Information Security

1 Access control

1.1 Access control

- Access control concepts
  - **Subjects** request **actions** on objects
- Access control (AC)
  - Authentication = verifying subject identity(implies identification of the subject)
  - Authorization = checking that the subject has the right to request the action on the subject
- Reference monitor
  - Reference monitor controls access by subjects to objects
    - Grants i.e. allows or denies access requests
    - Follows the policy (i.e. rules) set by administrators
    - Logs events to audit trail
- Access control matrix
  - Access control matrix is the simplest, most general AC model

$$M : Subjects \times Objects \rightarrow P(Actions)$$

  - AC matrix represents the protection state of a system

	file1.txt	file2.txt	file3.txt
Alice	read, write	write	-
Bob	read	read	-
Carol	read, write	-	-
David	append		open, read, write, close

- Access control list (ACL)
  - ACL = list of the access rights associated with an object

    - ACLs are a practical way to represent the AC matrix: one column of the matrix is stored for each object
- Capabilities
    - Capability = an access right associated with the subject
    - Capabilities are another way to represent the AC matrix: one row is stored for each subject
- Principal
    - Subjects are often ephemeral, e.g. processes; something more persistent is needed to define the access policy
    - Solution: access rights are assigned to principals
    - Principal is an authenticated identity, e.g. user account
    - Subjects act on behalf of principals
        - Process (subject) runs as as a specific user account (principal)
        - User's process running as admin or as normal user act on behalf of different principals and, thus, have different access rights

1.2 Discretionary AC

- Discretionary access control (DAC)
    - Data owner, usually a user, sets the access rights
    - Subjects can share their access rights to others
        - File owner or creator decides who can access it
        - User or process that can read a secret file can also copy and share it
    - Typical in commercial and consumer systems
    - Commonly implemented with ACLs or capabilities

1.3 Mandatory AC

- Mandatory access control (MAC)
    - Access control is based on rules (policy) set by administrator
    - AC policy is enforced by the reference monitor and cannot be changed by users
    - Subjects cannot leak access rights to others, e.g.
        - User can read a secret file but cannot copy, print or email it
        - Process can download data from the Internet or write to the file system, but not both → users cannot download malware
    - MAC is also called rule-based AC

- Clearance and classification
  - Mandatory access control policy is often based on security labels
    - Subject clearance
    - Object classification

$$l : (Subjects \cup Objects) \rightarrow Labels$$

  - Simple security property: S can read O iff *l*(S) ≥ *l*(O)
  - MAC based on clearance and classification levels is called multi-level security (MLS)
- Bell-LaPadula model
  - Bell-LaPadula (BLP) is a MAC policy for protecting secret
    - Military security model for multi-user computers that process secrets
  - Bell-LaPadula rules
    - S can read O iff *l*(S) ≥ *l*(O) simple security property
    - S can write O iff *l*(O) ≥ *l*(S) *-property
  - Also called: no read up, no write down
- Biba model
  - In computer systems, integrity of data and the system is often more important than confidentiality
  - Biba is a MAC policy for protecting integrity of data
  - Biba rules:
    - S can write O iff *l*(S) ≥ *l*(O)
    - S can read O iff *l*(O) ≥ *l*(S)
  - Also called: no write up, no read down
- Upgrading and downgrading
  - In practice, security levels need to be changed by humans
    - E.g. downgrading intelligence data to enable broader access
    - E.g. upgrading intelligence reports based on open data
  - Declassification = downgrading to unclassified level
- Data sanitization
  - Documents may need to be sanitized i.e. redacted before downgrading or declassification
    - E.g. removing names of personnel before publication
  - Sanitization of electronic documents is technically difficult
    - Painting black box over text in PDF is not enough
  - High subjects can leak data intentionally via **covert channels**
    - E.g. character spacing on printed documents; time between network messages; variation in processor load

1.4 Role-based AC

- Groups and roles
  - Group = set of subjects
    - E.g. Administrators, CS-C3130-students
    - Object's ACL can list groups in addition to individual users
    - Both group membership and ACLs change over time
  - Role = set of permissions (=set of actions on objects)
    - E.g. Administrator, CS-C3130-teacher, SCI-professor
    - Roles are usually static; only assignment to users
- Role-based access control (RBAC)
  - Roles may form a hierarchy with rights inheritance
    - e.g. Lecturer and Teaching-assistant are Teaching-staff
  - Constraints on role assignment and simultaneous
    - activation can implement separation of duty(defined later)
  - Roles are assigned to users for longer term but activated on demand for each session

1.5 Other AC models and variations

- Attribute-based access control (ABAC)
  - Access control is based in subject attributes (properties) instead of subject identity
    - ABAC = attribute verification + authorization
    - Separation of duty
    - Separation-of-duty policy: two or more persons are required to complete a task
    - Common in business and public administration:
      - Employees cannot approve their own expense claims
      - Financial auditors of large companies must be from outside the company
      - Safe may have two locks with keys held by two different persons
      - Lecturers issue grades to students, but only admin staff can enter them into the study register

2 User Authentifcation

2.1 Conception

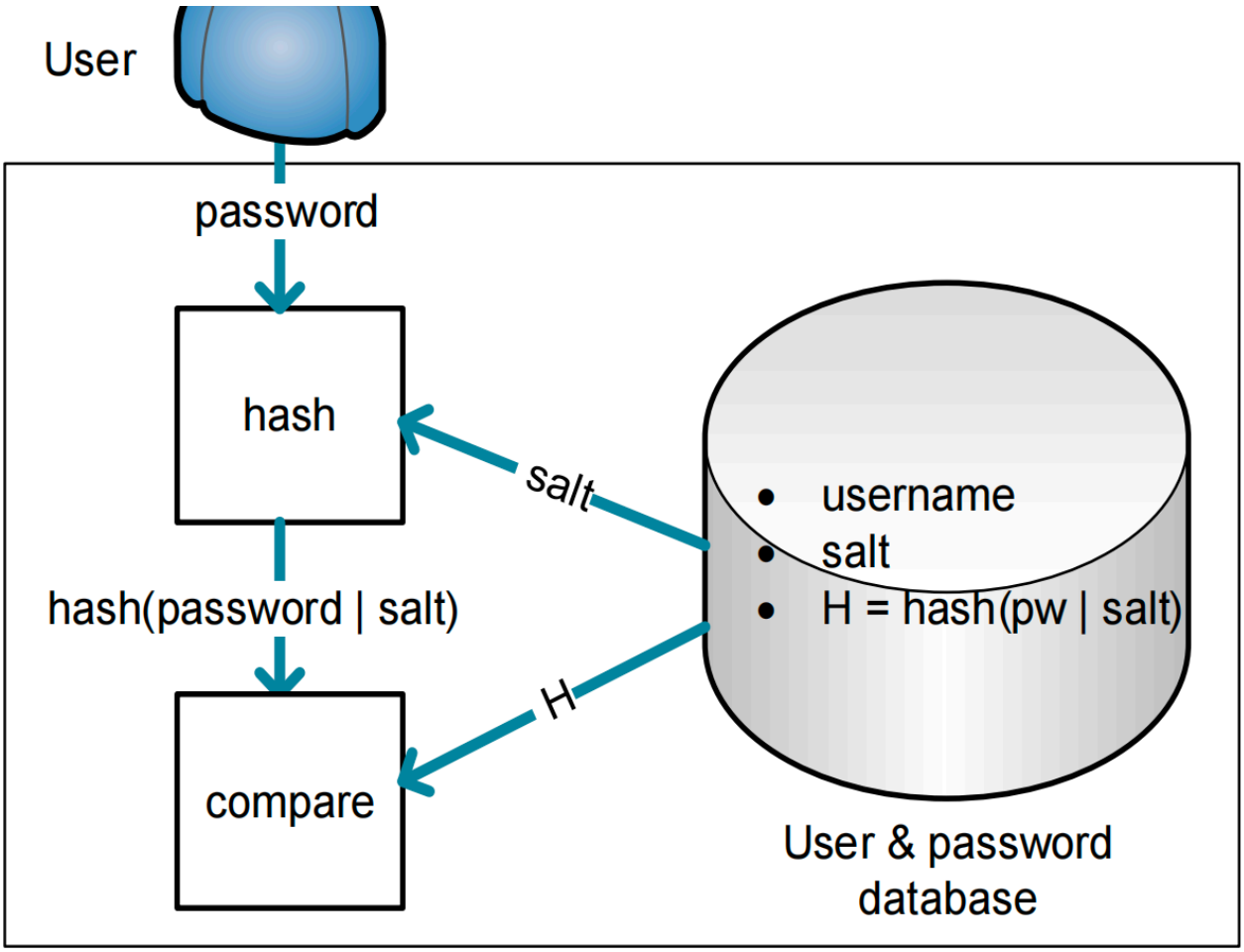
- User authentication
  - Verifying user identity
  - Needed for **access control and auditing**

