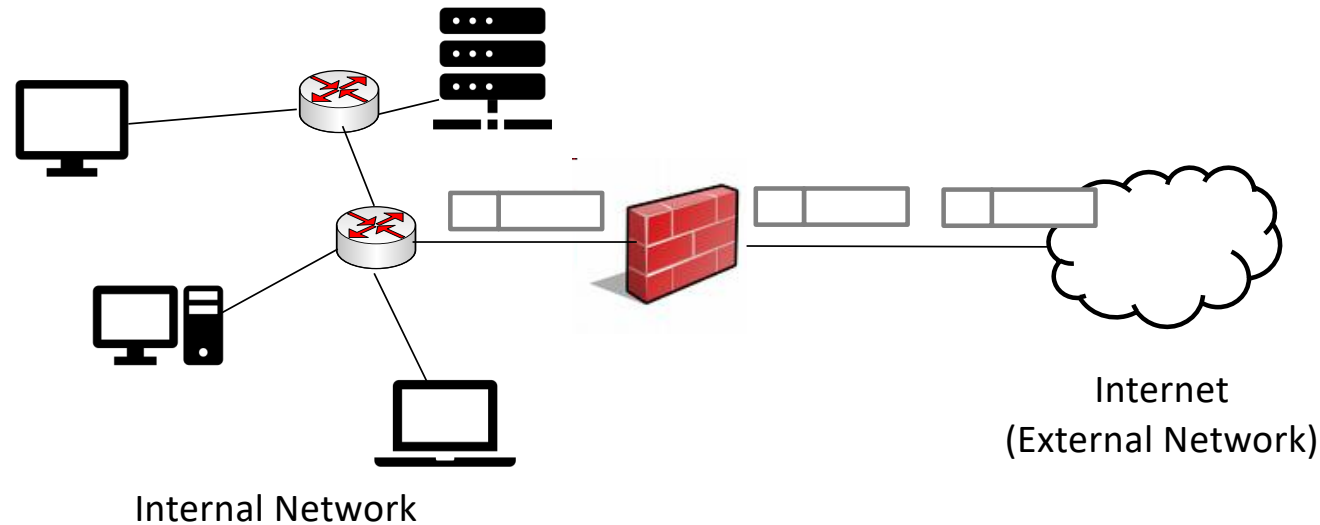
• Firewalls

- ▶ Firewalls policy
- ▶ Firewall types
- ▶ Placement of firewalls

• Intrusion Detection Systems

- ▶ Components of an IDS
- ▶ Performance and Base-rate fallacy
- ▶ Anomaly vs. Misuse-based detection
- ▶ Host-based vs. network-based
- ▶ Example: SNORT

Principle of Firewalls



- **A network firewall**
 - ▶ Controls access between an internal network and an external network
 - ▶ Allowing or denying (IP) packets according to a **security policy**
- **The internal network is to be secured, the external network is not trusted**

Firewall Policy

- **When a packet arrives at a firewall, a security policy is applied to determine the appropriate action**
 - ▶ Accept / deny
 - ▶ If a packet is denied it is either silently dropped or bounced back
 - ▶ In addition a firewall often logs information about packets arriving
- **A firewall policy can be viewed as a list of rules**
 - ▶ Each rule consists of a set of tuples and actions
 - ▶ Each tuple corresponds to a field in the packet header
 - E.g for IP packets: the protocol type, source IP, destination IP, source port, destination port

Simple Example for a Firewall Policy

No.	Protocol	Src IP	Src Port	Dest IP	Dest Port	Action
1	UDP	190.1.1.*	*	*	80	deny
2	TCP	180.*	*	180.*	90	accept
3	UDP	210.1.*	*	*	90	accept
4	TCP	210.*	*	220.*	80	accept
5	UDP	190.*	*	*	80	accept
6	*	*	*	*	*	deny

- **Simple packet filter firewall policy**

- ▶ Rules can be fully specified or contain wildcards
- ▶ Header information of passing packet is compared to the fields of a rule
- ▶ If the packet header information is a subset of the rule, the packet is said to match the rule

Rule Matching Policy: First Match Policy

Most firewalls use a **first-match policy** as rule matching policy

- ▶ The packet header information is matched sequentially with the rules **starting from the first rule**
- ▶ The action of the **first matching rule** is executed
- ▶ Any other rules further down in the policy that may also match the packet are ignored
- ▶ A **default rule** is often placed at the end of a policy **with action deny**
 - Makes the policy comprehensive

No.	Protocol	Src IP	Src Port	Dest IP	Dest Port	Action
1	UDP	190.1.1.*	*	*	80	deny
2	TCP	180.*	*	180.*	90	accept
3	UDP	210.1.*	*	*	90	accept
4	TCP	210.*	*	220.*	80	accept
5	UDP	190.*	*	*	80	accept
6	*	*	*	*	*	deny

Example

No.	Protocol	Src IP	Src Port	Dest IP	Dest Port	Action
1	UDP	190.1.1.*	*	*	80	deny
2	TCP	180.*	*	180.*	90	accept
3	UDP	210.1.*	*	*	90	accept
4	TCP	210.*	*	220.*	80	accept
5	UDP	190.*	*	*	80	accept
6	*	*	*	*	*	deny

- **Assume the following packet arrives:**
 - ▶ TCP, 210.1.1.1:3080, 220.2.33.8:80
- **What will be the rule to apply?**

Example

No.	Protocol	Src IP	Src Port	Dest IP	Dest Port	Action
1	UDP	190.1.1.*	*	*	80	deny
2	TCP	180.*	*	180.*	90	accept
3	UDP	210.1.*	*	*	90	accept
4	TCP	210.*	*	220.*	80	accept
5	UDP	190.*	*	*	80	accept
6	*	*	*	*	*	deny

- Assume the following packet:
 - ▶ TCP, 210.1.1.1:3080, 220.2.33.8:80
- First matching rule: rule 4, action: accept

Simple Mathematical Model of a Packet Filtering Firewall

- **Each tuple in a rule can be modeled as set of packets**
 - ▶ E.g ,the tuple 198.188.150.* corresponds to the set of IP addresses ranging from 198.188.150.0 to 198.188.150.255
- **The tuples of a rule collectively define a set of packets that match this rule**
 - ▶ E.g. the rule Proto = TCP, SIP = 190.150.140.38, SP = 188, DIP = 190.180.39.*, DP = 80, action = accept defines a set of 256 unique packet headers that match this rule
- **The overall set of possible packets is denoted by P**
- **Each firewall policy R can be described by three sets**
 - ▶ $A(R) \subseteq P$ describes the set of **packets** that will be **accepted**
 - ▶ $D(R) \subseteq P$ describes the set of **packets** that will be **denied**
 - ▶ $U(R) \subseteq P$ describes the set of **packets** that **do not match any rule in the policy**

Simple Mathematical Model (2)

- A firewall policy R is considered **comprehensive** if any packet from P will match at least one rule
 - ▶ I.e. $A(R) \cup D(R) = P$ or equivalently $U(R) = \emptyset$
 - ▶ typically ensured by adding a default rule of “deny” at the end of the policy
- This simple model also allows to compare two policies
 - ▶ Assume two firewall policies R, S
 - ▶ The two policies are said to be **equivalent** if their accept, deny and non-match sets are the same
- Note that being equivalent does not mean that the two policies have the same rules!!
 - ▶ Just that given any packet the two policies will lead to the same actions always

Anomalies on First-Match Policies - Shadowing

- In first-match policies more specific policy rules typically appear at the beginning of the policy and more general ones appear towards the end
- An **anomaly** is an unintended consequence of adding rules in a certain order
 - ▶ Introducing anomalies into large firewalls is very easy
- **Example: shadowing**
 - ▶ Occurs if an earlier rule i matches every packet that another lower rule j ($j > i$) matches

No.	Protocol	Src IP	S - Port	Dest IP	D - Port	Action
i	TCP	190.150.140.38	188	190.180.39.*	80	accept
j	TCP	190.150.140.38	188	190.180.39.180	80	drop

- If both rules have the same action, this is not a problem but if e.g. rule i is added after rule j the

Anomalies on First-Match Policies – Half-Shadowing

- Only a portion of an earlier rule i shadows a lower rule j ($j > i$)

▶ For example

No.	Protocol	Src IP	S - Port	Dest IP	D - Port	Action
i	TCP, SIP	190.150.140.38	188	190.180.39.*	80	accept
j	TCP, SIP	190.150.140.38	*	190.180.39.180	80	drop

- ▶ The rule j is partially shadowed by the first rule i
- ▶ By itself rule j will drop any TCP packet arriving from 190.150.140.38 and destined to 190.180.39.180 on port 80
- ▶ When rule i is added before rule j, then any packet like this with source port 188 will be accepted
- ▶ Only the system administrator will typically know whether or not this behavior was intended

Policy Optimization

- **The number of firewall rules will typically impact the firewall performance**
 - ▶ Every rule requires some processing time
 - ▶ More rules will require more processing time on average
- **Ways to enhance performance through optimizing the policy**
 - ▶ Policy reordering such that rules that match more packets are placed earlier in the policy
 - Must be done with care to avoid violating the integrity of a policy
 - I.e., after reordering, the policy should still accept and deny the same packets
 - ▶ Removing unnecessary rules by
 - Removing redundant rules
 - Combining rules if possible

Removing Unnecessary Rules

- Removing redundant rules

No.	Protocol	Src IP	S - Port	Dest IP	D - Port	Action
i	TCP	190.150.140.38	188	190.180.39.*	80	drop
j	TCP	190.150.140.38	188	190.180.39.180	80	drop

- Combining several rules ...

