OS11: Security

Including parts of Chapter 11 and Section 9.6.3 of [Hai19]

(Usage hints ⬀ for this presentation)

Computer Structures and Operating Systems 2023
Dr. Jens Lechtenbörger (License Information)

Data Science: Machine Learning and Data Engineering (Prof. Gieseke)
Dept. of Information Systems
WWU Münster, Germany

Speaker notes

- To toggle these notes, press `v`
  - If a slide contains audio, notes might show transcript
- Press `?` for key bindings (in particular, `a`, `o`, `n`, `p`, `Ctrl-Shift-f`)
- Presentations support two different PDF formats, see usage notes ⬈
  - Both hyperlinked on index page
    - Concise PDF format (replace `.html` and whatever follows in address bar ⬈ with `.pdf`)
    - Print browser view to PDF (add `?print-pdf` after `.html`, then print to PDF; suggested settings ⬈)
- If you find the amount of outgoing links to be distracting, see usage notes ⬈
  - Add `?hidelinks` (maybe with a number) after `.html`
- See usage notes ⬈ for other non-obvious features

1. Introduction

1.1. OS Plan

- OS Overview ⧉ (Wk 20)
- OS Introduction ⧉ (Wk 21)
- Interrupts and I/O ⧉ (Wk 21)
- Threads ⧉ (Wk 23)
- Thread Scheduling ⧉ (Wk 24)
- Mutual Exclusion (MX) ⧉ (Wk 25)
- MX in Java ⧉ (Wk 25)
- MX Challenges ⧉ (Wk 25)
- Virtual Memory I ⧉ (Wk 26)
- Virtual Memory II ⧉ (Wk 26)
- Processes ⧉ (Wk 27)
- **Security ⧉ (Wk 28)**

Applications (Users)

System Call Interface

Wk 20: OS Introduction

Wk 27/28: **Processes** and **security**
- OS isolates processes
- OS supports access control
- OS may support crypto primitives

Wk 26: Threads need **memory**
- Instructions and data
- OS supports virtual memory

Wk 24/25: **Concurrency**, **MX**
- Primitives provided by OS
- Lots of challenges

Wk 22: Thread **scheduling**

Wk 22: **Threads** as units of computation

Wk 21: **I/O** and **Interrupts**

Hardware as basis
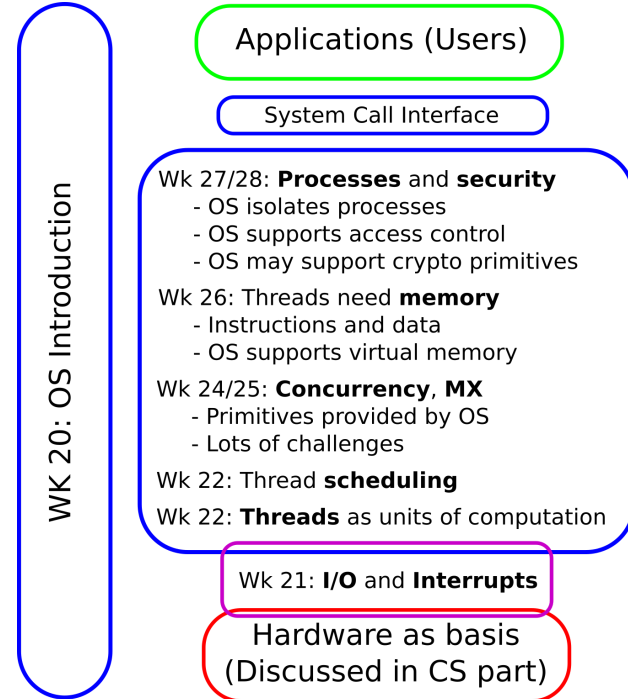(Discussed in CS part)

Table of Contents

- 1. Introduction
- 2. Cryptography
- 3. Message Integrity
- 4. OS Context
- 5. Conclusions

1.2. Today's Core Questions

- How can I ensure that my downloaded software has not been manipulated?
- What is e-mail self-defense?

1.3. Learning Objectives

- Explain confidentiality and integrity as security goals
  - Discuss differences between end-to-end and hop-by-hop goals
- Explain use of hash values and digital signatures for integrity protection and discuss their differences
  - Create and verify digital signatures (on e-mails and files/software)

1.4. Retrieval Practice

- Security — So far
  - Hardware building blocks
    - Kernel mode vs user mode⬈: Restrict instruction set
      - Protect kernel data structures
      - Enable access control via system call API
    - Timer interrupts⬈
      - Transfer control periodically back to OS
  - Process⬈ as major OS abstraction
    - Virtual address spaces
      - Isolate processes from each other
    - Access rights

1.4.1. Quiz on Hashing

Hashing is a basic technique.

1. Select correct statements about hash functions.

☐ Hash functions are functions in the mathematical sense.

☐ Hash functions map arbitrary-sized data to fixed-sized data.

☐ A hash collision occurs if two pieces of data are mapped to the same hash value.