    > access control = authentication + authorization

  - User authentication is based on **credentials**
    - Credentials: password, key, smart card etc.
- Password
  - Password is a shared secret between the user and computer system
  - Limitations arise from the reliance on human memory and input methods, and from the lack of cryptographic computing capability in humans

2.2 Password Storage On Server

- Storing passwords on server
  - Assume that your password database is public
    - Unix /etc/passwd is traditionally world readable
    - Attackers often read server files or database tables (e.g. with SQL injection)
  - methods
    - Store a **one-way hash** value of the password
    - When user enters a password, compute its hash and compare
    - Use a **slow hash function**,e.g. PBKDF2, Argon2
    - Include salt: a user specific random string. not secret (readable to adversaries)



- Slow hash function
  - Standards hash functions are unnecessarily fast!
  - Iterative hash:
    - hash(pw|hash(pw|salt)) takes twice as long as hash(pw|salt)
    - Iterate N times (N > 100 000) for desired delay
  - Not a significant cost when verifying user login, but increases a brute-force attacker's work by factor N
  - Slow functions designed specifically for password hashing: PBKDF2, Argon2

2.3 Password Cracking

- Offline cracking
  - Attacker obtains the password hashes or other data for verifying password guesses, then starts guessing
  - Brute-force attacks vs. intelligent dictionary attacks
    - Most password crackers combine both strategies
  - Attacker has great advantages:
    - Unlimited number of guesses

        - Can rent elastic computing capacity for quick results
    - To resist cracking, passwords must have cryptographic strength (~128 bits of entropy)
- Online trials
    - attacker tries to login many times
        - Try PIN codes on a phone or cash machine
        - Guess passwords for a web site
        - Port scan ssh servers and guess root password
    - System can limit the number or rate of login attempts
        - Possible in online services, smartcards, phone, Microsoft account
        - Huge improvement in security: success probability ≈ number of allowed guesses / number of possible passwords
        - Denial of service (DoS) is a danger, e.g. bricking a phone; use delay rather than a fixed limit on the number of trials when possible
- Cost of offline password cracking
    - Time to crack a random 10-character (printable ASCII) password from its SHA-256 hash?
        - High-end multi-core CPU on a PC computes up to 500 MH/s
        - High-end graphics card computes up to 7 GH/s, same cost
        - Bitcoin miner computes 15 TH/s
    - Always measure cracking cost in money, not in time, because brute-force cracking parallelizes easily and computing capacity can be rented on demand
        - One CPU or GPU day ≈ $1 (cloud CPUs may be cheaper)
    - Time can be shortened by parallelizing; cost remains the same!

> Converting between bases 2 and 10:
>
> - kilo k = 2^10 ≈ 10^3
> - mega M = 2^20 ≈ 10^6
> - giga G = 2^30 ≈ 10^9
> - tera T = 2^40 ≈ 10^12
>
> Conversion examples:
>
> - 300M ≈ 300 · 106 (< 256 · 220 = 228, > 128 · 220 = 227)
> - 2 ^ 34 = 2^4 · 2^30 = 16G ≈ 16 · 10 ^ 9

2.4 Entrophy And Password Strength

- Measuring password strength
  - Many possible metrics:
    - Number of possible passwords
    - Entropy = amount of missing information
    - Average/median cost to crack a specific password / any one password
    - Success probability / number of cracked passwords as function of cost
  - Metrics are useful for system designers and setting policies
  - Measuring strength of user-chosen passwords is impossible
- Password entropy
  - Entropy = the amount of missing information

$$Entropy\, H = -\sum_{x \in \text{passwords}} (P(x) \cdot \log_2 P(x)) \leq \log_2(\text{number of possible passwords})$$

  - With even probability distribution:
    - H = log2(number of possible passwords)
    - Example: random 8-character alphanumeric passwords:

$$\mathsf{H} = \log_2(62^8) = 8 \cdot \log_2(62) = 47.6\mathsf{bits}$$

  - One-bit increase in entropy approximately halves the success probability or doubles the cost of guessing attacks(exactly so with even probability distribution)
- Sufficient PIN and password entropy
  - What is sufficient entropy to resist online guessing?
    1. Determine the maximum number of guesses, e.g. K = 3
    2. Decide acceptable success probability, e.g. P = 10-6
    3. Required entropy H = log2(K/P) = 21.5 bits
  - What is sufficient entropy to resist offline cracking?
    1. Estimate maximum hash rate, e.g. Bitcoin network R = 1.2·1020 H/s (SHA-256) in 2020
    2. Decide how long the attack could take, e.g. T = 1 year = 31.5·106 s
    3. Decide acceptable success probability, e.g. P = 10-6
    4. Required entropy H = log2(R·T/P) = 66.7+24.9+20 bits = 111.6 bits
       - Human effort can crack 92-bit passwords and threaten 112-bit ones.

    > Traditionally, 128 bits has been considered cryptographically strong

- Human-selected passwords have less entrophy

2.5 Other Password Security Issues

- Password sniffing on the local network is prevented by cryptographic authentication (SSH, HTTPS, MS-CHAPv2,...)
- Key logger: software or hardware that stores all keystrokes typed on the computer
    - Problem in public-access computers
    - Malware can sniff passwords on any infected computer
- Trusted path
    - Trusted path is any mechanism that ensures direct and secure communication between user and a trusted part of the system
        - **Crtl+Alt+Del in Windows (secure attention key / sequence)**
            - with this sequence of key presses, user can be sure that the communication does not go to an arbitrary piece of software running on the computer, but it goes directly to the windows operating system
        - Reset button in all kinds of devices
        - **Web browser address bar**
    - With malware, virtualization and full-screen apps, it is increasingly hard to know what is real
- Password reuse
    - Same or related passwords on multiple accounts
        - compromise of one system or account leads to compromise of the user's other accounts
    - Solutions:
        - Password manager that stores and generates random passwords
        - Single sign-on (SSO)
            - Shibboleth SSO to university web pages
            - Microsoft AD, IBM Tivoli Access Manager, etc.
            - Facebook, Google, etc. login on many websites
- Password recovery
    - Humans are prone to forget things → need a process for recovering from password loss
        - Failure-recovery often enables new attacks!
        - This applies to security mechanisms in general
    - Some password recovery methods:
        - Physical visit to helpdesk
        - Security question or memorable secret, e.g. mother's maiden name, birthdate
        - Email or text message with authorization code or link
        - Paper notebook, sticky note under the keyboard
        - USB memory stick with a password recovery file

        - Print recovery code as QR code

2.6 Better User Authentication

- One-time passwords
    - Use each password only once. Protects against password sniffers and key loggers
        - Random one-time passwords
        - Lamport hash chain
        - Unix S/KEY or OTP
          1: HOLM BONG VARY TIP JUT ROSY
          2: LAIR MEMO BERG DARN ROWE RIG
          3: FLEA BOP HAUL CLAD DARK ITS
          4: MITT HUM FADE CREW SLOG HAST
        - Many commercial products such as RSA SecurID
        - Code apps and devices for Finnish banks
    - prevent: password reuse and logging
    - not prevent: man-in-the-middle attack where the attackers captures the password in real time
- Weak and low-entropy credentials
    - PIN, graphical passwords, face recognition, fingerprints have recently replaced strong passwords. Why would that be ok?
        - Only for **physical access to device**, not for remote access to the device or to related online services
        - For access to online services, physical possession of the user device is considered one authentication factor, PIN the second authentication factor
    - Main threat now is **lost and stolen mobile devices**
        - Attacker does not know the user
        - Hardware feature to lock the device after a few trials
- Online accounts
    - User authentication delegated to online server
        - Device cryptographically locked, and server releases keys after successful authentication
        - Online server can limit the number of password guesses and implement risk-based additional authentication, e.g. 2FA
        - Device must not store the password database and must be online
    - But are the password hashes cached locally?
        - e.g. Windows login with Microsoft account caches authentication information locally, unless disables by domain administator
    - Authentication delegated to a secure hardware module can have similar benefits

- Physical security tokens
  - Smart card is a typical physical security token
    - Stores cryptographic keys to prove its identity
    - Tamperproof: secret keys will stay inside
  - Used for door keys, computer login, bank cards
  - Other security tokens: smart button, USB dongle, trusted chip in mobile phone
- Two-factor authentication (2FA)
  - Two-factor authentication = require both a physical token and a PIN or password
  - Attacker needs to both steal the physical device and learn the PIN
    - clear qualitative increase in security
  - Context-aware or risk-based authentication:
    - Require additional authentication only when the user is suspicious or requested action requires stronger security
    - Online services can do this intelligently to avoid annoying the user