No.	Protocol	Src IP	S - Port	Dest IP	D - Port	Action
i	TCP	190.150.140.38	188	190.180.39.*	80	accept
j	UDP	190.150.140.38	188	190.180.39.*	80	accept

- ... to one

No.	Protocol	Src IP	S - Port	Dest IP	D - Port	Action
i	*	190.150.140.38	188	190.180.39.*	80	accept

Firewall Types

- Firewalls can be categorized into three general classes
 - ▶ Packet filters
 - ▶ Stateful firewalls
 - ▶ Application layer firewalls

Application	HTTP, FTP, SMTP,...
Transport	TCP, UDP,...
Network	IPv4 and IPv6
Data Link	802.11, 802.3
Physical	

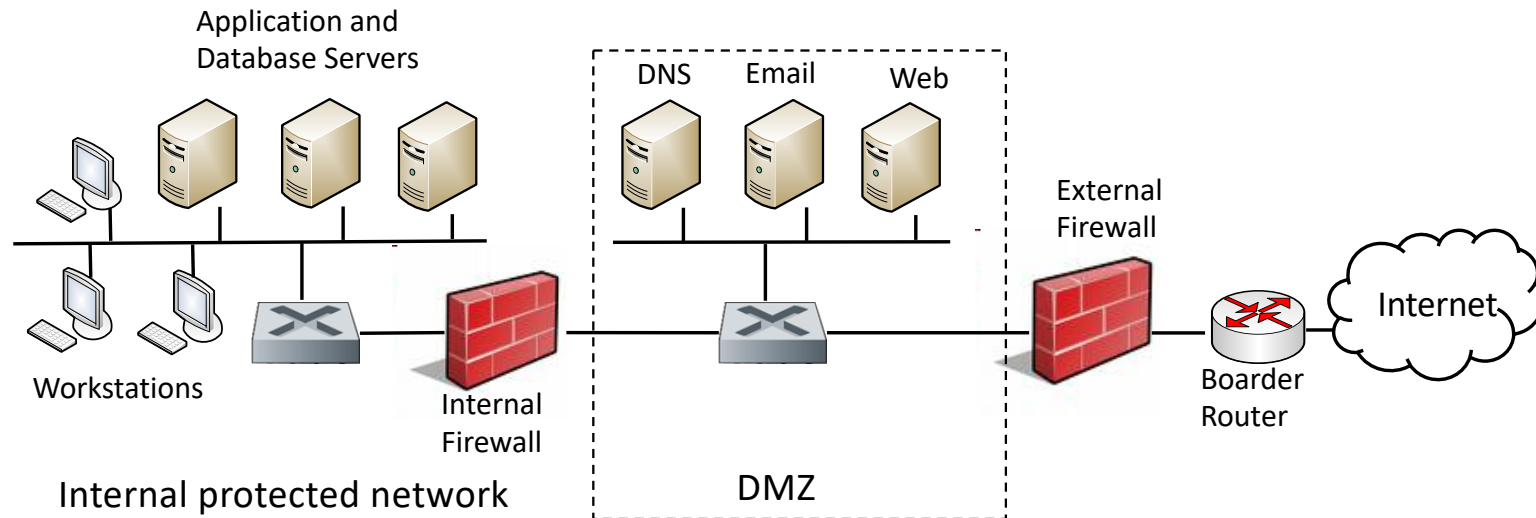
Why Stateless Filtering Is Not Enough

- **In TCP connections, ports with numbers less than 1024 are permanently assigned to servers**
 - ▶ 20,21 for FTP, 23 for telnet, 25 for SMTP, 80 for HTTP...
- **Clients use ports numbered from 1024 to 49151**
 - ▶ They must be available for clients to receive responses
- **Dynamic and/or Private Ports: 49152 through 65535**
- **What should a firewall do if it sees, say, an incoming request to some client's port 5612?**
 - ▶ It **must** allow it: this could be a server's response in a previously established connection...
 - ▶ ...OR it could be malicious traffic
 - ▶ Can't tell without keeping state for each connection

Stateful Packet Firewalls

- **Stateful firewalls perform the same operations as packet filter**
- **But they enable connection tracking**
 - ▶ E.g., if no stateful packet filter is used, allowing internal users to connect to any external webserver will require two rules
 - One for outgoing traffic to any webserver
 - One for incoming traffic to any user regardless of whether a user requested traffic from that webserver
 - ▶ A stateful firewall can support a more restrictive policy that allows incoming traffic from webserver only in response to requests by users
 - ▶ Dynamically add a rule to the policy that allows return packets when a connection is started
 - ▶ Delete this rule when the connection is closed
 - Typically based on timers as it is hard for the firewall to reliably determine whether a connection is closed

Firewall Placement: Using a Demilitarized Zone (DMZ)



- **Internal firewall adds more strict filtering capabilities compared to external firewall**
- **Internal firewall provides two-way protection to DMZ**
 - ▶ Filter attacks from DMZ towards internal network and vice versa
- **Multiple internal firewalls can be used to protect portions of internal network from each other**

Application Layer Firewalls

- **Can filter traffic at the network, transport, and application layer**
- **Typically come with proxy capabilities**
 - ▶ Application proxies are intermediaries for network connections
 - ▶ If a user on the internal network wants to connect to an application server on the external network
 - The proxy (here the firewall) would terminate the connection
 - The proxy would then create a connection to the external server
- **The firewall can thus inspect the content of the packets**
 - ▶ Like an intrusion detection system
- **Application layer firewalls and other security devices are being merged into one device**
 - ▶ E.g. intrusion prevention systems combine firewalls with intrusion detection
 - Can often filter packets as well as inspect packet contents for viruses, spam, attack signatures

Overview

• Access Control

- ▶ Access control matrices and lists
- ▶ Discretionary access control
- ▶ Access control on UNIX-based systems
- ▶ Role-based access control
- ▶ Attribute-based access control

• Firewalls

- ▶ Firewalls policy
- ▶ Firewall types
- ▶ Placement of firewalls

• Intrusion Detection Systems

- ▶ Components of an IDS
- ▶ Performance and Base-rate fallacy
- ▶ Anomaly vs. Misuse-based detection
- ▶ Host-based vs. network-based
- ▶ Example: SNORT

Intrusion Detection

- **Definitions from IETF RFC 4949 "Internet Security Glossary"**

Security Intrusion: A security event, or a combination of multiple security events that constitutes a security incident in which an intruder gains, or attempts to gain, access to a system (or system resource) without having authorization to do so

Intrusion Detection: A security service that monitors and analyzes system events for the purpose of finding, and providing real-time or near real-time warning of attempts to access system resources in an unauthorized manner

Logical Components of an Intrusion Detection System

Sensors

- ▶ Collect data from a monitored part of the system
 - Input to a sensor can, e.g., include network packets, log files, or system call traces recorded on a particular system

Analyzers

- ▶ Analyzers receive and store data collected by one or more sensors
- ▶ Tries to determine if an intrusion has occurred
- ▶ Output may include
 - Evidence supporting the detection
 - Guidance about appropriate actions to take

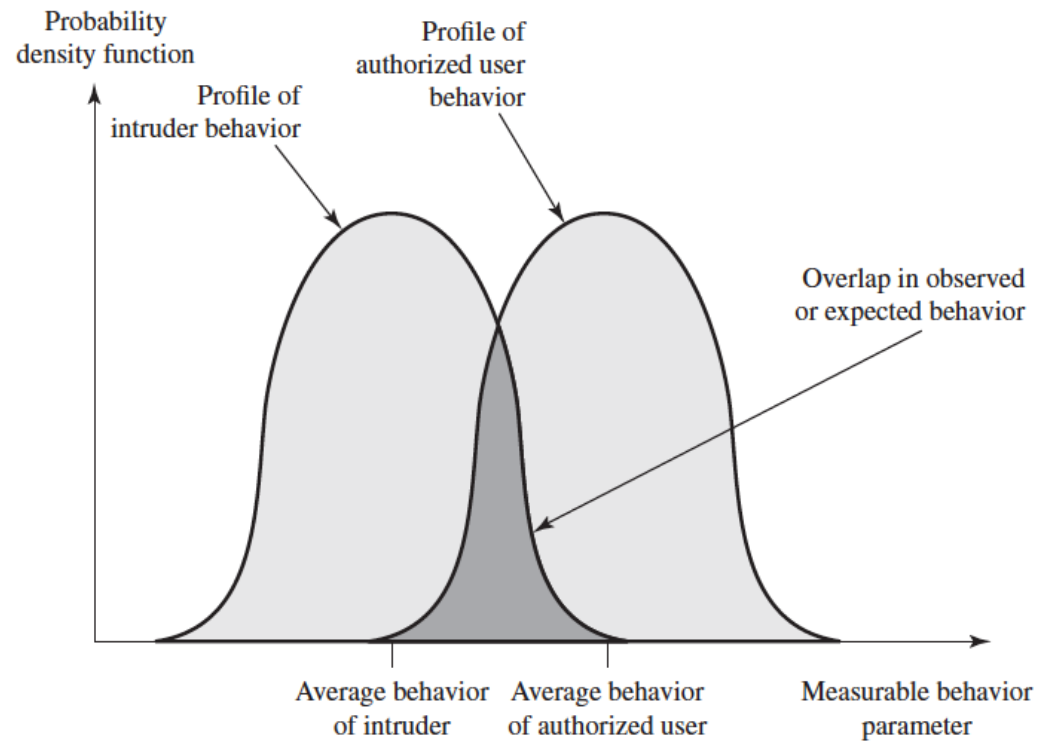
User Interface

- ▶ Enables user to view output from analyzer or control behavior of the IDS's components

Basic Principles (1)

- **Basic assumption underlying intrusion detection systems**

- ▶ Behavior of intruders differ from that of legitimate users in a quantifiable way



Basic Principles (2)

- **Typical behavior of an intruder differs from typical behavior of an authorized user, but there is an overlap in these behaviors**
- **So, any intrusion detection approach will make mistakes**
 - ▶ If it tries to catch all intruders, it will typically sometimes raise **false alarms**, i.e., cause **false positives**
 - ▶ If it tries to limit false alarms it will typically **miss some attacks**, i.e., cause **false negatives**
- **Ideally one would want an IDS to**
 - ▶ Maximize the **detection rate**, i.e., the ratio of detected to total attacks
 - = Recall = $TP / (TP + FN)$
 - ▶ Minimizing the **false alarm rate**, i.e., ratio of false positive to all negatives
 - = False Positive Rate = $FP / (FP + TN)$

TP = True Positives = Attacks rising alarm
FN = False Negatives = Attacks not rising alarm
FP = Benign behavior rising alarm
TN = Benign behavior not rising alarm

Problem: Base Rate Fallacy

- It is very difficult to meet this goal of high detection rate and low false alarm rate
- In general
 - ▶ if the actual numbers of intrusions is low compared to the number of legitimate uses of a system
 - ▶ Then if an alarm is raised the probability that indeed an attack takes place is very low unless the detection is extremely discriminative
- This phenomenon is called the **base rate fallacy**