☐ If a hash function maps arbitrary-sized data to fixed-sized data, an infinite amount of hash collisions is guaranteed.

☐ As hash collisions occur, hash functions cannot be invertible.

☐ Given a hash value h(x), it is impossible to compute the original data x from which the hash value was computed. (This points to a fundamental difference of hashing and encryption: Given an encrypted piece of data, say e(x), it is possible to compute the original data x [using a suitable decryption function and decryption key].)

1.5. Information Security

- **Safety**: Protection against unintended/natural/random events
    - (Not focus here; requires proper management, involves training, redundancy, and insurances)

- **Security**: Protection against deliberate attacks/threats
    - Protection of **security goals** for objects and services against **attackers**

1.5.1. Security Goals

- Classical security goals: **CIA triad**
    - **Confidentiality**
        - Only intended recipient can access information
        - Typically guaranteed by encryption mechanisms
            - (Or, e.g., with envelopes and protecting laws)
    - **Integrity**
        - Detection of unauthorized modification
        - Typically guaranteed by cryptographic checksumming mechanisms
            - (Or, e.g., with signatures and/or seals)
    - **Availability**
        - Information and functionality available when requested
        - Supported by redundancy
    - Further goals
        - Accountability, authenticity, anonymity, (non-) deniability, …

1.5.2. Relativity

- Security is **relative**
  - You need to **define your goals** and risks for **specific pieces** of information, e.g.:
    - How much confidentiality for course slides vs course exam?
    - Apparently, it's easy to keep the slides "secure"
      - Harder for the exam
  - Also: Who is the **attacker** with what **resources**?
    - Select appropriate security mechanisms, typically with **risk acceptance**
- Security via **design process** and **management**
  - BSI (Germany) and ISO standards
    - IT-Grundschutz 🚀
  - Topic in its own right

1.5.3. Attacker Models

- Sample classifications of attackers
  - Strategy
    - Targeted (specialized, looks for "weakest link")
      - E.g., espionage, blackmailing
    - Opportunistic (standardized, looks for "weakest target")
      - E.g., phishing, extortion, bot/zombie creation (DDoS, spam, bitcoin mining, proxy)
  - Financial resources
  - Compute capacity
  - Time
  - Knowledge (insider and position?)

1.6. Design Principles for Secure Systems

- Selected principles based on [SS75]
  - Fail-safe defaults (whitelisting): If no explicit permission, then deny
  - Least privilege (need to know): Subject has only those privileges that are necessary for given task
  - Economy of mechanism: Security mechanisms should be as simple as possible
  - Complete mediation: All accesses need to be checked
  - Open design: Security should not depend on secrecy; instead open reviewing
  - Separation of privilege: Permission not based on single condition
  - Psychological acceptability: Security should not hinder usage
  - And more, see [SS75] or [Hai19]

1.7. End-to-End Security

- Security goals may have varying **scope**
  - Hop-by-hop
  - End-to-end
- Integrity and confidentiality are **end-to-end goals**
  - Beware: That's not generally understood!
    - (See next slide…)
  - Consider hop-by-hop confidentiality
    - Alice wants to send confidential message M to Bob via one hop, Eve
      - Alice encrypts M for Eve, sends encrypted M to Eve
      - Eve decrypts M, encrypts M for Bob, sends encrypted M to Bob
    - Security gain or loss? (Compared to what?)
  - Hop-by-hop integrity similarly

Suppose that you want to send some e-mail to a friend, where the e-mail's contents are a private matter. In this case, the security goal confidentiality needs to be protected. Quite likely, you want confidentiality as an end-to-end goal meaning that only the communication endpoints, namely you and your friend, can read the message, independently of the number of hops or intermediary machines (such as Internet backbone routers) that forward the message from you to your friend.

If you send the e-mail as usual, sender and recipient need a password to access their accounts and e-mails at their providers' servers. Thus, some protection is offered for e-mails at their destinations. However, obviously also the providers' administrators and everybody else with access to their infrastructures (such as intelligence agencies violating human rights and other criminals) have access to the e-mails. Thus, those parties can access your draft folder as well as the recipient's inbox to access messages, violating confidentiality.

Besides, in the case of e-mail it is not clear whether e-mails forwarded between providers are encrypted or not. In response to the Snowden revelations there is a major shift towards encryption in transit; however, this type of encryption is not guaranteed. Thus, your e-mail might also traverse the Internet in plaintext, and on its way it typically passes a couple of computers owned by parties that are unknown to you and that might copy or change your e-mails. Actually, when e-mails cross country borders it's almost certain that intelligence agencies copy the messages, again violating confidentiality. Obviously, this type of confidentiality violation can be prevented if providers encrypt their message exchanges, which would guarantee confidentiality on a hop-by-hop basis.