3 Software Security

3.1 Untrusted input

- Untrusted input
  - User and network input is untrusted
  - Does my software get input from the Internet?
    - Documents, streams, messages, photos may all be untrusted
    - All modern applications are Internet clients or servers
    - Intranet hosts, backend databases, office appliances etc. are directly or indirectly exposed to input from the Internet

> All software must be able to handle malformed and malicious input safely

3.2 SQL injection

- SQL injection example
  - SQL query:
    - `SELECT * FROM users WHERE username = 'Alice';`
  - Code with embedded SQL:

        - `"SELECT * FROM users WHERE username = '" + input + "';"`
    - Attacker sends input:
        - `input = "Bob'; DROP TABLE users; --"`
    - The query evaluated by the SQL database:
        - `SELECT * FROM users WHERE username = 'Bob'; DROP TABLE users; --';`
- Mitigating SQL injection
    - Minimum privilege: set tables as read-only; run different services as different users
    - Sanitize input: allow only the necessary characters and string formats
        - but it is hard to do correctly!
    - Escape input strings with safe library functions, e.g.
        - `mysql_real_escape_string()` in PHP
        - `MySQLdb.escape_string()`, `MySQLdb.execute()`, `sqlalchemy.text()` in Python
    - Prepared statements and stored procedures:
        - precompiled SQL queries that can be executed many times with different parameter values
    - Disable SQL error messages to normal users
        - harder to build exploits

3.3 Buffer overrun

- Buffer overrun
    - Bug: failure to check for array boundary
- Buffer overruns may cause
    - data modification → unstable program behavior
    - access violation “segmentation fault” → process crashing
    - malicious data modification
    - code injection → attacker gains full control of the process
- Running exploit code
    - How attacker gains control:
        - Stack overruns may overwrite function return address or exception handler address
        - Heap overruns may overwrite function pointer or virtual method table
    - How difficult is writing an exploit?
        - Instructions and code widely available
        - There are people and companies actively developing exploits
- Buffer overrun prevention
    - Preventing buffer overruns:

        - Type-safe languages (e.g. Java, C#)
        - Programmer training, code reviews
        - Avoiding unsafe and difficult-to-use libraries: strcpy, gets, scanf, MultiByteToWideChar, etc.
        - Fuzz testing
        - Static code analysis, symbolic model checking: proving safety
    - No reliable way to find all buffer overrun vulnerabilities → need to also mitigate consequences
- Buffer overrun mitigation
    - Stack canary
        - Store an unguessable value on the top of the stack frame in function prologue; check before returning
        - GCC -fstack-protector-all, Visual Studio /GS
    - Non-executable (NX) bit
        - Set the stack and heap memory pages as non-executable
        - Breaks some code, e.g. JIT compilation
        - Often combined with memory layout randomization
- Integer overflow
    - Integers in programming languages are not ideal mathematical integers
    - Integer overflow can cause buffer overrun

3.4 Web vulnerabilities: CSRF, XSS

- Cross-site request forgery (CSRF)
    - Interaction between web sites is usually prevented by the same origin policy. However, web links and HTTP GET and POST redirection to another site is allowed.
- Preventing CSRF
    - Server checks **Referer** (sic) field in HTTP requests
    - **CSRF token** i.e. secret session identifier in all GET URLs or POST payloads; attacker would need to guess it
- Cross-site scripting (XSS)
    - Stored XSS
        - User-posted content on web sites may contain malicious JavaScript
        - Fictional example:
            - Social.net allows users to post comments. Bob's comment: `<b onmouseover="$.get(' http://social.net/AddFriend.php?name=Bob');">Be my friend!<b>`

        - Another user reads the blog and moves mouse over the text
    - This is stored XSS: malicious script stored on the server
- Reflected XSS
    - PHP code on a web server:
      `<html><body><?php print "Page not found: ". urldecode($_SERVER["REQUEST_URI"]); ?></body></html>`
    - Typical output: Page not found: /foo.html
    - Bob tricks Alice to follow this URL-encoded link:
      `http://social.net/%3Cscript%3E%3D%22%24.get%28%27http%3A%2F%2Fsocial.net%2FAddFriend.php%3Fname%3DBob%E2%80%99%29%3B%3C%2Fscript%3E`
    - The error page on social.net will contain this:
      `Page not found: /<script>="$.get('http://social.net/AddFriend.php?name=Bob’); </script>`
    - This is reflected XSS: malicious script sent via the server but not stored there

3.5 Input validation

- File path validation
    - The same file path has many representations →use the realpath(3) function to obtain the canonical representation
    - Online services should be executed in a sandbox to limit their access: chroot(2), virtual machine or container

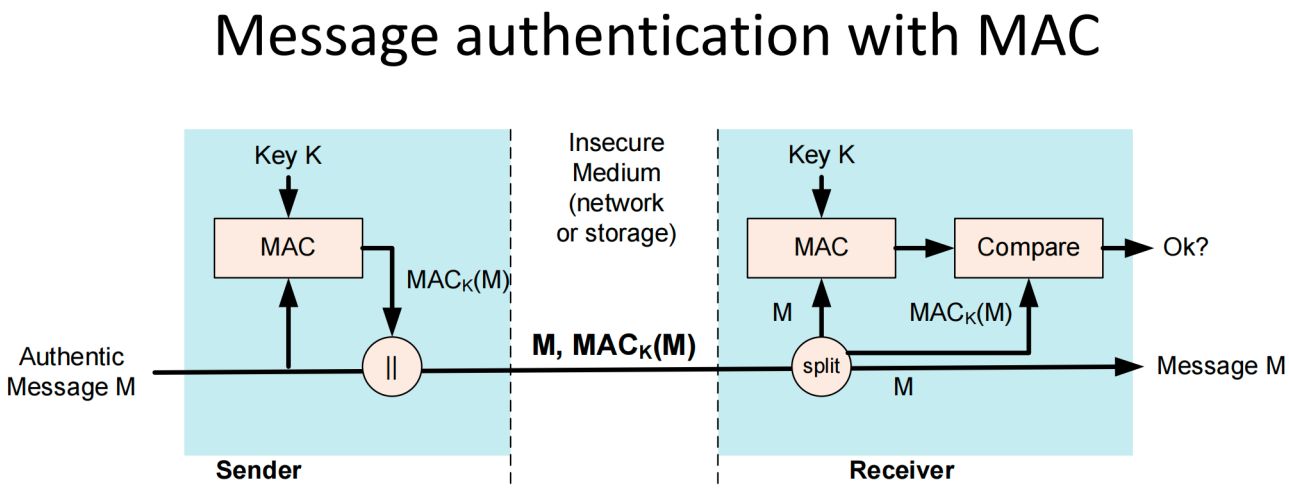
4 Cryptography

4.1 Cryptographic hash function and HMAC

- Cryptographic hash
    - = message digest = fingerprint
    - The algorithm is public, no keys or other secrets needed
    - Examples: SHA-256, SHA-512, SHA3-256
- Cryptographic hash: security requirements
    - One-way = pre-image resistant: given only output, impossible to compute input, except by guessing

- Second-pre-image resistant: given one input, impossible to find a second input that produces the same output
- Collision-resistant: impossible to find any two inputs with the same output
  - Old hash functions with broken collision resistance: MD5, SHA-1