Reminder: Conditional Probability and Bayes Theorem

- Suppose two events A and B occur with probability $\Pr(A)$ and $\Pr(B)$, respectively
- Let $\Pr(A \cap B)$ be probability that both A and B occur
- What is the **conditional probability** that A occurs given that B has occurred?

$$\Pr(A | B) = \frac{\Pr(A \cap B)}{\Pr(B)}$$

- Applying this twice we get Bayes Theorem

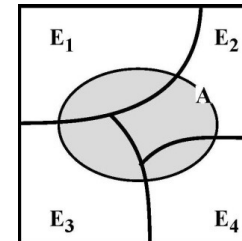
$$\Pr(B | A) = \frac{\Pr(A | B) \Pr(B)}{\Pr(A)}$$

Law of Total Probability

- Suppose mutually exclusive events E_1, \dots, E_n together **cover the entire set** of possibilities
- Then probability of any event A occurring is

$$\Pr(A) = \sum_i \Pr(A \mid E_i) \cdot \Pr(E_i)$$

Intuition: since E_1, \dots, E_n cover entire probability space, whenever A occurs, some event E_i must have occurred



Example for Base-Rate Fallacy

- **Assume**

- ▶ 1% of traffic is SYN floods, 99% of traffic is valid connections
- ▶ IDS's detection rate is 90%, i.e. IDS classifies 90% of SYN floods as attack
- ▶ IDS's false alarm rate is 1%, i.e. IDS classifies 1% of valid connections as attack

- **What is the probability that a connection flagged by IDS as a SYN flood really is a valid connection?**

$$\begin{aligned}\Pr(\text{valid} \mid \text{alarm}) &= \frac{\Pr(\text{alarm} \mid \text{valid}) \cdot \Pr(\text{valid})}{\Pr(\text{alarm})} \\ &= \frac{\Pr(\text{alarm} \mid \text{valid}) \cdot \Pr(\text{valid})}{\Pr(\text{alarm} \mid \text{valid}) \cdot \Pr(\text{valid}) + \Pr(\text{alarm} \mid \text{SYN flood}) \cdot \Pr(\text{SYN flood})} \\ &= \frac{0.01 \cdot 0.99}{0.01 \cdot 0.99 + 0.90 \cdot 0.01} \approx 52\% \text{ chance that traffic is valid} \\ &\quad \text{given an alarm is raised}\end{aligned}$$

General IDS Approaches

- **Anomaly Detection**

- ▶ Collect data corresponding to behavior of legitimate users over a period of time
- ▶ Built a model of normal behavior from it
- ▶ Try to determine whether current behavior is of a legitimate user or of intruder by comparing it to the model

- **Signature or Heuristic detection**

- ▶ Use a set of known malicious data patterns (signatures) or attack rules (heuristics) and compare them to currently observed data
- ▶ Also known as **misuse detection**
- ▶ Can only identify known attacks

In Other Words,...

- **Anomaly detection assumes that**

- ▶ What is usual, is known
- ▶ What is unusual, is bad
- ▶ Problem
 - Does not necessarily detect undesirable yet usual behavior
 - False alarm rates can be high
 - Very hard to obtain (attack free) usual behavior

- **Misuse detection assumes that**

- ▶ What is bad, is known
- ▶ What is not bad, is good
- ▶ Problem
 - Cannot detect new attacks, i.e. false negatives typically very high

Anomaly Detection

- **Training: develop a model of legitimate behavior**
 - ▶ Collect and process sensor data from normal operation of monitored system
 - ▶ May occur at distinct times or may be a continuous of monitoring and evolving the model
- **Detection: Compare observed sensor data to the trained model**
 - ▶ Classify as normal or anomalous activity
- **Detection approaches**
 - ▶ **Statistical**: analysis of the observed behavior using univariate, multivariate, or time-series models of observed metrics
 - ▶ **Knowledge-based**: approaches use an expert system that classifies observed behavior according to a set of rules that model legitimate behavior
 - ▶ **Machine-learning**: approaches automatically determine a suitable model from features extracted from the sensor data using machine-learning techniques
 - E.g. Bayesian networks, Markov models, neural networks, fuzzy logic, clustering, but also classifiers such as SVMs, Random Forests, deep neural networks,...
 - Often make use of attack data as well, i.e. train model with benign AND malicious data

Misuse Detection

- **Signature approaches**

- ▶ Match large collection of known attack patterns against data monitored on a system or in transit over the network
- ▶ Signatures need to be specific enough to minimize false alarm rate but still detect malicious data
- ▶ Typically, low cost with respect to time and resources
- ▶ But: significant effort necessary to review new attacks and generate signatures

- **Rule-based heuristic identification**

- ▶ Rules that identify suspicious behavior
- ▶ Typically, specific to the machine and operating system monitored
- ▶ Often derived from analyzing attack tools and scripts collected on the Internet
- ▶ **SNORT** is a rule-based network intrusion detection system

Misuse Detection

- **Set of **patterns** defining a behavioral signature likely to be associated with attack of a certain type**
 - ▶ Example: buffer overflow
 - A setuid program spawns a shell with certain arguments
 - A network packet has lots of NOPS in it
 - Very long argument to a string function
 - ▶ Example: SYN flooding (denial of service)
 - Large number of SYN packets without ACKs coming back
 - ...or is this simply a poor network connection?
- **Attack signatures are usually very specific and may miss variants of known attacks**
 - ▶ Why not make signatures more general?

Extracting Misuse Signatures

- Use **invariant** characteristics of known attacks

- ▶ Bodies of known viruses and worms, port numbers of applications with known buffer overflows, RET addresses of overflow exploits
- ▶ Hard to handle mutations
 - Polymorphic viruses: each copy has a different body

- **Big research challenge: fast, automatic extraction of signatures of new attacks**

- **Honeypots** are useful for signature extraction

- ▶ Try to attract malicious activity, be an early target

Host-based Intrusion Detection Systems (HIDS)

- Examines user and software activity **on a specific host**
- Aims to detect both attacks from the outside as well as internal attack
- Can use anomaly or misuse-based detection approach
- But: provide only a local view on an attack
- Are only able to detect attacks when they already hit the target system

Data Sources for HIDS

- **System call traces:** Record sequences of system calls made by processes on the system
 - ▶ Works well on Linux and Unix systems
 - ▶ Difficult on Windows systems as use of DLLs hides which process uses which system calls (use of DLL function calls proposed as alternative)
- **Audit (log file) records:** Operating systems include software that collects information on user activity
 - ▶ Problem: audit records may not include relevant information; intruder may manipulate the records
- **File integrity checksums:** Detect intruder activity on a system by periodically scanning critical files for changes
 - ▶ Use message authentication code to compute checksums, compare them to a known baseline
 - ▶ Tripwire is a well-known system using this approach
- **Registry access:** monitor access to the registry on Windows machines

Network-Based Intrusion Detection System (NIDS)

- **Monitors traffic at selected points on a network**
 - ▶ Examines traffic for a large number of hosts with a variety of devices and software
 - ▶ Examines traffic packet by packet in real-time or close to real-time
 - ▶ May examine network-, transport- and/or application-level protocol activity
 - ▶ Typically included in perimeter security infrastructure, e.g. firewall
- **Challenge**
 - ▶ Arranging the monitoring to minimize the number of agents but cover the complete network
- **Agent must have same view of traffic as destination**
 - ▶ TTL tricks, fragmentation may obscure this
 - ▶ End-to-end encryption defeats content monitoring
 - Not traffic analysis, though

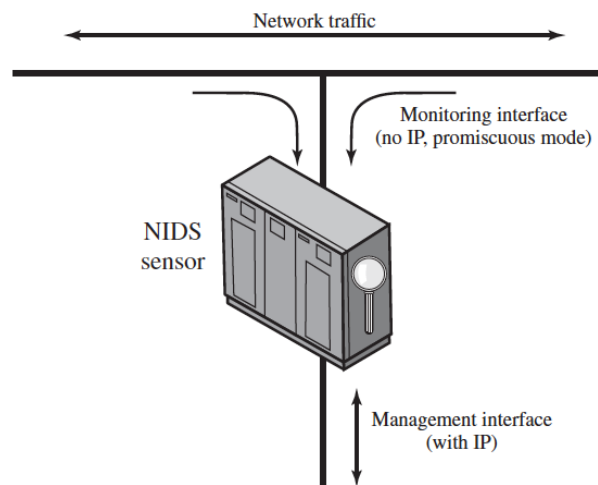
Types of NIDS Sensors

- **Inline sensor**

- ▶ Inserted into a network segment such that monitored traffic must pass the sensor
- ▶ E.g., incorporate directly in firewall or as standalone component

- **Passive sensor**

- ▶ Monitors copy of network traffic, actual traffic does not pass through it



Logging of Alerts

- **When a sensor detects potential violations it**
 - ▶ Sends an alert
 - ▶ Logs information related to the event
- **Typical information logged by an NIDS sensor includes**
 - ▶ Timestamp
 - ▶ Connection or session ID
 - ▶ Event or alert type and Rating (e.g., severity, impact, confidence,...)
 - ▶ Network, transport, and application layer protocol
 - ▶ Source and destination IPs and ports, number of bytes transmitted over the connection
 - ▶ Decoded payload data
 - ▶ State-related information (e.g., authenticated username)

Summary

- **In this chapter we looked at basic non-cryptographic security mechanisms**
- **Access Control**
 - ▶ Access Controls implement an access policy
 - ▶ An access policy determines which subjects have which rights over which objects
 - As opposed to authentication which determines who is who
 - ▶ Access Controls can be implemented on all layers of a system
 - Hardware, operating system, middleware, application
 - ▶ In discretionary access control
 - Each object has an owner, access to object is at the owner's discretion
 - ▶ In mandatory access control
 - a global policy determines access to all objects
 - ▶ In role-based access control roles are used as subjects and users are assigned one or more roles

Summary

- ▶ In attribute-based access control
 - Access is granted based on attributes of subjects, objects and the environment they act in