Clearly, encryption on a hop-by-hop basis is better than no protection, while you need to take protection into your own hands if you are interested in end-to-end goals.
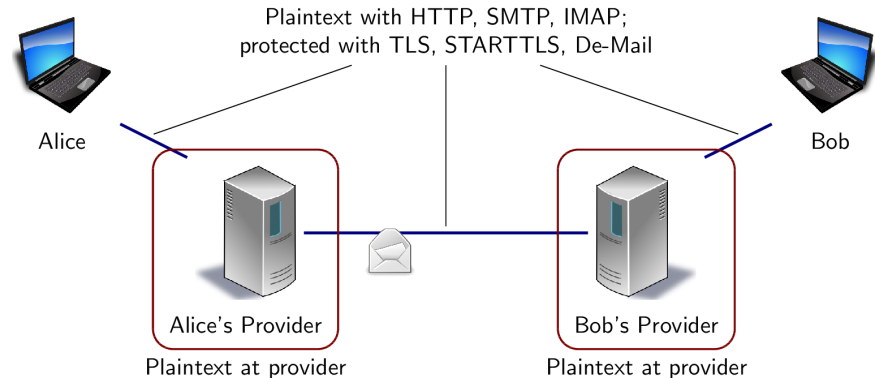
1.7.1. (Counter-) Example: De-Mail

- De-Mail 🚀 is a German approach **defining** legally binding, "secure" e-mail
  - General picture
    - Strong (hop-by-hop) security for each of the three blue links
    - Plaintext at both providers (and broken approach towards integrity, see [Lec11])
      - End-to-end encryption allowed
      - Digital signatures used in special cases

Plaintext with HTTP, SMTP, IMAP;
protected with TLS, STARTTLS, De-Mail

Alice

Bob

Alice's Provider
Plaintext at provider

Bob's Provider
Plaintext at provider

De-Mail serves as example for hop-by-hop security and as counter-example for end-to-end security. Key characteristics are shown on this slide. While De-Mail may be attractive for legal reasons when it allows to replace paper with digital communication, I don't see much value for individuals.

The broken aspect of integrity protection mentioned here is that the technical specification for De-Mail includes a step "Metadaten setzen und Integrität sichern" which adds a simple hash value that is later checked in a step called "Integritätssicherung prüfen". As part of a self-study assignment you should convince yourself that such a hash value provides no integrity protection against attackers.

2. Cryptography

2.1. Key Notions

- Cryptography = Art of "secret writing"
- Set of mathematical functions
  - Cryptographic **hash** functions
  - Classes of **encryption** algorithms
    - **Symmetric**, **secret key**: en- and decryption use the same shared secret key
    - **Asymmetric**, **public key**: participants own **pairs** of secret (decryption, signature creation) and public (encryption, signature verification) keys
    - **Hybrid**: asymmetric initialization to establish symmetric keys for encryption
  - Basis for various security mechanisms
- Performance
  - Hashing > Symmetric Enc. > Asymmetric Enc.
    - (One can hash more data per second than one can encrypt)
    - (One can encrypt more data per second symmetrically than asymmetrically)

2.1.1. Basic Assumptions

- Fundamental Tenet of Cryptography from [KPS02]
  - "If lots of smart people have failed to solve a problem, then it probably won't be solved (soon)."
  - The problem to solve here: Break specific crypto algorithm
    - If that did not happen for a long time, probably the algorithm is strong
    - (Lots of crypto algorithms come without security proof)

- Kerckhoffs' Principle⤤ (1883)
  - Security of crypto systems should not depend upon secrecy of en- and decryption functions (but on secrecy of the used keys)
  - "Open Design" principle from [SS75]
    - Not respected in national security/military/intelligence settings in Germany
      - From Enigma through Libelle (approved for "Streng geheim"; developed by BSI, not published)
  - Opposite: Security through obscurity

2.1.2. Names

- Alice and Bob; Charlie, Carol, Dave, …
  - Communicate frequently
  - Value their privacy
  - Have limited trust in third parties
  - Appeared to be subversive individuals in the past
    - Growing understanding in general public
  - And, of course, politically correct names instead of "A" and "B"

- Eve, Mallory, Trudy
  - Eavesdropper, malicious attacker, intruder

2.1.3. Notation

- M, C: Message and ciphertext (encrypted messages)
- K: Key (random bits, maybe with certain structure)
- E, D: En- and decryption functions
- K_{AB}: Secret key shared between Alice and Bob
- K_{A-}: Alice's private key
- K_{A+}: Alice's public key
- K(M): Message M encrypted with key K (if function E is clear from context)
- $[M]_K$: Message M signed with key K

2.2. GnuPG

- ## GNU Privacy Guard⬀
  - Free software for (e-mail) encryption and digital signatures
  - Implementation of OpenPGP standard 🚀
    - Secure e-mail based on hybrid cryptography
  - In addition, lots of cryptographic algorithms via command line
    - `gpg --version` … gpg (GnuPG) 2.1.13 … Öff. Schlüssel: RSA, ELG, DSA, ECDH, ECDSA, EDDSA Verschlü.: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH, CAMELLIA128, CAMELLIA192, CAMELLIA256 Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
  - Start by creating key pair: `gpg --gen-key`