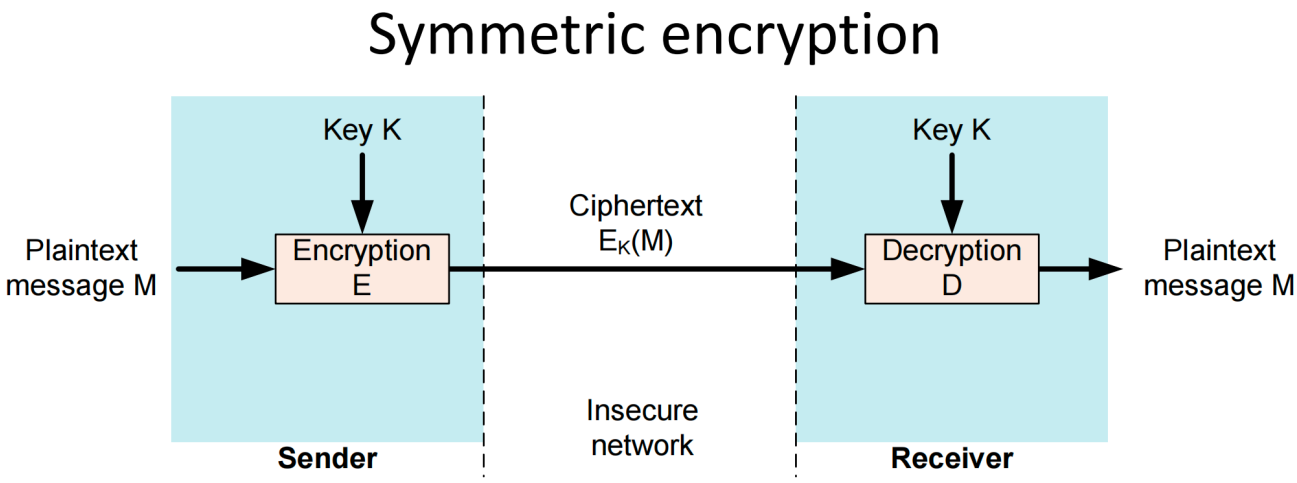
- Message authentication code (MAC)



  - Secret key is needed to create and to check the MAC
  - HMAC is a standard way to construct a MAC from a hash function, e.g. HMAC-SHA256
- Message authentication with MAC
  - Message authentication and integrity protection
  - Endpoints share the secret key K (thus, it is symmetric cryptography)
  - MAC is appended to the original message M

4.2 Symmetric encryption

- Symmetric encryption



  - Message encryption based on symmetric cryptography, i.e. a shared secret key

    - Kerckhoff's principle: the encryption and decryption algorithms are public algorithms; only the key is secret
    - Encrypted message content looks like random bits – unless you know the key
    - The key must be shared over a secure out-of-band channel
        - a 128...256-bit random number
        - sometimes computed from a passphrase with a cryptographic hash function (should use PBKDF2 to make cracking slower)
- Encryption and message integrity
    - Encryption alone protects secrets, not integrity
        - Attacker can usually modify the secret message
        - Receiver of the modified secret message usually leaks some information, e.g. error in message
    - Always combine encryption with integrity protection
        - Encrypt-then-MAC: encrypt with block cipher e.g. in CBC mode, then compute and append a MAC
        - Authenticated encryption modes do encryption and integrity in one pass, e.g. AES-GCM

4.3 Symmetric key and hash lengths

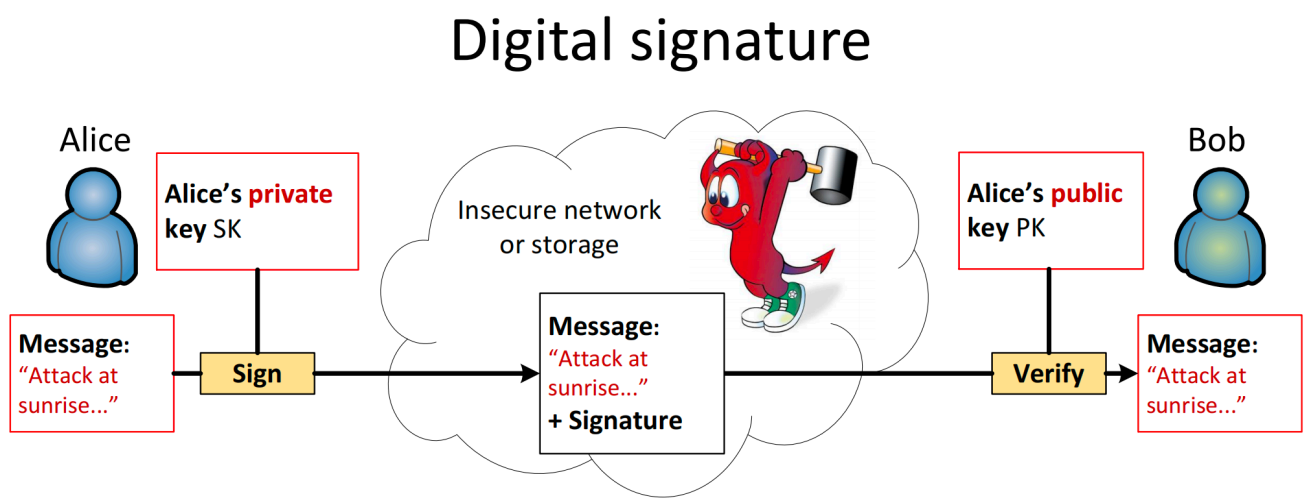
- Key length
    - Shared key of ≥ 128 bits is strong, < 80 bits is weak
        - To resist brute-force guessing, the secret key must be random with (almost) even probability distribution
        - Quantum cryptoanalysis may require keys of 256 bits in the future
    - Brute-force attacks are easy to parallelize; thus, cost should never be measured in time but in money (EUR, USD, CPU days)
        - 1 CPU day = $1 on high-end PC, less on cloud infrastructure
    - Strength of a key derived from passphrase?

$$K = SHA-256(\text{“verYsekReTT123pasSfraZe”})$$

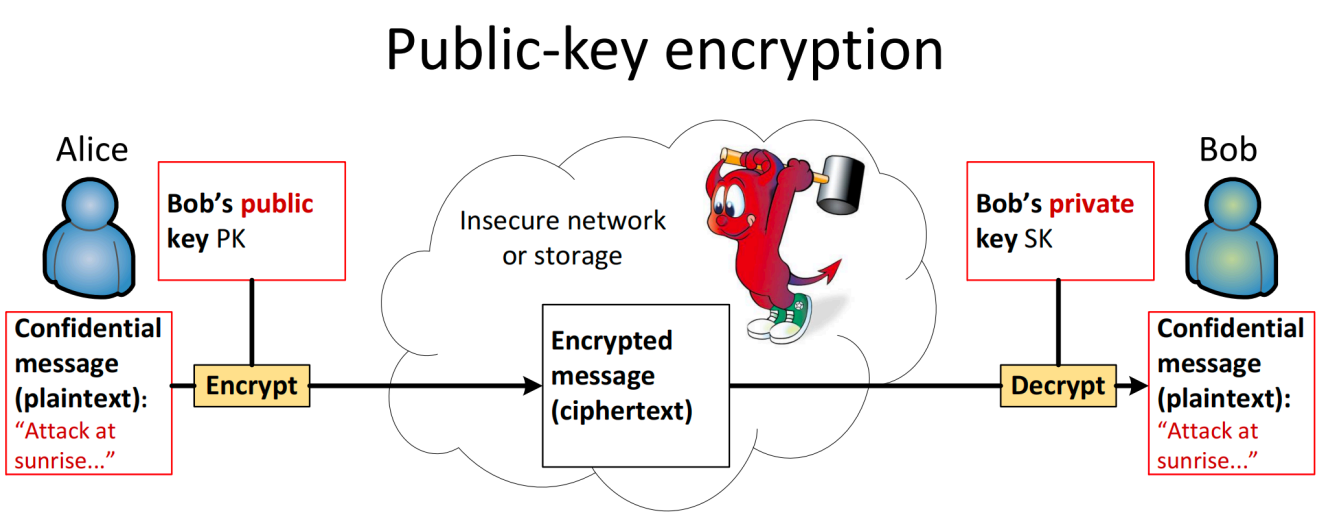
        - Dictionary attack to guess human-invented passphrases is possible, while brute-forcing a random 128 or 256-bit key is not
- Hash length and birthday paradox
    - One-wayness and second preimage resistance require has length of 128..256 bits.
        - Attacker tries different inputs to match a known hash value. Impossible to perform 2^128 hash computations
    - Collision resistance requires almost twice that length.
        - Birthday attack: store computed hash values and find a match between any two of them

4.4 Asymmetric encryption

- Digital signature



  - Message authentication and integrity protection
  - Asymmetric i.e. public-key cryptography
  - Key pair with public and private parts
- Public-key encryption



  - Asymmetric encryption: public key and private key
  - Protects secrets, not integrity
- Hybrid encryption
  - Symmetric encryption is fast; asymmetric is convenient
  - Hybrid encryption = symmetric encryption with random session key + asymmetric encryption of the session key
- Key distribution
  - The advantage of public-key cryptography is easier key distribution

    - Shared secret keys, symmetric cryptography:
        - O(N2) pairwise keys for N participants → does not scale
        - Keys must be kept secret → hard to distribute safely
    - Public-key protocols, asymmetric cryptography:
        - N key pairs needed, one for each participant (or 2·N if different key pairs for encryption and signature)
        - Public keys are public → can be posted on the Internet