● Network Firewalls

- ▶ Control network traffic flow to and from one network or network segment to another
- ▶ In packet filters IP packets are accepted or blocked dependent on
 - Header Information in TCP/IP headers
- ▶ In stateful firewalls additional state is kept on previously inspected packets and acceptance / denial depends on this state
- ▶ In application layer firewalls
- ▶ Application layer content is inspected

Summary

• Intrusion Detection Systems

- ▶ Consist of sensors that collect information and analyzers that receive information from sensors
- ▶ **Network-based** intrusion detection systems
 - focus on sensors that collect information on network traffic
- ▶ **Host-based** intrusion detection systems
 - focus on sensor that collect information on individual hosts
- ▶ Intrusion detection systems try to detect attacks and provide evidence
 - Underlying assumption: attack will be visible in the collected information
- ▶ The **goal** of an IDS is to
 - Maximize the **detection rate**, i.e., the ratio of detected to total attacks and to
 - Minimizing the **false alarm rate**, i.e., ratio of false positive to all negatives

Summary

- ▶ Main approaches for IDS
 - Anomaly-based approach
 - Model normal behavior, classify anormal behavior as attack
 - Misuse-based approach
 - Model attack behavior and try to detect it
 - Model both normal and attack behavior and try to distinguish it
- ▶ Snort is an example for a
 - host-based intrusion detection system
 - that collects information on network traffic of a host
 - it uses a misuse-based approach

References

- **W. Stallings, Cryptography and Network Security: Principles and Practice, 8th edition, Pearson 2022**
 - ▶ Chapter 21: Network Endpoint Security
 - Firewalls, Intrusion Detection Systems
- **Wenliang Du, Computer Security a Hands-on Approach, 3rd edition, 2022**
 - ▶ Chapter 1: Linux Security Principle
 - Access Control



IT-Security

Chapter 10: Malware and Binary Exploitation

Prof. Dr.-Ing. Ulrike Meyer



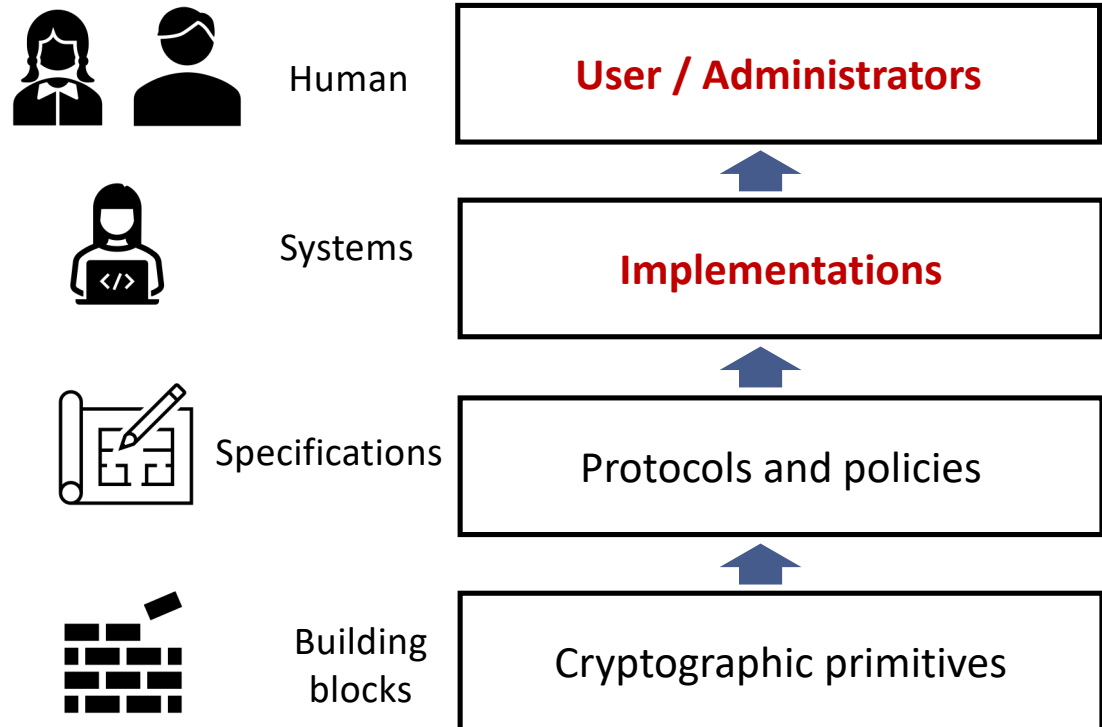
Overall Lecture Context

- So far, we mainly looked at

- ▶ Secure cryptographic building blocks
- ▶ Design of security protocols
- ▶ Users / Administrator when it comes to password selection

- Now we look at

- ▶ **Implementation** vulnerabilities
- ▶ **Social engineering**



Overview

Malware Types by Spreading

- ▶ Viruses, Worms, Trojans

Initial Infection

- ▶ Software Vulnerabilities
- ▶ Misconfigured access controls
- ▶ Vulnerable Authentication
 - Weak passwords
 - Protocol weaknesses
- ▶ Social engineering

Botnets

- ▶ C&C Infrastructures
- ▶ Taking down Botnets

Typical Payloads

- ▶ DDoS Engines
- ▶ SPAM Engines
- ▶ Phishing Engines
- ▶ Information Stealing
- ▶ Miners

Definition

Malware = Malicious Software

- ▶ According to NIST SP 800-83:

“A program that is inserted into a system, usually covertly, with the intent of compromising the confidentiality, integrity or availability of the victim’s data, applications, or operating system or otherwise annoying or disrupting the victim”

Owner of the system and victim do not necessarily coincide

Motivation to Write Malware

- Experimenting how to write malware

- Testing own programming skills



- Get famous



- Vandalism

- **Fighting authorities**



- **Direct Financial gain**



- **Corporate Espionage**



- **Combatting crime and terrorism**



- **Cyberwar**



Simple Example for Malicious Code

- **Attacker writes a small shell script on a UNIX system:**

```
cp /bin/sh /tmp/.xyz
```

```
chmod u+s,o+x /tmp/.xyz
```

```
rm ./ls
```

```
ls $*
```

- **Attacker saves this script in a file called “ls” and tricks a victim user into executing it**
- **This leads to a copy of the shell in a hidden file `.xyz`**
- **Shell executable by anyone with the `userid` set to **who-ever-executed-the-script****
 - ▶ **If `who-ever-executed-the-script` acted as root, shell will be a root shell executable by anyone**
- **To the victim user, the result will look the same as result of real `ls`**
 - ▶ **Script removes itself**

Malware Types with respect to Spreading

Trojan Horse

- ▶ Program with an
 - overt purpose known to the user
 - covert purpose unknown to the user
- ▶ Typically installed by the user itself

Virus

- ▶ Software fragment that attaches to an existing executable
- ▶ Can replicate itself from one infected executable to another

Worm

- ▶ Program that actively seeks for machines to infect
- ▶ Infects new machines by exploiting one or more software vulnerabilities
- ▶ Uses network connections, shared media email,...to spread from one machine to another

Overview

Malware Types by Spreading

- ▶ Viruses
- ▶ Worms
- ▶ Trojans

Initial Infection

- ▶ Malicious Attachments
- ▶ Installing malicious Applications
- ▶ Software Vulnerabilities
- ▶ Misconfigured access controls
- ▶ Social engineering

Botnets

- ▶ C&C Infrastructures
- ▶ DGAs
- ▶ Sinkholing

Typical Payloads

- ▶ DDoS Engines
- ▶ SPAM Engines
- ▶ Phishing Engines
- ▶ Information Stealing
- ▶ Miners

Command and Control Techniques

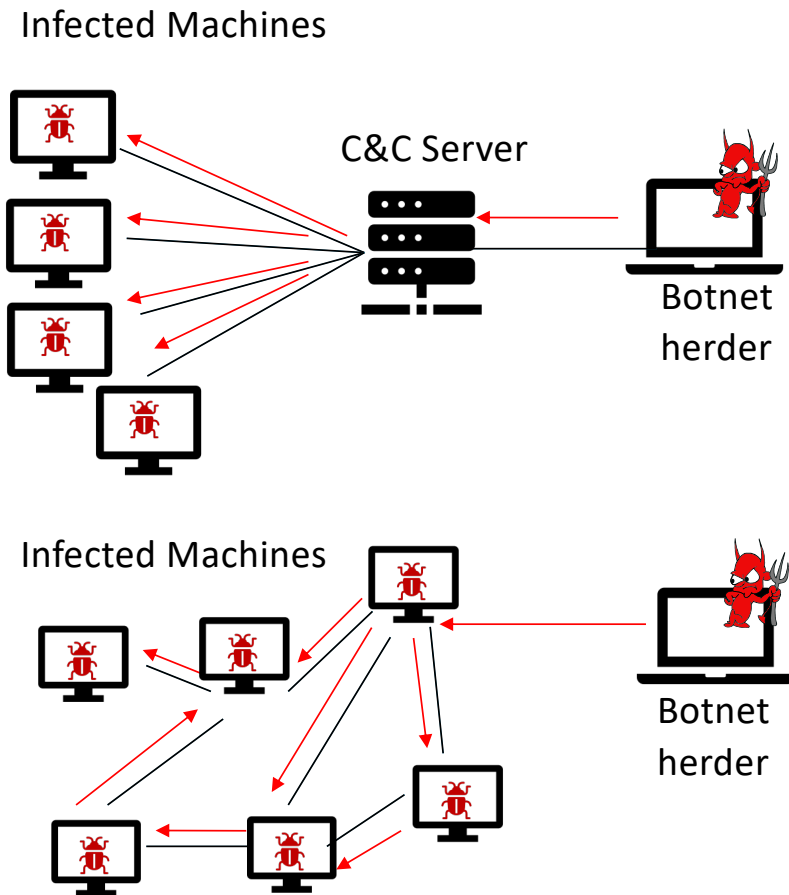
- **Centralized**

- ▶ Attacker operates central infrastructure to distribute commands to the victim machines
- ▶ Two main techniques used
 - IRC Servers: commands are pushed to connected clients
 - HTTP Servers: commands are pulled by victim clients

- **Decentralized**

- ▶ The victim machines form a P2P network
- ▶ Commands of an attacker are distributed from P2P directly