2.2.1. E-Mail Self-Defense

- My suggestion: Try out OpenPGP
  - Create key pair, upload public key to server, send/receive encrypted (possibly signed) e-mails
- More specifically, follow Email Self-Defense 🚀
  - GnuPG and Thunderbird
  - Of course, other implementations exist
    - The choice is yours
  - Note: That guide contains instructions concerning the **e-mail robot Edward**, which can reply to your encrypted (and signed) test e-mails

2.3. (Cryptographic) Hash Functions

- Hash function ⬚ (or message digest)
  - Input: Message M (bit string of arbitrary length)
  - Output: Hash value H(M) (bit string of **fixed** length)
    - Computation is one-way: Given H(M), we cannot compute M
  - Collision: Different messages mapped to same hash value
- **Cryptographic** hash value ≈ **digital fingerprint**
  - Collision **resistant** (different hash values for different messages)
  - Weak collision resistance of hash function H
    - Given message M it is computationally infeasible to generate M' such that H(M) = H(M')
      - (Computationally infeasible means that attackers should not be able to create collisions due to resource or time limitations)
  - Strong collision resistance of hash function H
    - Computationally infeasible to generate M and M' such that H(M) = H(M')

I suppose that you remember hash functions for fast searching. Recall that hash collisions are to be expected.

With cryptographic hash functions, collisions are a Bad Thing since hash values are supposed to serve as digital fingerprints. Ideally, each message (or document or piece of data or code) should have its own, unique fingerprint. When a message is changed, also its fingerprint should change. However, if a hash collision occurs and two messages produce the same cryptographic hash value, the fingerprint becomes unusable to distinguish them.

On the slide you see two versions of collision resistance. Please take a moment to convince yourself that the strong version implies the weak version.

2.3.1. On Collision Resistance

- Later ▶: Hash values are essence of digital signatures
  - Consider contract between Alice and Mallory
    - "Mallory buys Alice's used car for 20,000€"
      - Contract's text is message M
      - Digital signatures of Alice and Mallory created from H(M)
  - Suppose H not weakly collision resistant
    - Mallory may be able to create M' with price of 1€ such that H(M) = H(M')
    - As H(M) = H(M'), there is no proof who signed what contract

- Birthdays, collisions, and probability
  - Hash people to their birthdays (day and month, without year)
  - (a) Weak collision resistance: Anyone sharing *your* birthday?
  - (b) Strong collision resistance: Any pair sharing *any* birthday?
    - Birthday paradox ⊠

The importance of weak collision resistance is best understood in the context of digital signatures, which are used to create legally binding digital contracts proving who signed what. Without going into details of digital signatures right now, it is sufficient to know that the contract's text is a message M and that digital signatures on M are created from the cryptographic hash value H(M).

Suppose Alice and Mallory agree that Mallory buys Alice's used car for 20,000€. Both digitally sign the contract's message M. However, Mallory changes his mind and does not want to buy the car any longer.

If hash function H is not weakly collision resistant, Mallory may be able to create a second contract M' which includes the price of 1€ for Alice's car such that H(M) = H(M'). In this situation, as digital signatures are derived from hash values, the digital signatures of Alice and Mallory created for M are also valid for M'. Thus, Alice has no proof that Mallory signed M in the first place.

So: If a message M is given, nobody should be able to create a second message M' with the same hash value under weak collision resistance.

For strong collision resistance, nobody should be able to create any collision at all, even if those collisions only occur for messages that look like gibberish without practical value.

A different angle on collision resistance is provided by the following birthday analogy. Consider the hash function mapping each person to his or her month and day of birth. Essentially, there are 366 different hash values (including February 29), and a collision occurs when two people share the same birthday.

Suppose you are in class. When you wonder whether some of your fellow students shares *your* birthday, you consider weak collision resistance. In contrast, when you ask whether *any pair* of students shares the same birthday, you consider strong collision resistance.

For simplicity, ignore leap years and consider just 365 different birthdays, all with the same probability. I'm confident that for a class of 30 students you can compute the probabilities of (a) somebody sharing your birthday as well as (b) any

pair sharing a common birthday. If you do the math for the first time, you may be surprised by the high probability in case (b), which is known as the birthday paradox (whose essence is the fact that the number of pairs grows quadratically, about which you can read more at Wikipedia). As the probability of case (b) is larger than that of case (a), it is harder to defend against case (b). Thus, hash functions targeting strong collision resistance must be "stronger" than those offering weak collision resistance.