4.5 Diffie-Hellman key exchange

- Diffie-Hellman key exchange
    - Creating a shared key based on commutative operation, such as exponentiation modulo p:

$$(g^x \mod p)^y \mod p = (g^y \mod p)^x \mod p$$

    - Diffie-Hellman assumption: given g, p, gx and gy, it is infeasible to solve gxy
    - Security depends on the difficulty of the discrete logarithm problem, i.e. solving x from (gx mod p) when p is large
    - Elliptic curve Diffie-Hellman uses commutative operations in a different field

5 Data Encruption

5.1 Scenarios for data encryption

- Data encryption
    - Scenarios:
        - lost and stolen notebook computers
        - stolen workstations and servers
        - decommissioning hard drives
          → Risk of disclosure of confidential data
    - The obvious solution: encrypt data on drive
- Scenarios for data encryption
    - Lost and stolen laptops

        - Contain confidential data and access credentials
    - Physically compromised servers
        - Contain business secrets, customer data and PII
        - Unauthorized insiders have physical access
        - Datacenter or shared server room is not fully in our control
    - Decommissioned hard drives
        - Secure decommissioning is expensive
        - Hardware recycling is typically done in the cheapest and fastest way: no time for secure disk wipe
        - Old PCs may be sent abroad for recycling

5.2 File encryption

- Simple file encryption
    1. User enterspassphrase
    2. Passphrase hashed to produce a key
    3. File encrypted with the key
        - Symmetric encryption, e.g. AES in CBC mode
        - Integrity check with HMAC or AES-GCM

        > Examples: crypt(1), gpg, openssl

- Limitations of file encryption
    - User action needed, and users are lazy
      – Automation (scripting) is difficult. How to store passphrase?
    - Passphrase can be brute-forced
    - After encryption, what happens to the old plaintext file?
    - Software creates temporary files and backup copies
        - Unencrypted file versions and data fragments may be left on disk
    - Incompatibility with advanced services:
        - Background processing like search indexing
        - Cloud storage and sharing
- Difficulty of deleting files
    - Deleting a file marks the space free but does not erase data
    - Overwriting a file does not always erase the old contents
        - Unpredictable file system behavior: backups, version history, RAID, copy on write, journal, bad blocks etc.
        - Solid-state disks (SSD) avoid rewriting the same physical blocks

- Wiping files
  - Overwriting all free disk space
    - Must write pseudorandom data, not just zeros or ones
    - Should read and verify after write
    - Erases most data, but no 100% guarantee (e.g. SSD overprovisioning can hide up to 20% capacity)
    - Magnetic data remanence on magnetic media
  - Physical destruction
    - Heating magnetic medium above its Curie temperature
    - Grinding SSD or disks to fine powder

5.3 Encrypting file system EFS

- Windows encrypting file system (EFS)
  - Encryption is a file attribute in NTFS
  - Enable encryption for a folder → new files encrypted from start
  - Files can be read when the user is logged in
  - Encryption and decryption are transparent to applications
- EFS key hierarchy
  1. User logs in with password
  2. Hashed to produce key
  3. Used to decrypt User's Master Key
  4. Used to decrypt User's Private EFS Key
  5. Used to decrypt File Encryption Key (FEK)
  6. Used to encrypt on write and decrypt on read
- EFS limitations: partial protection
  - Encrypts contents of specific files and folders
  - Some data is not encrypted:
    - Folder and file names
    - EFS-unaware software may create backup copies, temp files
    - Registry, system files and logs cannot be encrypted
    - Page file can be encrypted but requires policy configuration
    - Search index, if you has chosen to index the encrypted files
  - When encrypting plaintext files, the original file is not wiped: always create files in an encrypted folder

    - Hibernation file may store decryption keys: disable hibernation
    - EFS only on NTFS and FAT; files copied elsewhere are decrypted
- EFS limitations: attacks
    - Requires strong passwords; otherwise, the password could be cracked from the password hash stored on the disk
    - Key logger or hidden camera can capture user password
    - Malware can steal EFS data when the user is logged in
    - Attacker with physical access to the computer or its hard disk can compromise the OS and install a root kit or key logger. Easiest to configure a malicious data recovery agent (DRA)
- EFS limitations: usability
    - User must understand what data is encrypted and when
    - Non-domain users must backup EFS keys or risk data loss
    - User password needed for decryption
        - OS cannot index files when user logs out
        - Backups contain encrypted files (store old EFS keys as long as the backup!)
        - Password reset by admin or special tools may lead to loss of data
    - Transparent decryption can cause surprises
        - Files copied to online storage and other media are decrypted
        - Files copied to USB stick (FAT) are converted by default to an encrypted PFILE, which cannot be opened in another computer

5.4 Full disk encryption: BitLocker

- Full disk encryption
    - Entire disk is encrypted: protects all information
    - Usually, user must provide password, key, or physical token at boot time or to mount the disk; thereafter transparent encryption
    - Software vs hardware based solutions:
        - Security of entirely software-based products depends on the strength of the password or key
        - Tamper-proof hardware prevents brute-force cracking and allows the use of a short PIN
    - The main innovation in BitLocker was to use tamper-proof hardware for unsupervised boot
- Trusted platform module
    - Trusted platform module (TPM) is a smart-card-like tamperproof module on the motherboard or integrated in the CPU
    - Stores cryptographic keys
        - Protects keys against both software and hardware attacks
        - Key cannot be stolen, but can it be misused?

    - Accumulates platform measurements in platform configuration registers (PCR) for checking software integrity during boot
- Sealing data with TPM
    - TPM operations:
        - Sealing (TMP2_Create): encrypt data in any platform configuration
        - Unsealing (TPM2_Unseal): decrypt the data, but only if the platform configuration has not changed from the time of sealing
    - Before unsealing, the TPM compares current PCR values to ones saved when sealing
- Windows BitLocker
    - Principle: Encrypt drive with symmetric encryption. Seal the symmetric encryption key with TPM and store the sealed key on the disk. Unseal the key when booting
        - TPM detects tampering of system software and prevents booting by refusing to unseal the key
- DMA attack
    - Direct memory access (DMA) enables auxiliary devices to access main memory directly
        - FireWire, PCIe (also Thunderbolt, M.2, ExpressCard) busses
        - Hot-pluggable helps attacker
    - DMA memory dump attack: Attacker connects malicious device to a DMA port, reads the memory, and recovers the key (FVEK)
    - Windows 10 **disables** hot-pluggable PCIe ports when the screen locked
- Hardware security issues
    - Communication between the TPM and CPU is vulnerable
        - If TPM is on discrete chip, attacker can tap the bus to it. Breaks security of the TPM-only mode by sniffing the VMK
        - Some TPMs are on a separate PCB, making the attack easy [link]
    - The TPM itself must be secure
        - Infineon TPMs had a key-generation bug in 2017 that required firmware update and generation of a new storage root key [link]
        - Sleep mode implementation flaws allow overwriting of PRC values, which allows attacker to unseal VMK in TPM-only mode
- Cold boot attack
    - Data remanence in main memory:
        - After power loss, DRAM memory contents decay in seconds
    - Cold boot attack (reset attack)
        - Reboot into a minimal hacker OS from USB stick
        - Rebooting cuts the power only for a fraction of a second
        - When power is back, memory contents do not decay any further

        - Secret keys can be recovered from memory (possibly with some bit errors)
    - Best protection: never given attacker access to a running computer
        - Disable sleep on the computer, shut down or hibernate instead
        - Do not leave computer running unsupervised
        - Set BitLocker to a supervised boot mode (TPM+PIN, TPM+USB)
    - Weaker protection:
        - Password-protect BIOS or UEFI and disable USB boot
        - BitLocker network unlock for non-mobile domain computers
        - UEFI secure boot: only allow certified OS loaders
    - OS and hardware solutions:
        - Overwrite memory before reboot (easy to circumvent)
        - Memory Overwrite Request Control (MOR): UEFI cleans the memory after reboot
        - Encrypted main memory in Xbox
        - Self-encrypting drive: encryption on the disk drive itself; keys are only briefly in main memory

5.5 Data recovery

- Need for data recovery: EFS

    > If the decryption key is lost, encrypted files will be lost

    - EFS files cannot be read after password reset
        - Reset by admin and hacking tools have the same effect
        - Password change by the user, of course, is ok
    - EFS files in backup media unreadable if the user's EFS private keys are lost
        - Can happen when rebuilding or cleaning user profile or replacing the computer
    - Removable FAT drives may contain encrypted files that depend on the computer that wrote them
- Data recovery in EFS: domain
    - Windows domain has a data recovery agent (DRA)
        - FEK is encrypted also with the DRA public key
        - Domain Administrator is the default DRA
        - Other DRAs can be defined in a Group Policy in the domain
    - If uncertain, check on any EFS-encrypted file: cipher /c .\file.txt
- Data recovery in EFS: non-domain
    - Standalone machine has no default DRA
        - Possible to set up your own DRA (cipher.exe)