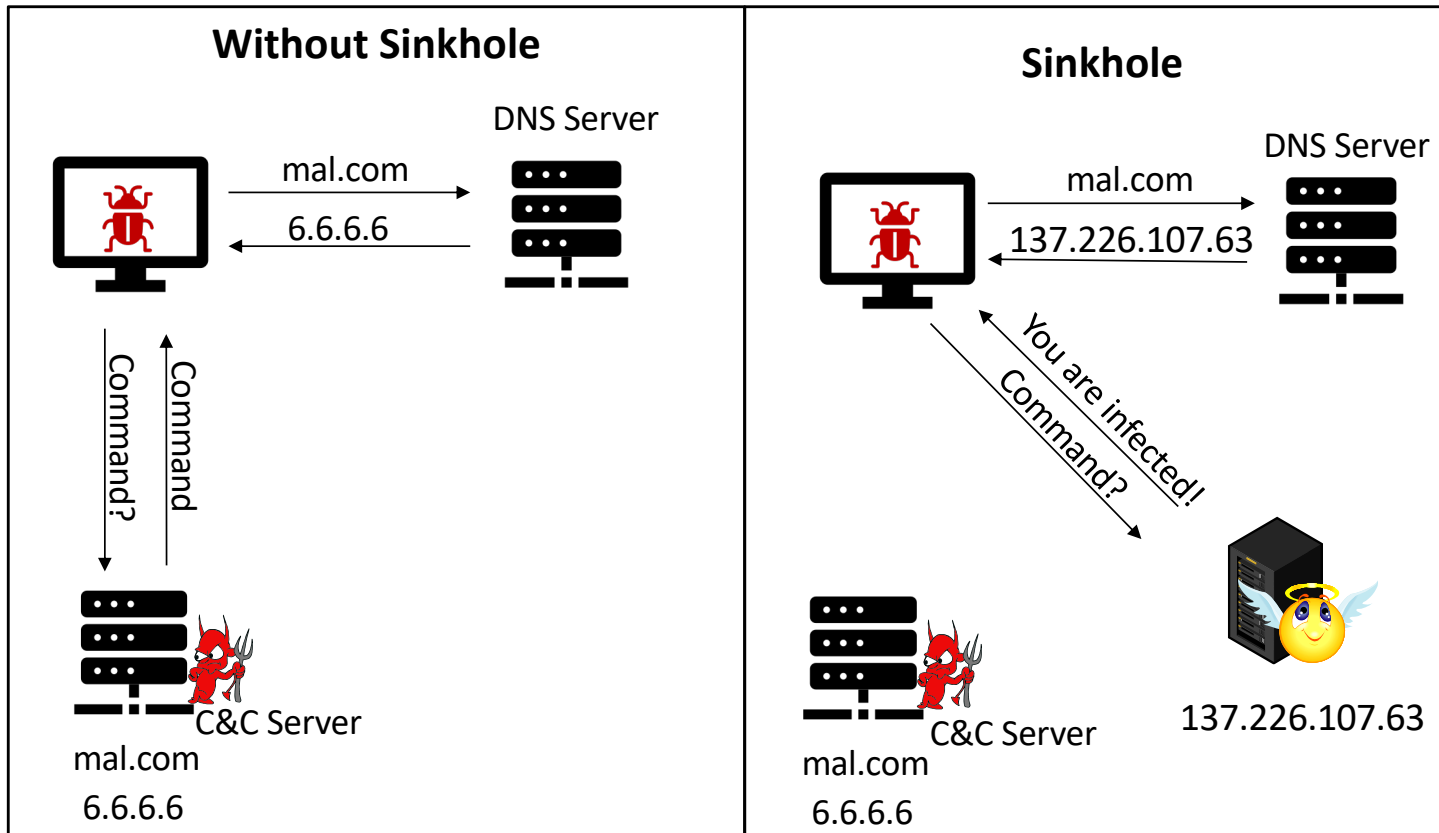
- **Many of today's bots are hybrid**



Taking Down a Centralized C&C Infrastructure

- **Locate C&C servers and take them down**
 - ▶ Analyze network traffic of infected machines
 - ▶ Analyze bot malware itself by reverse engineering the code
 - ▶ If it is C&C server is a compromised machine, contact legitimate owner
- **Make C&C server impossible to contact**
 - ▶ Block domain name in DNS
 - ▶ Block IP range of C&C infrastructure
 - ▶ Disconnect rogue hosting companies
- **Find out which devices in your network are infected by**
 - ▶ **Sinkholing** the corresponding domain names and see who connects
 - ▶ Automatically warn users of infected machines

DNS Sinkholing of known Malicious Domains



Hiding the IPs of C&C Servers to Impede Take Down

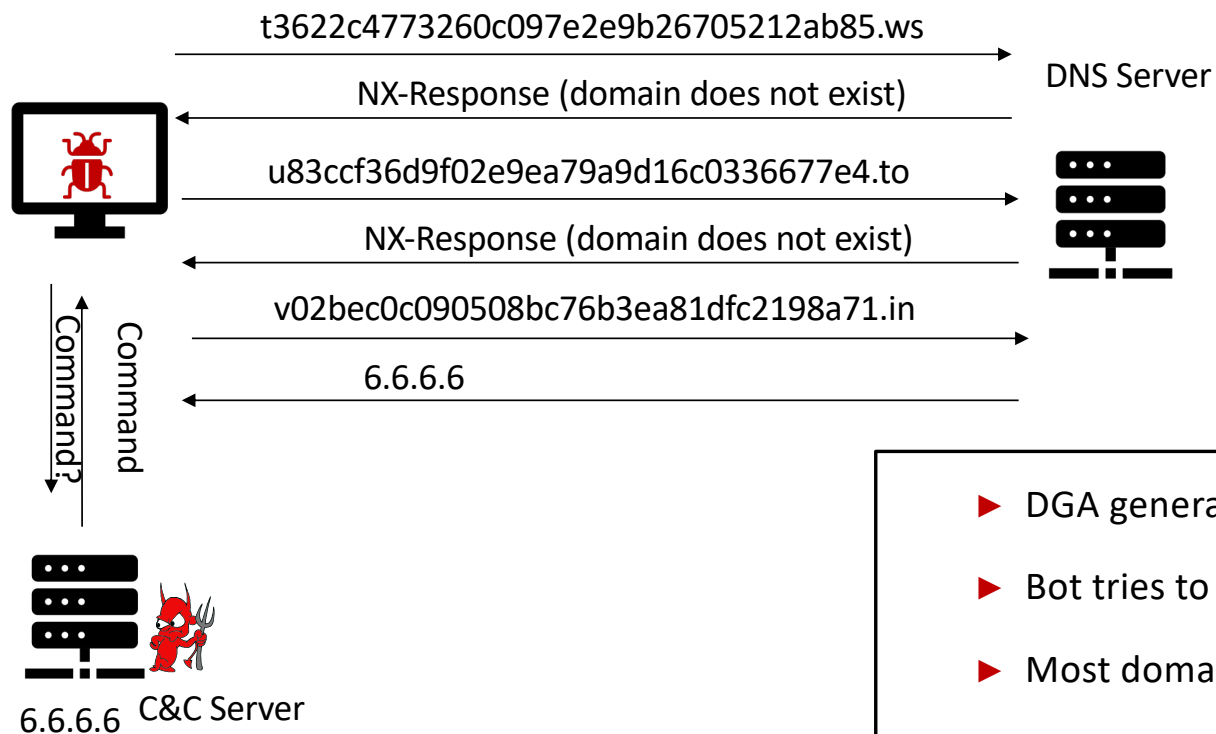
- **Use of Domain Generating Algorithms (DGAs)**

- ▶ Change domain name of machine queried for commands e.g. by an HTTP-bot based on a DGA using a seed (e.g. time stamp, twitter post,...) as input
- ▶ Domain names queried change frequently
- ▶ Attack has to register the queried domain names in order to be able to distribute commands
- ▶ If DGA and seed are known domain names can be blocked in local DNS

- **Use of Fast Flux in DNS**

- ▶ Multiple IP addresses associated with a single domain name, no **one** server to take down
- ▶ IP addresses quickly changed by changing DNS records
- ▶ IP addresses typically belong to compromised servers
- ▶ Still domain name can be blocked locally at DNS server on the victim's network

Hiding C&C Server by DGA



- ▶ DGA generates domains
- ▶ Bot tries to resolve domains
- ▶ Most domains are not registered
- ▶ Bot herder registers one or more domains per day
- ▶ Bot connects to C&C server and asks for commands

Malware Terminology

Name	Description
Advanced Persistent Threat (APT)	Sophisticated malware directed at specific business or political targets applied persistently and effectively
Adware	Advertising integrated in software, often results in pop-up ads or redirection of a browser to a commercial site
Attack kit	Set of tools for generating malware, including propagation and payload mechanisms
Auto-rooter	Malicious hacking tool used to remotely break into machines
Backdoor	Any mechanism that bypasses a security check, allows unauthorized access to functionality in a program or system
Downloader	Code that installs other items on a machine, e.g. loads a larger malware packed after initial infection
Drive-by-downloads	Uses code in a compromised web site that exploits a vulnerability in the browser or browser plugins
Exploit	Code specific to exploiting a single vulnerability or set of vulnerabilities
Flooder (DoS engine)	Generates large volume of data, e.g. to carry out denial of service attack

Malware Terminology

Name	Description
Key logger	Captures keystrokes on the infected system
Logic bomb	Code inside a malware, triggers when a specific condition is met
Macro virus	Uses macro or scripting code, typically embedded in document
Mobile code	Code that is portable between different platforms
Rootkit	Set of hacker tools used to hide the malware and gain root access
Spam engines	Used to send large volumes of unwanted email
Spyware	Collects information from a computer and transmits it to another system (e.g. key strokes, screen shots, network traffic...)
Trojan horse	Appears to be useful but also has a secondary malicious purpose
Virus	Tries to replicate itself into executable of script code when executed
Worm	Runs independently and propagates copies of itself, typically uses software vulnerability
Bot (Zombie)	Activated on an infected machine to gain remote control to launch attacks on other machines

Overview

Malware Types by Spreading

- ▶ Viruses
- ▶ Worms
- ▶ Trojans

Botnets

- ▶ C&C Infrastructures

Initial Infection

- ▶ Malicious Attachments
- ▶ Installing malicious Applications
- ▶ Software Vulnerabilities
- ▶ Misconfigured access controls
- ▶ Social engineering