2.3.2. Sample Hash Applications

- Avoidance of plain text passwords
- Integrity tests
- Digital signatures

2.3.3. Sample Hash Standards

- MD4, MD5, SHA-1: Broken
- SHA-1, SHA-2: Designed by NSA
  - Bruce Schneier, 2004: "Algorithms from the NSA are considered a sort of alien technology: They come from a superior race with no explanations 🚀 "
  - Cryptographic hashing is extremely difficult, experts in 2006:
    - "Joux says that we do not understand what we are doing and that we do not really know what we want; there is agreement from all the panelists. 🚀 "
  - 2017: SHA-1 broken (deprecated by NIST in 2011)
    - Attack called **SHAttered**, see https://shattered.io/ 🚀 if you are interested
  - SHA-3 🚀 (aka Keccak), standard since 2015
    - Winner of **public competition** from 2007 to 2012

2.3.4. On Resource Limitations

- Previous slide mentions SHAttered 🚀
  - In 2017, researchers demonstrated that SHA-1 is broken
- Sample results
  - Brute force attack would have taken about **12,000,000 GPU years**
  - Researchers exploited weaknesses in SHA-1
    - Their attack took **6,500 years** of single-CPU computations and **110 years** of single-GPU computations
    - **Computationally feasible** on Google cloud infrastructure as of 2017

2.3.5. Sample Message and Fingerprints

```
Hi Bob,

let's get started tomorrow!

Best wishes
Alice
```

- Sample hash values with GnuPG
  - `gpg --print-md SHA1 alice.txt`
    - alice.txt: 6FC1 F66C 598B D776 BA37 1A5C 2605 06CB 4CF9 0B89
  - `gpg --print-md SHA256 alice.txt`
    - alice.txt: 84E500CB 388EE799 05F50557 43C5481B 08B0BF17 1A2AE843 F4A197AD 2BA68D2E

- (Besides, specialized hashing tools exist, e.g., `sha256sum`)

2.4. Quiz

Cryptographic hashing is a wonder.

1. Select correct statements.

☐ Cryptographic hash functions are a subclass of hash functions that aim for collision resistance.

☐ Hash values can be computed by anyone (including attackers) as hash functions are publicly standardized.

☐ Strong collision resistance implies weak collision resistance, i.e., every hash function that is strongly collision resistant is also weakly collision resistant.

☐ Hash functions that are not strongly collision resistant are useless for digital signatures.

2.5. Symmetric Encryption

- Sender and recipient share secret key, K_{AB}
- **Encryption**, by function E, of **plaintext** message M with K_{AB} into **ciphertext** C
  - C = E(K_{AB}, M) (abbreviated to K_{AB} (M) for agreed E)
    - Bits of M and K_{AB} are mixed and shuffled using reversible functions (e.g., XOR, bit shift)
    - Simplest, yet provably secure case: One-time pad 🚀 with XOR of **random** bit string and M

- **Decryption**, by function D, with same key K_{AB}
  - M = D(K_{AB}, E(K_{AB}, M))
  - Notice: Need to exchange secret key ahead of time
- Typical symmetric algorithms: AES 🚀 , 3DES 🚀

2.6. Intuition of Asymmetric Encryption

- Participants own **key pairs**
    - Private key, e.g., K_{B-}: **secret**
    - Public key, e.g., K_{B+}: **public** / **published**
    - En- and decryption based on "hard" mathematical problems
- Think of key pair as **safe/vault** with numeric **key pad**
    - Open safe = public key
        - Everybody can deposit messages and lock the safe
    - Opening combination = private key
        - Only the owner can open the safe and retrieve messages

While symmetric encryption with shared keys, in particular the one-time pad, may seem intuitively clear, asymmetric cryptography requires some thought. Every participant needs a key pair, which consists of a private key and a public key. As the names suggest, a private key needs to be kept secret and must only be accessible by its owner, whereas the public key can be published, e.g., on web servers or special key servers.

This slide offers an analogy of public key cryptography with physical safes, which might help to convey essential ideas: The public key of Alice is used by others to encrypt messages to her, while she uses her private key to decrypt them. Similarly, she might offer opened safes in the real world, into which messages can be placed and which can be locked by everyone. Only Alice is able to open the safe using its opening combination to retrieve and read contained messages. Thus, the opening combination corresponds to her private key.

A noteworthy challenge of asymmetric cryptography, which is mentioned on the next slide, is the reliable distribution of public keys: How does Bob know that he really obtained Alice's public key and not one created by Mallory and distributed in her name? Or in the above analogy: How does he make sure that he places his messages into Alice's safe and not into one owned by Mallory to which Mallory attached the name tag "Alice"? Answers to this question are provided under the term "public key infrastructure", and they frequently rely on the idea that Bob needs to verify a fingerprint of Alice's public key through an out-of-band communication channel. This highly relevant, fascinating, and challenging topic is beyond the scope of this presentation, though.