    - User may backup their EFS certificate and private key to a USB or cloud drive
        - Open certmgr.msc, under Personal, export all Encrypting File System certificates with private key
- Need for data recovery: BitLocker

> If the TPM fails to unseal keys, encrypted data will be lost

    - TPM cannot unseal the key when
        - Installing or updating Linux boot loader (dual boot)
        - Motherboard with TPM replaced
        - Processor with embedded TPM replaced
        - TPM has been reset
        - Hard drive moved to another computer
        - TPM boot PIN forgotten
        - TPM boot PIN mistyped many times
- Data recovery in BitLocker
    - Recovery password:
        - User can print a 48-digit recovery password or store it on a USB stick or remote disk; it is actually a 128-bit key
        - BitLocker encrypts the VMK with the recovery password and stores it with the volume metadata (in the same way as the TMP-sealed VMK)
    - Domain computers save recovery key to Active Directory, others to Microsoft account or employer/university Azure AD

6 Network Security

6.1 Network threat model

- Network-security threat model
    - Endpoints are trusted; network is the attacker
    - The network may deliver, delete, modify, and send fake messages
- Network security goals
    - Data confidentiality: secrets only revealed to intended parties
    - Data integrity: receiver can detect data modification
    - Data-origin authentication: receiver verifies who sent the data
    - Data and service availability: communication successful
- Basic attack types

- Data confidentiality
  - ↔ sniffing = eavesdropping = interception = spying
- Data integrity
  - ↔ unauthorized data modification = tampering
- Data-origin authentication
  - ↔ spoofing or impersonation
- Data and service availability
  - ↔ denial of service (DoS)

6.2 Replay and freshness

- Replay attack
  - attacker records the message and resends later
  - Sequence number prevents replays
    - Receiver checks that the number increases and never repeats
    - Attacker cannot copy the message but can delay it
  - Timestamp prevents delaying of messages
    - Receiver does not accept messages older than e.g. one minute
    - message be replayed back to the sender
      - Can the same entity act as both user U and device D? Often possible
  - Explicit direction, or sender and receiver identity & Separate key (and counter) for each direction
    - Maybe the device does not have a reliable clock

6.3 Authenticated Diffie-Hellman

- Unauthenticated Diffie-Hellman
  - A and B have previously agreed on g and p
  - All operations are modulo p

```
A chooses a random x. B chooses a random y.
1. A → B: A, g^x
2. B → A: B, g^y
   A calculates shared secret SK = (g^y)^x = g^xy
   B calculates shared secret SK = (g^x)^y = g^xy
```

  - **Sniffer learns g^x and g^y, cannot compute x, y, or g^xy**
- Man-in-the-middle

- Unauthenticated Diffie-Hellman is secure against passive sniffing but insecure against active attackers

- Authenticated Diffie-Hellman

1. A → B: A, B, N_A, g, p, g^x, $Sign_A$("Msg1", A, B, N_A, g, p, g^x), $Cert_A$
2. B → A: A, B, N_B, g^y, $Sign_B$("Msg2", A, B, N_B, g^y), $Cert_B$, MAC_{SK}(A, B, "Responder done.")
3. A → B: A, B, MAC_{SK}(A, B, "Initiator done.")

SK = h(N_A, N_B, g^{xy})

7 PKI & TLS

7.1 X.509 certificates and PKI

- Key distribution problem
  - How to find out someone's authentic public key?
  - Solution: an authority issues identity certificates that bind public keys to names
  - Certificate is a message signed by the issuer, containing the subject's name (or identifier) and the subject's public key

$$Certificate = Sign_{issuer}(Name, PK_{subject}, validity_{period})$$

- X.509 certificate fields
  - Mandatory fields:
    - Version
    - Serial number — together with Issuer, uniquely identifiers the certificate
    - Signature algorithm — for the signature on this certificate; usually sha1RSA; includes any parameters
    - Issuer — name (e.g. CN = Microsoft Corp Enterprise CA 2)
    - Valid from — usually the time when issued
    - Valid to — expiry time
    - Subject — distinguished name of the subject
    - Public key — public key of the subject
  - Common extension fields:
    - Key usage — bit field indicating usages for the subject key (*digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment, keyAgreement, keyCertSign, cRLSign,*

encipherOnly, decipherOnly)

        - Subject alternative name — email address, DNS name, IP address, etc.
        - Issuer alternative name
        - Basic constraints — (1) is the subject a CA or an end entity, (2) maximum length of delegation to sub-CAs after the subject
        - Name constraints — limit the authority of the CA
        - Certificate policies — list of OIDs to indicate policies for the certificate
        - Policy constraints — certificate policies
        - Extended key usage — list of OIDs for new usages, e.g. server authentication, client authentication, code signing, email protection, EFS key, etc.
        - CRL distribution point — where to get the CRL for this certificate, and who issues CRLs
        - Authority info access — where to find information about the CA and its policies
- X.509 CA hierachy
    - One root CA
    - Each CA can delegate its authority to sub-CAs
    - All end-entities trust all CAs to be honest and competent
- Self-signed certificate
    - Issuer and subject keys are the same
    - Often included in the certificate chain
    - Not really a certificate; just a way to store and communicate the root CA public key
- Real-world PKIs
    - Original X.500 idea: one global CA hierarchy to certify all countries, organizations, users, computer and services
    - Reality: many application and organization specific PKIs
        - Web PKI for certifying web servers
            - Many commercial and free root CAs, e.g. Verisign, Telia, Let's Encrypt
        - S/MIME for signed (and encrypted) email
            - Commercial CAs certify organizational CAs for cross-organization email
        - Smart-card PKIs
            - Bank cards, national identity cards
        - Organizational PKIs
            - Windows domain users, computers and services; user login with smartcard; internal web pages; VPN and Wi-Fi access; S/MIME email; Adobe document signing PDF documents
- Need for certificate revocation
    - When might CA need to revoke (i.e. cancel) a certificate?

        - If the conditions for issuing the certificate no longer hold:e.g., employee leaves, student graduates, or computer is decommissioned
        - If the certificate was originally issued in error
        - If the subject private key has been compromised
        - When upgrading cryptographic algorithms
    - Certificate can be verified offline, but revocation requires online checks
- Revocation list
    - Certificate revocation list (CRL)
        - = signed list of revoked certificate serial numbers and a timestamp
        - Who issues the CRL? How to find it?
            - CRL distribution point and issuer may be specified in each certificate
            - By default, CRL is signed by the same CA that issued the certificate
        - Certificate verifier downloads the CRL (or delta) and caches it
            - If CRL server is offline, the certificate verification fails
        - Expired certificates can be removed from the CRL
        - In X.509, only certificates are revoked, not keys
    - OCSP
        - Online certificate status protocol (OCSP)
            - Request-response protocol for checking certificate status from issuer
            - Timestamp and optional nonce for response freshness

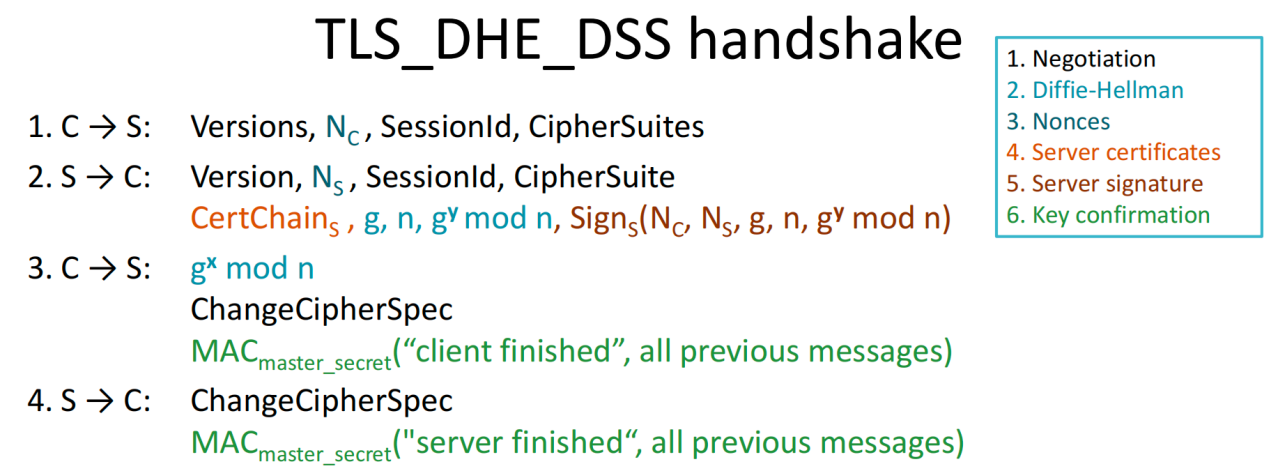
7.2 Web PKI

- Web PKI
    1. Root CA's self-signed certificate Issued by the root CA to itself;
        - essentially just the CA public key
    2. Root CA issues a CA certificate to a sub-CA
        - Typically one sub-CAs in the chain (why?)
    3. Sub-CA issues end-entity certificate to a web server
- Problems with revocation in web PKI
    - Web clients typically ignore CRL or OCSP, especially if the server does not respond
    - If a sub-CA is compromised, it won't be detected and, thus, won't be revoked
- Certificate Transparency
    - Certificate Transparency (CT) ensures that all certificates are publicly visible.
    - Both clients and servers can benefit from CT.