Typical Payloads

- ▶ DDoS Engines
- ▶ SPAM Engines
- ▶ Phishing Engines
- ▶ Information Stealing
- ▶ Miners

Malicious Attachments

- Spread mostly over Emails but also over Instant Messengers and SMS
- May contain executable code or files with macro viruses
- Often used in connection with social engineering, e.g.,
 - ▶ Email pretending to be from some reputable business
 - Pretending to contain an order confirmation, tax information, bill,...
 - ▶ Email pretending to answer to job advertisements or call for bids, ...
 - Pretending to contain application papers, offers,...
 - ▶ Emails pretending to alert users of security breaches etc.
 - Pretending to contain cleaning software that urgently needs to be run,...
- E.g., according to BSI-Lagebild 2022:
 - ▶ 34 000 emails **per month** filtered in German government networks

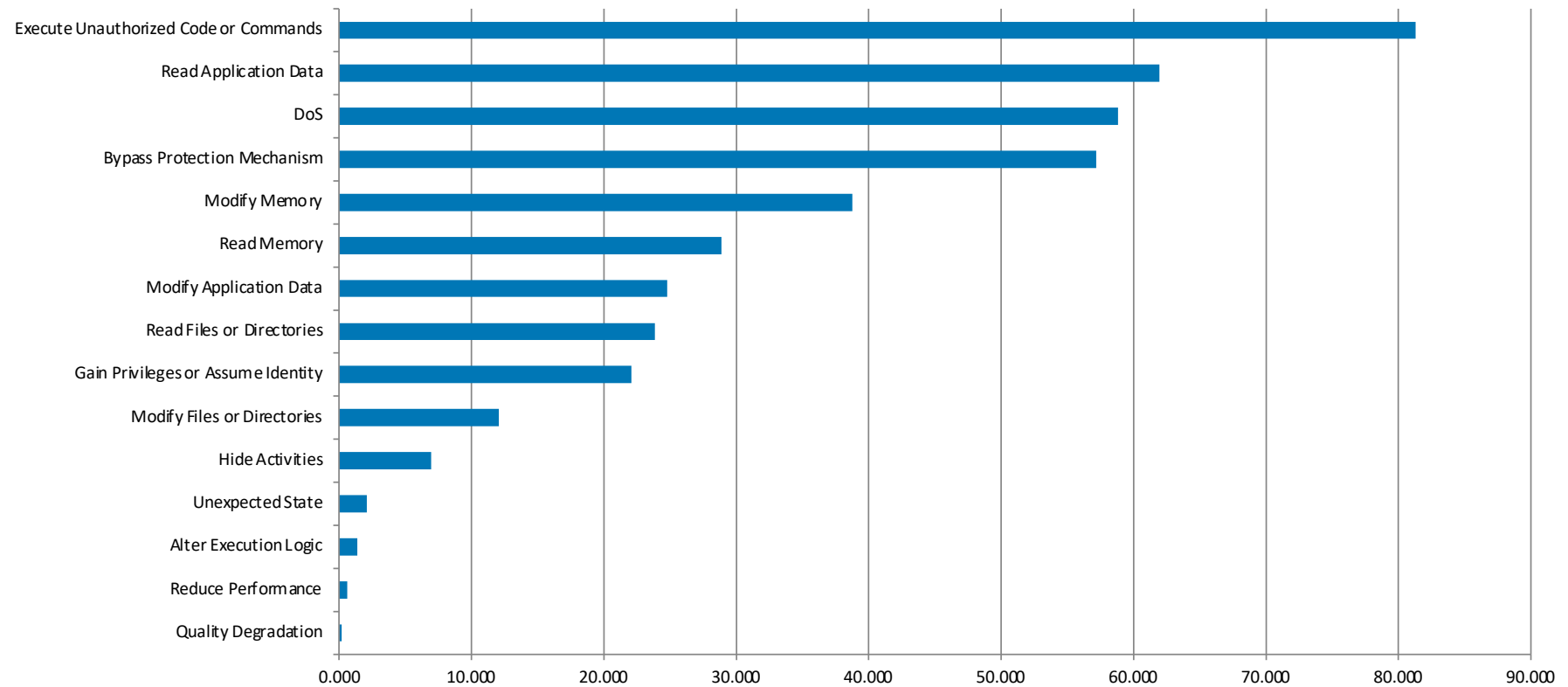


Installing Malicious Applications

- **Trojans are typically deliberately installed by users**
- **User tricked into installing them by claimed functionality**
 - ▶ Free versions of games
 - ▶ Free anti-virus products
 - ▶ ...
- **Most common strategy used to infect mobile devices still**

Software Vulnerabilities

Bekannt gewordene Schwachstellen nach möglicher Schadwirkung
Anzahl



© Bundesamt für Sicherheit in der Informationstechnik 2023

Example for Execution of Unauthorized Code: Buffer Overflow – Definition by NIST

Buffer Overflow: A condition at an interface under which more input can be placed into a buffer than the capacity allocated for it, 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

Example for a Basic Buffer Overflow in C Code

```
int main(int argc, char *argv[]) {  
    int valid = FALSE;  
    char str1[8];  
    char str2[8];  
  
    next_tag(str1);  
    gets(str2);  
    if (strncmp(str1, str2, 8) == 0)  
        valid = TRUE;  
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);  
}
```

Copies some expected tag value into str1

gets() does not do any length checking!

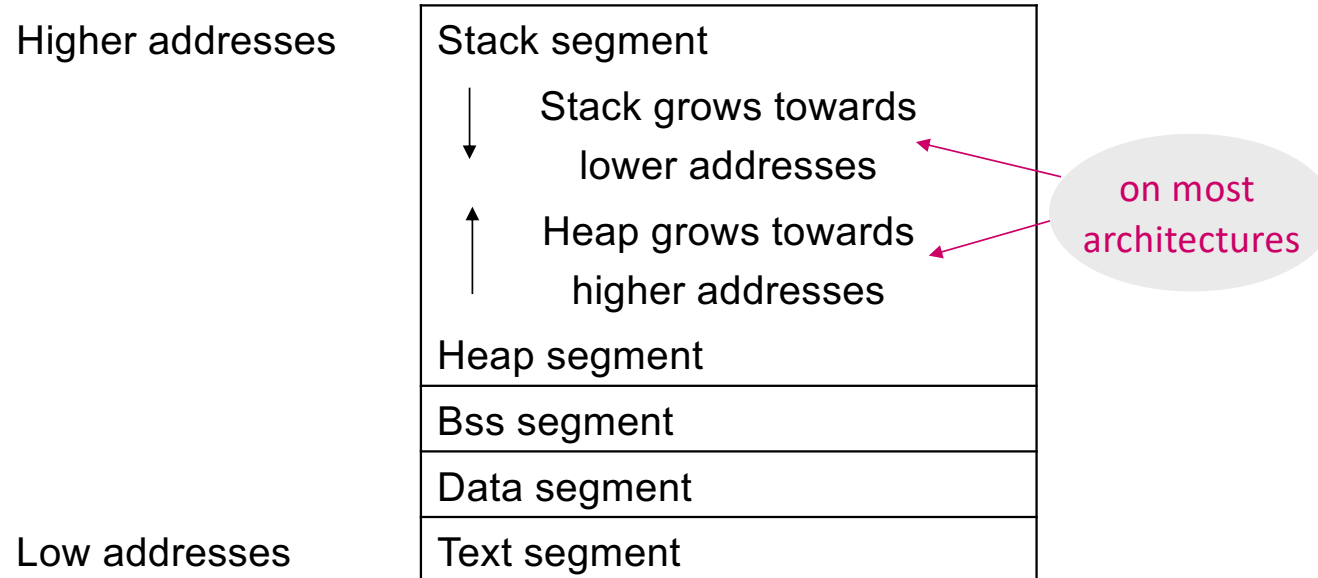
Assume tag is START

```
$ cc -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)
```

Basic Buffer Overflows

- **The simple example on the last slide results in a variable corruption**
 - ▶ Overly long input data overwrites memory location of another variable
 - ▶ This may already result in a serious attack
 - E.g., if `next_tag` contained a password to which the input (`str2`) is to be compared before access to some system resources are granted
- **More sophisticated buffer overflows target corruption of program control addresses in order to alter the flow of execution of the program**
- **To exploit any type of buffer overflow vulnerability 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
 - E.g., by inspecting the source code of a program or using fuzzing tools
 - ▶ Understand how that buffer will be stored in the processes memory and can thus be used to corrupt adjacent memory locations (architecture and compiler dependent)

Executable Program's Memory Segments



- **Compiled program's memory is divided into five segments**
 - ▶ text, data, bss, heap, stack
 - ▶ Text, data and bss segments are of static size,
 - ▶ Heap and stack shrink and grow dynamically during program execution

Stack Buffers

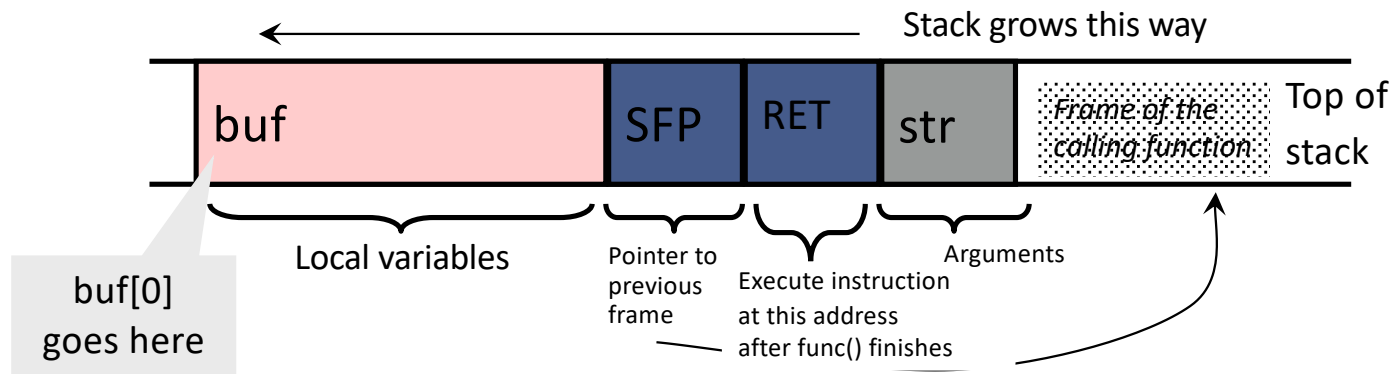
- Suppose Web server contains this function

```
void func(char *str) {  
    char buf[126];  
    strcpy(buf, str);  
}
```

Allocate local buffer
(126 bytes reserved on stack)

Copy argument into local buffer

- When this function is invoked, a new **frame** with local variables is pushed onto the stack



What If Buffer is Overstuffed?