2.7. Asymmetric Encryption

- Participants own **key pairs**
  - Private key, e.g., K_{B-}: **secret**
  - Public key, e.g., K_{B+}: **public** / **published**
- **Encryption** of message for Bob with Bob's **public key**
  - $C = E(K_{B+}, M) = K_{B+} (M)$
  - Notice: No secret key exchange necessary
- **Decryption** with Bob's **secret key**
  - $D(K_{B-}, K_{B+}(M)) = K_{B-}(C) = M$
  - Notice: Only Bob can do this
- Challenge: Reliable distribution of public keys
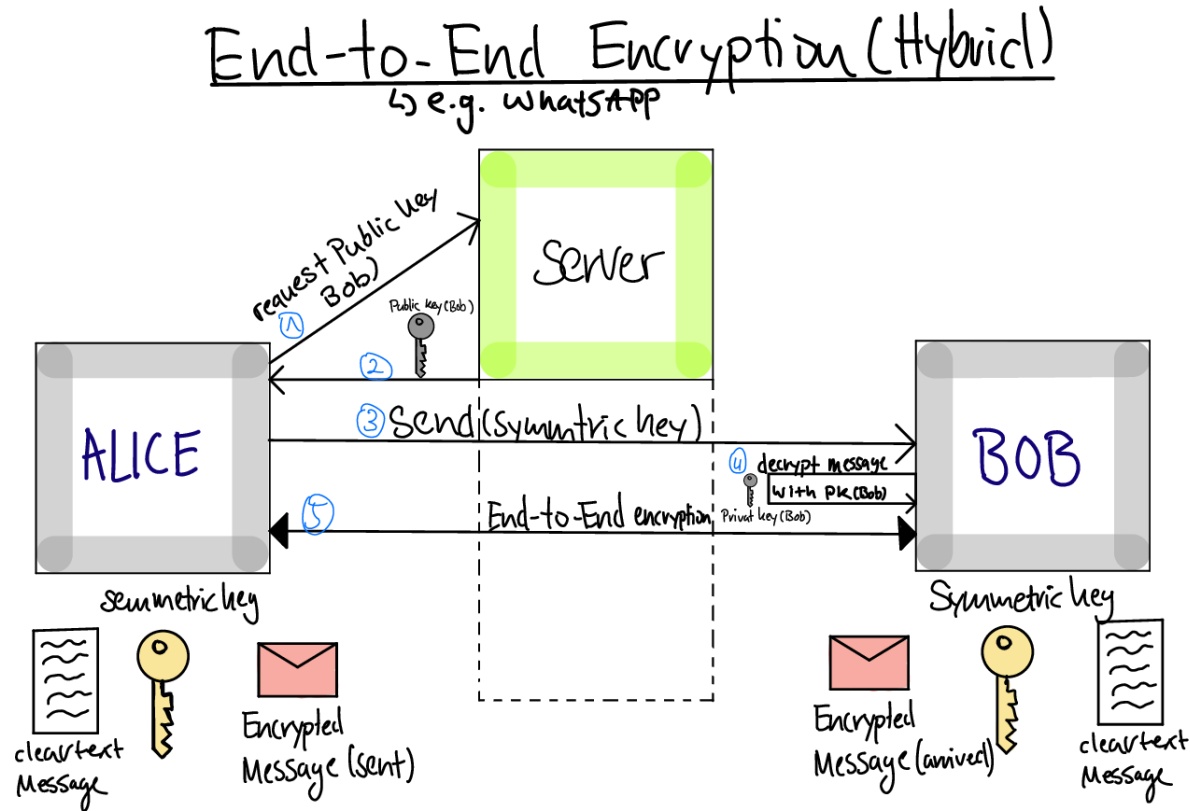 - Solution: Certificates in Public Key Infrastructure 🚀 , PKI

2.7.1. Sample Asymmetric Algorithms

- Diffie-Hellman Key Exchange 🚀 (1976)
  - Used, e.g., in IPsec, SSL/TLS, Tor 🚀, OTR 🚀
    - RFC 7568 🚀, June 2015: SSLv3 MUST NOT be used
- RSA 🚀 (Rivest, Shamir, Adleman 1978)
  - Turing award 🚀 2002, most famous, PGP, GnuPG
- ElGamal 🚀 (1985)
  - Based on Diffie-Hellman, GnuPG and newer PGP variants
- Others more recently, e.g.:
  - Elliptic curves 🚀 with shorter keys, also GnuPG
  - Post-quantum cryptography 🚀
    - 4 standards 🚀 in 2022 based on NIST competition 🚀 since 2016
      - One of them, SIKE, found to be insecure in August 2022

2.7.2. Hybrid End-to-End Encryption

End-to-End Encryption (Hybrid)
↳ e.g. WhatsApp

Server

request Public key (Bob) ①

② Public key (Bob)

ALICE

③ Send (symmetric key)

⑤ End-to-End encryption

semmetric key

cleartext Message

Encrypted Message (sent)

④ decrypt message with Pk (Bob)
Privat key (Bob)

BOB

Symmetric key

Encrypted Message (arrived)

cleartext Message

This figure was created by students in the context of the course Communication and Collaboration Systems (CACS) in 2020.

In the simplistic and non-realistic protocol shown here, Alice obtains Bob's public key from some server. This server could be an ordinary web server (e.g., Bob's homepage), a separate key server, or a server used in the background of her application, e.g., in case of Signal or WhatsApp. Then, she creates a symmetric encryption key, encrypts this with Bob's public key and sends the result to Bob. Bob uses his private key to decrypt the message and obtains the symmetric key of Alice. Now, Alice and Bob share a symmetric key that can be used to encrypt subsequent communication.