        - Server owners can watch the CT logs to ensure that unauthorised certificates are not issued for their domains, providing a direct benefit for them.
        - In addition, security experts can monitor the logs to ensure that certificates aren't issued that are insecure or violate other policies that can't be automatically checked by the browser, e.g. using obsolete cryptographic algorithms such as SHA-1, or with too-long validity periods, or backdated to the past.

7.3 Transport Layer Security (TLS)

- TLS in the protocol stack
    - TLS implements cryptographic encryption and authentication for TCP connections
        - Secure socket API for applications
- Handshake and session
    - Two stages of a typical network security protocol:
        - Handshake = authenticated key exchange creates a shared session key
        - Session protocol protects the confidentiality and integrity of the session data with symmetric cryptography and the session key
    - TLS handshake
        - Client and server create a shared secret key with Diffie-Hellman
        - Server authenticates to the client with a certificate chain and signature
        - Client authentication optional, usually left to the application layer
    - TLS session protocol uses symmetric encryption and HMAC to protect the application data
- TLS handshake



        - Shared secret: g^{xy} mod n
        - master_secret = h(g^{xy} mod n, "master secret", N_C, N_S)
        - ChangeCipherSpec turns on session protection with the new key

    - Trust chain

- In the handshake, browser receives a certificate chain from the server
- Browser checks that the chain start with a (usually self-signed) certificate that is in its trusted CA list
- Browser checks the certificate chain:
  - Verifies the signature on each certificate using the subject public key of the certificate above
  - Checks that all but the last certificate are CA certificates
  - Many other details, e.g. validity time, CRL/OCSP, key usage, constraints
  - Checks that certificate appears in CT log
- If the certificate chain is valid, the last certificate binds together
  - the host name and public key of the server
  - Public key is used for server authentication in the TLS handshake
  - Host name shown to user in the browser address bar

- Certificate checking details
  - Certificate verification is quite complex and difficult to implement correctly:
    1. Browser has a list of self-signed certificates for trusted root CAs, and it may have lists of certificates for trusted sub-CAs and servers.
    2. In the TLS handshake, the browser may tell the server which root CAs it recognizes.
    3. The browser receives a certificate chain from the server.
    4. Browser checks the validity of the certificate chain backwards ("upwards") from the end-entity-certificate towards the root:

       > A. There must be exactly one end-entity certificate at the bottom of the chain. The other certificates in the chain must be CA certificates.
       > B. Issuer of each certificate must match the subject of the CA certificate above it.
       > C. The browser verifies the signature of each certificate with the subject public key of the certificate above, i.e., from the issuer's CA certificate.
       > D. Browser checks for certificate revocation from the OCSP server or CRL of if the certificate specifies these.
       > E. Browser checks that the certificate is in a CT log.
       > F. There may be constraints in the certificates, which must also be checked. Name constraint limits the authority of a CA to specific names, usually a domain suffix. The name in the end-entity certificate must match all the name constraints in the certificate chain.
       > G. The browser must recognize and process all critical extension fields in the certificates, but it may ignore non-critical extensions. (For example, name constraint and key usage are critical extensions, but CLR distribution point and Certificate Transparency timestamps are non-critical.)
       > If a trusted certificate is found, stop going up the chain and move to the next step. On the other hand, if the root of the chain is reached and no trusted certificate is found, the chain verification fails.

5. Browser checks that the certificate has been issued for the right purpose: extended key usage field of the end-entity certificate must specify TLS server authentication.
6. Browser checks that the host name in the browser's address bar or requested ULR matches the subject name of the end-entity certificate. (Subject name matching rules are pretty complex, too. There can be many names and wildcards in the certificate.)
7. Browser uses the subject key from the end-entity certificate to authenticate the server in the TLS handshake (authenticated key exchange).
8. The session key created in the handshake is used to encrypt and authenticate data between the browser and server for the duration of the TLS session.

> This process proves that the web page shown in the browser comes from the server whose name is in the address bar

- TLS session protocol
  - After the handshake, data is protected with the session protocol
  - Data confidentiality is protected with symmetric encryption, e.g. AES in CBC mode
  - Data integrity is protected with message authentication codes (MAC)
  - Secret session keys for encryption and authentication in each direction are derived from the master_secret

8 Threat Analysis

8.1 Security terminology

- Some security terminology
  - Threat = bad event that might happen
  - Attack = intentionally causing the bad thing to happen
  - Vulnerability = weakness in an information system that enables attacks
  - Exploit = implementation of an attack
  - Risk = probability of an attack × damage in euros
- Security Goals
  - Confidentiality, Integrity, Availability "CIA"
    - Confidentiality — protection of secrets
    - Integrity — only authorized modifications
    - Availability —service works, business continuity
  - Examples: web server, customer data
  - Many security goals are not covered by CIA:
    - Access control — only authorized use of resources

        - Privacy — control of personal data and space
- Typical attackers:
    - Curious individuals
    - Friends and family
    - Dishonest people — for personal gain, making and saving money
    - Hackers, script kiddies — for challenge and reputation
    - Companies — for business intelligence and marketing, industrial espionage
    - Organized criminals, rogue countries — for money and power
    - Governments and security agencies — NSA, SVR RF, GCHQ, DGSE, etc.
    - Military SIGINT — strategic and tactical intelligence, cyber defense

8.2 Threat analysis

- Views
    - Systems architecture
    - Assets
    - Actors and their possible motivation
    - Threats and potential attacks

8.3 Systematic threat modeling

- Threats considered in STRIDE:
    - Spoofing vs. authentication
    - Tampering vs. integrity
    - Repudiation vs. non-repudiation, accountability
    - Information disclosure vs. confidentiality
    - Denial of service vs. availability
    - Elevation of privilege vs. authorization, access control
- In DREAD, risk has many dimensions:
    - Damage: how much does the attack cost to defender?
    - Reproducibility: how reliable is the attack
    - Exploitability: how much work to implement the attack?
    - Affected users: how many people impacted?
    - Discoverability: how likely are attackers to discover the vulnerability?