- Memory pointed to by str is copied onto stack...

```
void func(char *str) {  
    char buf[126];  
    strcpy(buf, str);  
}
```

strcpy does NOT check whether the string at *str contains fewer than 126 characters

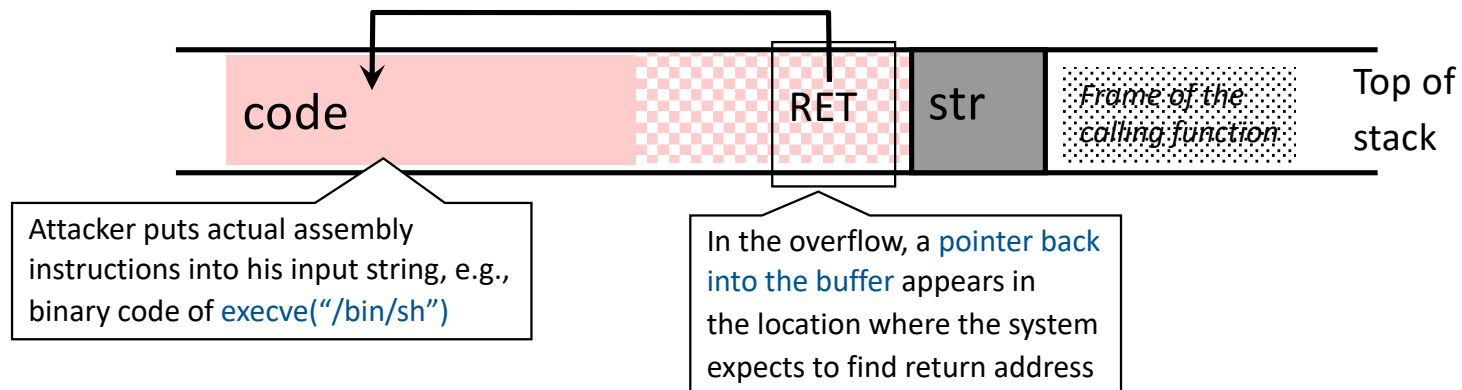
- If a string longer than 126 bytes is copied into buffer, it will overwrite adjacent stack locations



Executing Attack Code

- **Suppose buffer contains attacker-created string**

- ▶ For example, `*str` contains a string received from the network as input to some network service daemon



- **When function exits, code in the buffer will be executed, giving attacker, e.g., a shell**

- ▶ **Root shell** if the victim program is setuid root

Cause: No Range Checking

- **strcpy does not check input size**

- ▶ strcpy(buf, str) simply copies memory contents into buf starting from *str until “\0” is encountered
- ▶ ignores the size of area allocated to buf

- **Many C library functions are unsafe**

- ▶ strcpy(char *dest, const char *src)
- ▶ strcat(char *dest, const char *src)
- ▶ gets(char *s)
- ▶ scanf(const char *format, ...)
- ▶ printf(const char *format, ...)
- ▶ ...

Does Range Checking Help?

- **strncpy(char *dest, const char *src, size_t n)**

- ▶ If strncpy is used instead of strcpy, no more than n characters will be copied from *src to *dest
 - Programmer has to supply the right value of n

- **strncat(char *dest, const char *src, size_t n)**

- ▶ If strncat is used, then the first n characters from *src will be appended to *dest

- **Potential overflow in httpasswd.c (Apache 1.3):**

```
... strcpy(record, user);  
    strcat(record, ":");  
    strcat(record, cpw); ...
```

Copies username ("user") into buffer ("record"), then appends ":" and hashed password ("cpw")

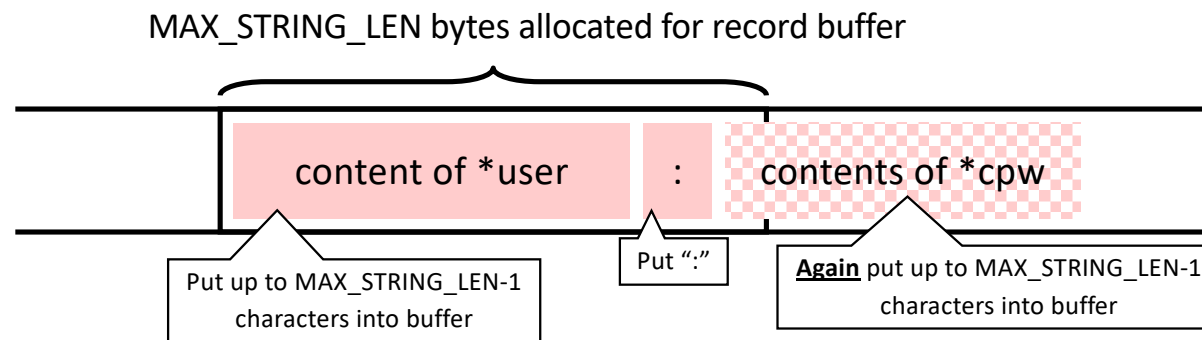
- **Published "fix" (do you see the problem?):**

```
... strncpy(record, user, MAX_STRING_LEN-1);  
    strcat(record, ":");  
    strncat(record, cpw, MAX_STRING_LEN-1); ...
```

Misuse of strncpy in htpasswd “Fix”

- Published “fix” for Apache htpasswd overflow:

```
... strncpy(record,user,MAX_STRING_LEN-1);  
   strcat(record,":");  
   strncat(record,cpw,MAX_STRING_LEN-1); ...
```



Defense against Buffer Overflows

- **Defense Mechanisms can broadly be divided into**

- ▶ **Compile time defenses**, which aim to harden new programs to resist attacks
- ▶ **Run-time defenses**, which aim to detect and abort attacks in existing programs

- **Compile-time defenses**

- ▶ Choose a high-level programming language that does not permit buffer overflows
 - Programs may still be vulnerable if existing system libraries are used
 - Disadvantage: direct access to some instructions and hardware resources lost
- ▶ Encourage safe coding standards
- ▶ Language extensions and use of safe standard libraries such as libsafe
- ▶ Include additional code at compile time to detect corruption of the stack frame at runtime
 - E.g. gcc extensions such as Stackguard, Stackshield, and Return Address Defender

Run-Time Defenses – Executable Address Space Protection

- **Typical memory exploit involves code injection**
 - ▶ Put malicious code at a predictable location in memory, usually masquerading as data
 - ▶ Trick vulnerable program into passing control to it
 - Overwrite saved EIP, function callback pointer, etc.
- **Idea: Make stack and other data areas non-executable**
 - ▶ Needs to be supported by the processor's memory management unit
 - Tag pages of virtual memory as non-executable
 - ▶ Some useful functionality also uses executable code on the stack, e.g., nested functions in C, Linux signal handlers,...
- **Support has become standard in most modern operating systems**
 - ▶ Protects against classic overflows, where shellcode is included in stack buffer
- **Consequence:**
 - ▶ Newer buffer overflow exploits use more sophisticated techniques such as using code already existing on the target machine,...

Misconfigured Access Controls

- **Examples for misconfigurations include**
 - ▶ Weak user-selected passwords
 - ▶ Weak default passwords that are not changed
 - ▶ Open port such as open ssh port
 - ▶ ...

Social Engineering

- **Essential part of many already mentioned infection paths**
 - ▶ Malicious attachments
 - ▶ Installing malicious applications
 - ▶ ...
- **Other examples**
 - ▶ Trick users into revealing their password
 - ▶ Trick administrators into resetting passwords of specific users
 - ▶ Trick users on the phone / via email
 - ▶ Trick users into entering account credentials into fake websites
 - Phishing
 - ▶ ...

Overview

Malware Types by Spreading

- ▶ Viruses
- ▶ Worms
- ▶ Trojans

Botnets

- ▶ C&C Infrastructures

Initial Infection

- ▶ Malicious Attachments
- ▶ Installing malicious Applications
- ▶ **Software Vulnerabilities**
- ▶ Misconfigured access controls
- ▶ Social engineering

Typical Payloads

- ▶ DDoS Engines
- ▶ SPAM Engines
- ▶ Phishing Engines
- ▶ Information Stealing
- ▶ Miners

Examples for Malicious Purposes aka Payload

Ransome Ware

- ▶ Encrypt all or some files on the victim machine
- ▶ Ask for ransom to release encryption key
- ▶ Makes use of crypto currencies for payment

DDoS Engine

- ▶ Enables infected machine to participate in DDoS attacks w/o user's consent

Data Theft and Espionage

- ▶ Steal sensitive information from infected machine

SPAM or Phishing Engine

- ▶ Engines that allow to sent spam or phishing emails from the victim machine

Key Logger or general Spyware

- ▶ Logs a user's keystrokes and stores them
- ▶ Sends them off to the attacker
- ▶ Thereby steals, e.g., account credentials, credit card information...
- ▶ Turn on camera remotely to spy

Examples for Payloads and Additional Malicious Functionalities

Crypto Miners

- ▶ install a malicious program on the victim's host that helps in mining crypto currencies
- ▶ Runs in the background and typically uses computing resources while victim machine is idle
- ▶ Spreads the energy consumption and computing time over multiple victim machines

Bot

- ▶ enables attacker to remotely control an infected machine via a command-and-control infrastructure

Rootkit

- ▶ allows to maintain covert root access to the infected machine
- ▶ hides any evidence of its presence, e.g., by installing malicious versions of standard system programs such as netstat, ps, ls, du, et.