To appreciate what might be involved in real protocols, first note that only Bob's private key is used here. So, Mallory could pretend to be Alice and send a symmetric key in her name. Second, for messaging we are usually interested in a property called forward secrecy 🚀. Briefly, this means that even if long-term asymmetric keys were broken, attackers should still need to break the encryption of individual messages. Clearly, the shown protocol does **not** satisfy this property: Suppose attacker Eve records years of communication and later somehow obtains Bob's private key. Then, Eve can first decrypt the messages containing the symmetric keys and afterwards everything else.

To guarantee forward secrecy, protocols usually involve some form of Diffie-Hellman key exchanges to set up shared symmetric keys, which are changed frequently. A famous example used for messaging is the double ratched algorithm 🚀 which is part of the Signal protocol 🚀 .

2.7.3. GnuPG: Hybrid Encryption

- Create asymmetric key pair
  - `gpg --gen-key`
  - Various options/alternatives
- Encryption for Bob
  - `gpg -e -a -r Bob file`
    - Creates `file.asc`; more precisely:
    - Creates random secret key K_{AB}
    - Symmetric encryption of file with K_{AB}
      - Specific algorithm obtained from Bob's public key
    - Asymmetric encryption of K_{AB} with K_{B+}
      - Beware! No naïve encryption, but, e.g., PKCS 🚀 #1
    - Result: $K_{B+}(K_{AB}) + K_{AB}(file)$
      - ("+" between ciphertexts denotes string concatenation)

3. Message Integrity

3.1. Situation and Goal

- Alice sends message M to Bob
  - (Parts of) Network controlled by unknown parties (Eve and Mallory)
- Goals of integrity
  - Bob is sure that **M came from Alice**
    - Notice: Need authentication (proof of identity ▶)!
  - Bob can **detect modifications to M**
- Non-goals: Alice cannot be sure
  - that no third party receives M
  - that Bob receives M
  - that Bob receives M in unchanged form

3.2. General Idea

- Alice sends message along with its fingerprint
  - ◄ Hint: A hash value is not good enough
  - Instead: Use some ingredient that is **unknown to the attacker**
- Bob receives message and fingerprint and verifies whether both match
  - If message changed by Mallory, he cannot produce a matching fingerprint
- Typical techniques
  - Message authentication codes
    - E.g., Alice and Bob share secret K_{AB}, concatenate that to message before hashing
  - Digital signatures (next slides)

3.3. Digital Signatures

- Based on asymmetric cryptography
  - En- and decryption **reversed**
- Basic idea
  - **Signature** created by encryption with **private** key: $K_{A-}(M)$
    - Only Alice can create this!
  - **Verification** via decryption with **public** key: $D(K_{A+}, K_{A-}(M))$
    - Everyone can do this as public key is public!
- Practice: Encrypt hash value of M, e.g., $K_{A-}(SHA\text{-}3(M))$
  - Recall
    - Performance
   - Hash collisions

3.3.1. Some Details of Digital Signatures (1/2)

- **Signing** of M by Alice with private key K_{A-}
  - Signature $S = K_{A-}(h(M))$
    - Only Alice can do this
  - Transmit signed message $[M]_{K_{A-}} = M + S = $ message + signature
    - ("+" is concatenation)
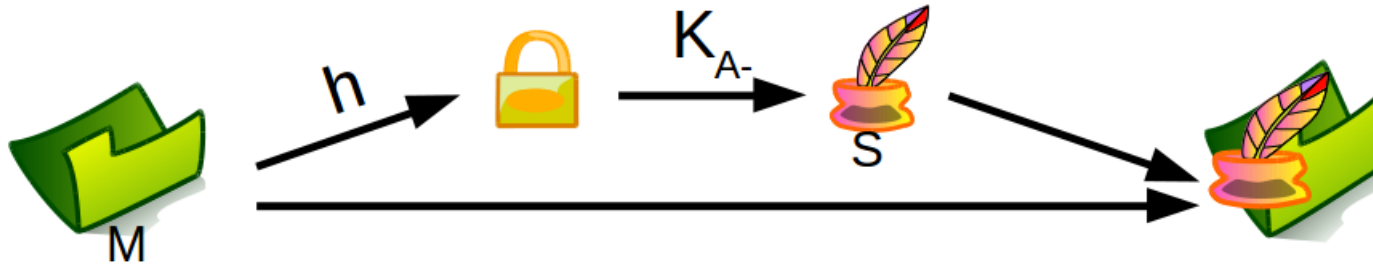
3.3.2. Some Details of Digital Signatures (2/2)