References

- **W. Stallings, Cryptography and Network Security: Principles and Practice, 8th edition, Pearson 2022**
 - ▶ Chapter 21: Network Endpoint Security
 - 21.3 Malicious Software
- **Wenliang Du, Computer Security a Hands-on Approach, 3rd edition, 2022**
 - ▶ Chapter 4: Buffer Overflows



IT-Security

Summary

Prof. Dr.-Ing. Ulrike Meyer



Chapter 1: Introduction

- **Security goals**
 - ▶ Confidentiality, Integrity, Availability
- **Examples for attacks against these goals**
- **Definition of security services and security mechanisms**
 - ▶ Which of them aim at prevention, detection, or deterrence
- **Categorization of attackers according to**
 - ▶ Skills
 - ▶ Knowledge on and access to target
 - ▶ Computational resources
 - ▶ Motivation

Chapter 2: Symmetric Encryption (1)

- **Definition of an encryption scheme**
- **Kerckhoffs' principle**
- **Examples for classical ciphers**
 - ▶ Caesar cipher (easily breakable with brute force due to short key length)
 - ▶ Monoalphabetic substitution cipher (easily breakable with frequency analysis)
- **Perfect Secrecy**
 - ▶ Shannon's theorem
 - ▶ One-time pad and perfect secrecy of the one-time pad
 - ▶ Practical problems with the one-time pad
- **Computational Security**

Chapter 2: Symmetric Encryption (2)

- **Modeling attacks against ciphers**

- ▶ W.r.t power of the attack (ciphertext-only attack, known-plaintext attack...)
- ▶ W.r.t. attack result ((partial) plaintext recovery, (partial) key recovery)
- ▶ W.r.t technique used (brute force, time-memory trade-off, differential, algebraic..)

- **Block ciphers versus stream ciphers**

- ▶ How are they defined
- ▶ What's the problem with key stream re-use when a stream cipher is used

- **Basic facts on DES, 2DES, 3DES, AES**

- ▶ Key sizes, block sizes, attacks
- ▶ Meet-in-the-middle attack on 2DES

Chapter 2: Symmetric Encryption (3)

- **Modes of encryption (ECB, CBC, CFB, OFB, CTR, GCM)**
 - ▶ Encryption / decryption, properties
- **Stream ciphers and block ciphers alone do not provide integrity**
 - ▶ Understand that plaintext encrypted with a stream cipher can be changed by anyone

Chapter 3: Integrity (1)

- **Definitions for**

- ▶ hash function, cryptographic hash function, pre-image resistance, 2nd pre-image resistance, collision resistance, relations between the properties

- **Complexity of brute force attacks against ideal hash functions**

- **Basic facts on MD-5, SHA-1, SHA-2, SHA-3**

- ▶ Length of hash value, broken / not broken (yet ;-))

- **Definition message authentication code**

- **HMAC, CMAC constructions in detail**

- ▶ Including advantage of HMAC over other constructions
- ▶ Including advantage of CMAC over CBC-MAC

Chapter 3: Integrity (2)

- **Methods for replay protection**
- **Ways to combine integrity protection and encryption**
- **Galois Counter Mode (GCM)**

Chapter 4: Asymmetric Cryptography (1)

- **RSA key generation, encryption, decryption in detail**
 - ▶ Extended Euclidian algorithm
 - ▶ Including security proofs and why we need Optimal Asymmetric Encryption Padding
 - ▶ Details on how OAEP works and adds semantic security to RSA
- **Symmetric versus asymmetric encryption**
- **RSA Backdoors general idea and examples**
- **Definition of digital signatures**
 - ▶ Details of RSA-Signatures (with hashing)
 - ▶ Why we hash messages before signing

Chapter 4: Asymmetric Cryptography (2)

- **Type of attacks against signature schemes**
 - ▶ wrt power of attacker (key-only etc...)
 - ▶ wrt result of the attack, e.g. total break, existential forgery,...
- **Type of attacks against signature schemes**
 - ▶ wrt power of attacker (key-only etc...)
 - ▶ wrt result of the attack, e.g. total break, existential forgery,...
- **Comparison of MACs and digital signatures**
- **Details on key generation, signature generation/verification in DSS**
- **Details on Diffie-Hellmann key agreement and MitM against DH**

Chapter 5: Authentication and Key Agreement (1)

- **Definition of entity authentication**

- ▶ Correctness, resistance against transfer, impersonation resistance
- ▶ mutual vs. unilateral authentication

- **Example Building Blocks for unilateral and mutual authentication**

- ▶ With time stamps, with random challenges, with signatures, with MACs
- ▶ Understand the problem of reflection attacks in this context

- **Definiton of the properties of session key establishment protocols**

- ▶ key agreement vs. key transport protocols
- ▶ authenticated key establishment
- ▶ explicit key authentication, implicit key authenticaiton, key freshness, perfect forward secrecy, known key attacks

Chapter 5: Authentication and Key Agreement (2)

- **Analyze key establishment protocols w.r.t. these properties**
- **Diffie-Hellmann,**
 - ▶ Man-in-the middle attack in DH, implicit key authentication in Diffie-Hellmann, authenticated DH
- **Trusted Third Parties in Key Establishment**
 - ▶ Main idea of Key distribution center, example protocol
 - ▶ Main idea of Certification authorities,
 - Example authenticated DH with certificates
 - Content of a certificate
 - Certificate verification
 - Certificate revocation
 - Chains of certificates

Chapter 5: Authentication and Key Agreement (3)

- **Typical password-based authentication between client and server**
 - ▶ Relation between randomly selected passwords and effective key length
 - ▶ Password based user authentication by a server
 - Advantage storing cryptographic hashes of passwords over plaintext / encrypted storage
 - ▶ Purpose of salting passwords
 - ▶ Dictionary attacks on password files
- **Typical password-based authentication between two peers**
 - ▶ MAC-keys generated from password
 - ▶ Vulnerability against offline password cracking

Chapter 6: Network Security Protocols (1)

- **IPSec**

- ▶ Transport mode vs tunnel mode
- ▶ Security services offered by ESP and AH
 - What does an IP packet look like that is protected with ESP/AH in tunnel/transport mode
 - Which part of the packet is encrypted/integrity protected in ESP/AH in tunnel/transport mode
- ▶ Fields in AH and ESP protocol headers / ESP trailer
- ▶ Replay protection in ESP and AH
- ▶ Main content of SAs and SA selectors
- ▶ Inbound / outbound processing overview

Chapter 6: Network Security Protocols (2)

- **IPSec**

- ▶ IKE v2 protocol details

- In particular: how do initiator and responder authenticate each other?
 - What's the basis for the key agreement?
 - How are security algorithms for IKE itself / for ESP and/or AH negotiated?

- **TLS 1.3**

- ▶ Understand the details of the handshake protocol

- Different options to authenticate the handshake (mutual or unilateral authentication with signatures, PSK-based authentication only, DH with PSK)
 - Properties these different options have

- **Comparison between IPSec and TLS including main use cases**

Chapter 7: Email, DNS, SSH (1)

- **Email Security**

- ▶ End-to-end vs hop-by-hop protection of email
- ▶ End-to-end security goals
- ▶ Basic principle used in PGP and S/MIME (hybrid encryption, signatures for non-repudiation...)
- ▶ Web of trust in PGP
 - Introducer Trust, certificate trust, key legitimacy
- ▶ Main ideas of DKIM, SPF, and DMARC

Chapter 7: Email, DNS, SSH (2)

- **DNS**

- ▶ General operation of DNS

- Concept and types of resource records
- Recursive and iterative queries
- Purpose of caching

- ▶ Security issues of DNS

- Authenticity of resource records
 - cache poisoning
- Confidentiality

- ▶ DNSSec

- New types of resource records
- Keys used in DNSSec and how they are distributed and authenticated

Chapter 7: Email, DNS, SSH (3)

- **SSH**

- ▶ Details on the transport layer protocol

- Including the mandatory key exchange method
- Including algorithm negotiation

- ▶ User authentication protocol

- Including the details on the three user authentication protocols (public key, password, host-based)

Chapter 8: Denial of Service Attacks

- **Classification of DoS attacks w.r.t. the type of resource they target**
 - ▶ network bandwidth, system resources, application resources
 - ▶ Example attack for each type
 - Flooding with ICMP echo requests, SYN Flooding, HTTP Flood
- **Source address spoofing**
- **DDoS attacks**
- **Principle of a reflection attack**
 - ▶ Amplification attack as a subtype of reflection attacks
- **Preventive defense mechanisms**

Chapter 9: Access Control, Firewalls, IDSs (1)

- **Access Control**

- ▶ Discretionary vs. Mandatory access control
- ▶ Access control subjects, objects rights
- ▶ Access control matrices and Access control lists
- ▶ How do different Discretionary access control systems differ
 - Who can change acl associated with an object
 - How ACLs apply to privileged user
 - Support of groups and wildcards
 - Handling of contradictory permissions
 - Default settings

Chapter 9: Access Control, Firewalls, IDSs (2)

- **Access Control**

- ▶ Access Control in UNIX file systems

- rights
 - changing rights
 - meaning of rights on directories
 - user ids

- ▶ Roll based Access Control

- Main idea

- ▶ Attribute based access control

- Main idea

Chapter 9: Access Control, Firewalls, IDSs (3)

- **Firewalls**

- ▶ Packet filters
- ▶ Firewall policy
 - First match policy
 - Comprehensiveness of a fire wall policy
- ▶ Interpret rules in a simple packet filtering policy
 - Find redundant rules
 - Find (half-)shadowing rules
 - Combine rules
- ▶ Stateful firewall and why we need them
- ▶ What is a DMZ

Chapter 9: Access Control, Firewalls, IDSs (4)

- **Intrusion Detection Systems**

- ▶ Components of an IDS
- ▶ Basic assumption underlying any IDS
- ▶ Definition of detection rate and false alarm rate
- ▶ Base rate fallacy problem
- ▶ Anomaly detection vs. misuse (signature based) detection
- ▶ Host based vs. network-based intrusion detection
- ▶ Inline vs. passive network-based intrusion detection

Chapter 10

- **Types of Malware w.r.t. spreading**
 - ▶ worms, viruses, trojans
- **Botnets**
 - ▶ Command and Control Infrastructures
 - ▶ DGAs
- **Buffer Overflows**
 - ▶ Basic principle
 - ▶ Explain on an example if given a vulnerable piece of code
 - ▶ Types of defenses
- **Typical malware payloads**

Good Luck!



... and don't forget to look at the **exercises** and **e-tests** as well!!!!