- [M] **received** by Bob

- **Verification** whether [M] sent by Alice and unchanged along the way

  - Split [M]: [M] = M' + S'
  - Hash M': H = h(M')
  - Decrypt S': H' = K_{A+}(S')
    - Bob needs public key of Alice to do this
    - Everyone can do this
  - Verify H = H'
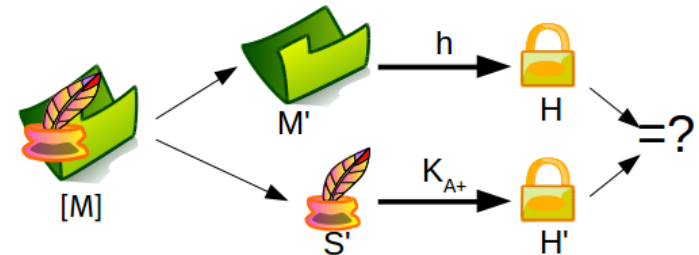
3.3.3. GnuPG: Digital Signatures

- `gpg --sign -a -b alice.txt`
  - Creates digital signature `alice.txt.asc` for input `alice.txt`

- `gpg --verify alice.txt.asc`
  - Expects to be verified content as `alice.txt`
  - Verifies signature
  - Frequently used to verify integrity of downloads

3.4. Electronic Signatures

- "Signatures" of varying legal impact in IT environments
  - Different types, e.g., simple (e.g., sender's name in e-mail), advanced (digital signature as discussed above), qualified
  - Qualified electronic signatures may replace paper based signatures (e.g., dismissal, invoice)
    - Subset of advanced electronic signatures
    - Based on qualified certificates (with qualified electronic signature, issued by accredited organization; law prescribes rules concerning infrastructure and processes)
    - Created on secure signature-creation devices (nPA ⊠ may store qualified certificate; additional reader necessary)

4. OS Context

4.1. Basic OS Security Services

4.1.1. Service Overview (1/2)

- Rights management, **authorization**
  - Discussed already: Access rights⬈
    - What is Bob allowed to do?

- **Logging**
  - Who did what when?
  - (Not considered here)

- Basic **cryptographic** services
  - Offering selection of above techniques: a/symmetric techniques, hashing

4.1.2. Service Overview (2/2)

- **Identification/Authentication**
  - Identification: Claim of identity
    - I'm Bob …
  - Authentication: Proof of identity (more on subsequent slides)
    - My password is "p@ssw0rd"
      - (Bad idea, easily broken!)

- **Integrity protection**
  - E.g., installation and updates of software under GNU/Linux with apt 🚀

4.1.3. Authentication

- Proof of identity
  - Something the individual knows
    - Password, PIN, answer to security question
  - Something the individual possesses
    - Private key (on smartcard or elsewhere), iTAN
  - Something the individual is
    - Static biometrics, e.g., fingerprint, iris scan
  - Something the individual does
    - Dynamic biometrics, e.g., voice or typing pattern

- Necessary prerequisite to enforce access rights⇗
  - Who is allowed to perform what operation on what resource?

4.1.4. Two-Factor Authentication

- Combinations of above categories
  - Physical banking
    - Bank card (possession) plus PIN (knowledge)
  - Online banking
    - Password for login (knowledge) plus mTAN or iTAN (possession)
  - Beware: Must keep factors separate
    - Do not record PIN on card
    - Do not perform online banking on device that receives mTAN

4.2. Key Security Best Practices

- Consult others
- Adopt a holistic risk-management perspective
- Deploy firewalls and make sure they are correctly configured
- Deploy anti-virus software
- Keep all your software up to date
- Deploy an IDS
- Assume all network communications are vulnerable
- … (see Sec. 11.8 in [Hai19])

5. Conclusions

5.1. Summary

- Security is complex, requires design and management
- Cryptography provides foundation for lots of security mechanisms
  - Don't implement cryptographic protocols yourselves!
  - Use proven tools, e.g., GnuPG
- Asymmetric crypto with key pairs
  - Public key for encryption and signature verification
  - Private key for decryption and signature creation
- Hash functions and digital signatures for integrity

Bibliography

[Hai19] Hailperin, Operating Systems and Middleware – Supporting Controlled Interaction, revised edition 1.3.1, 2019. https://gustavus.edu/mcs/max/os-book/ ⬀

[KPS02] Kaufman, Perlman & Speciner, Network Security: Private Communication in a Public World, Second Edition, Prentice Hall Press, 2002.

[Lec11] Lechtenbörger, Zur Sicherheit von De-Mail, Datenschutz und Datensicherheit 35(4), 268-269 (2011).

[SS75] Saltzer & Schroeder, The protection of information in computer systems, Proceedings of the IEEE 63(9), 1278-1308 (1975).
http://web.mit.edu/Saltzer/www/publications/protection/ ⬀

License Information

This document is part of an Open Educational Resource (OER) course on Operating Systems. Source code and source files are available on GitLab under free licenses.

Except where otherwise noted, the work "OS11: Security", © 2017-2023 Jens Lechtenbörger, is published under the Creative Commons license CC BY-SA 4.0.

> *No warranties are given. The license may not give you all of the permissions necessary for your intended use.*

In particular, trademark rights are *not* licensed under this license. Thus, rights concerning third party logos (e.g., on the title slide) and other (trade-) marks (e.g., "Creative Commons" itself) remain with their respective holders.