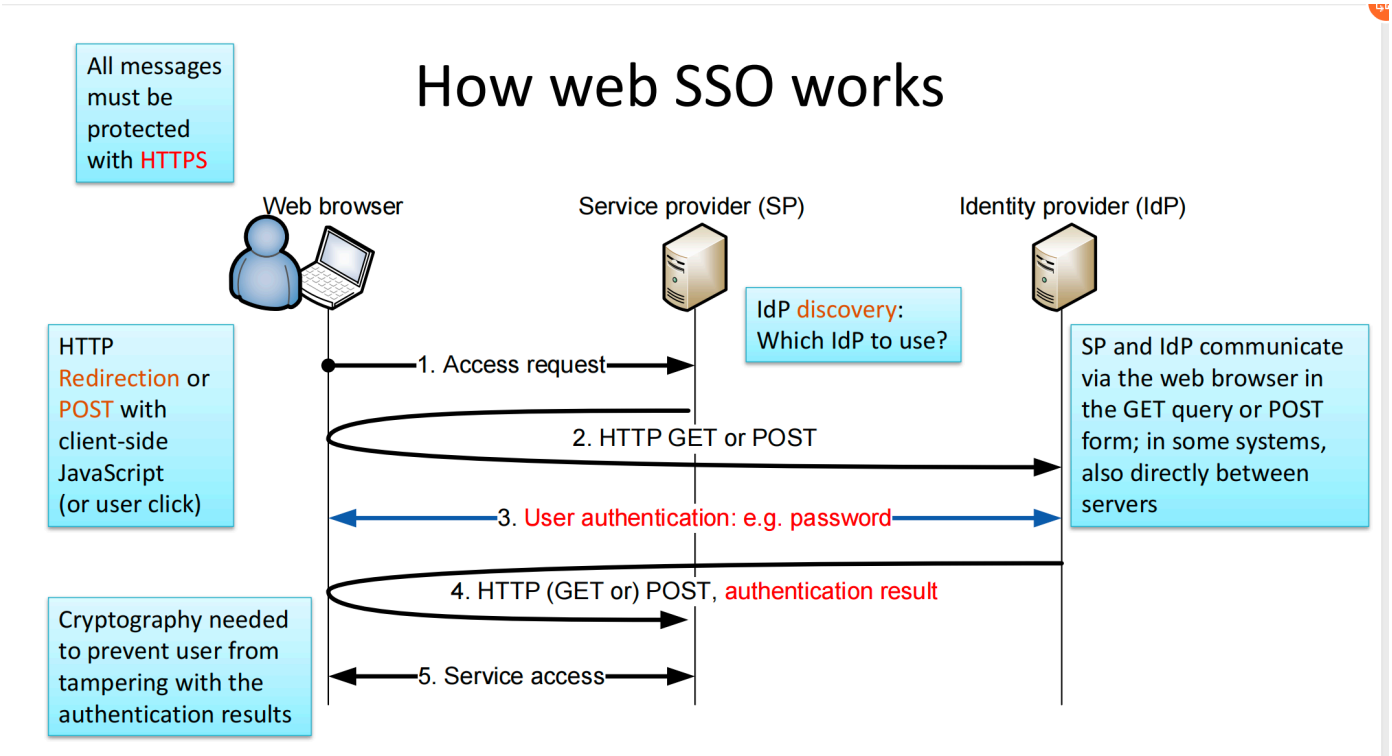
9 Electronic Identity

9.1 Single sign-on (SSO)

- One authentication credential for many services. Why?
- For users:
  - Fewer accounts and passwords to remember
  - More convenient service access
  - Lower risk from old, forgotten accounts
- For intranet and extranet services within an organization:
  - Central user management within organization
- For public web sites:
  - Outsourcing credential provisioning and authentication
  - Tracking web users for advertising

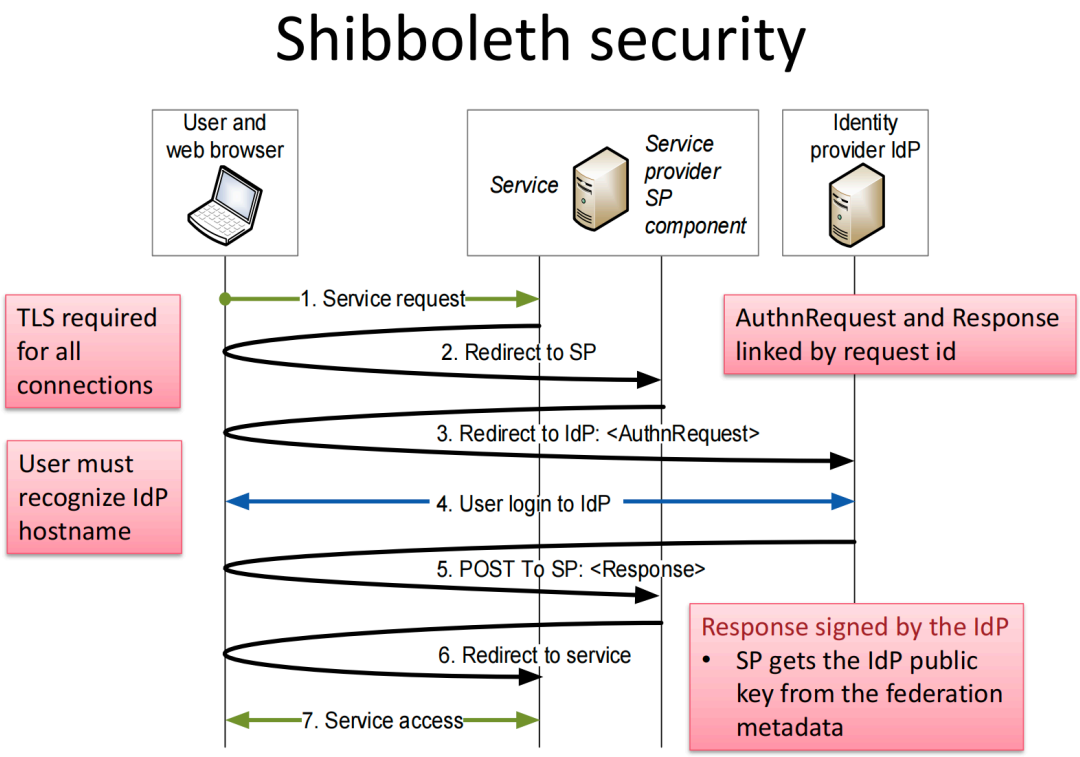
9.2 Web single sign-on within organization

- Web SSO



- SAML & SHIBBOLETH
  - Shibboleth 2
    - Open-source implementation of SAML 2.0
      - SAML web browser SSO profile

        - Used by research and educational institutions
    - Shibboleth Security

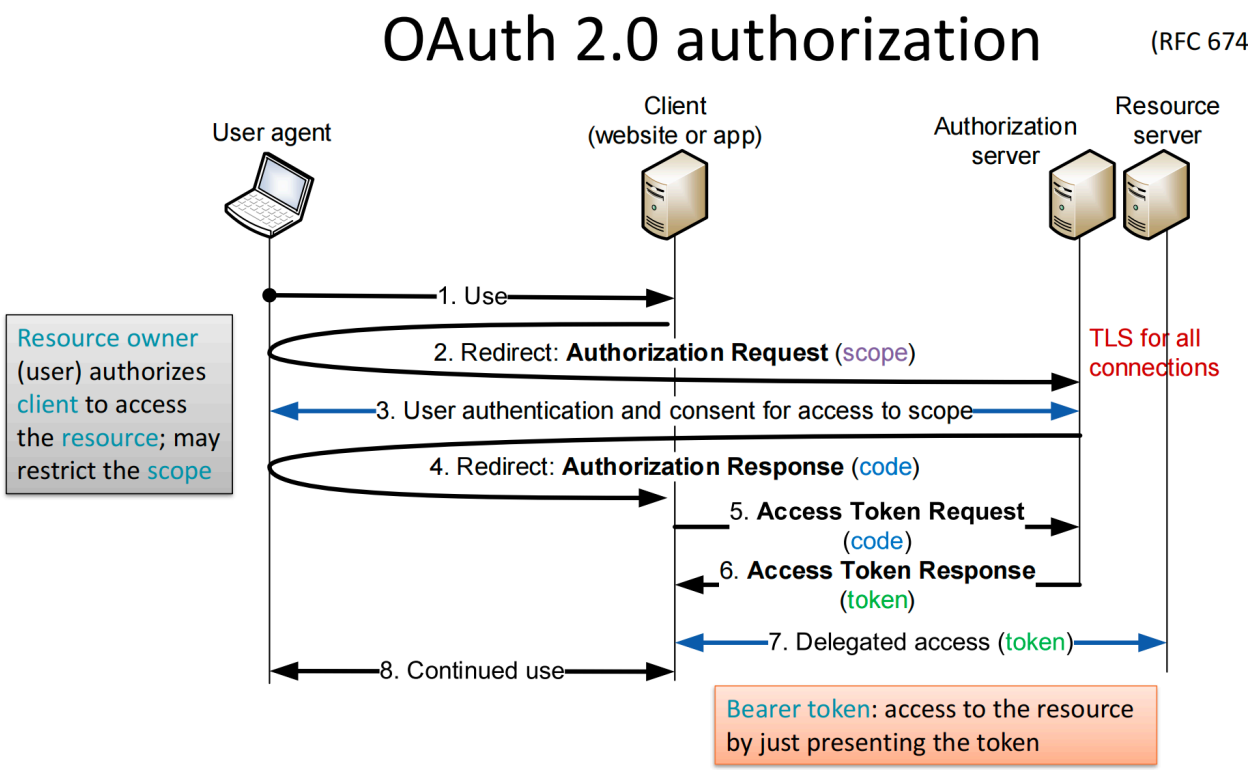


- Sessions in Shibboleth
    - 3 sessions
        - IdP session
        - SP session
        - service session
    - Many different sessions → logout is confusing (which sessions end?)
    - Logout is a problem in all SSO systems: hard to implement, and not obvious how it should work

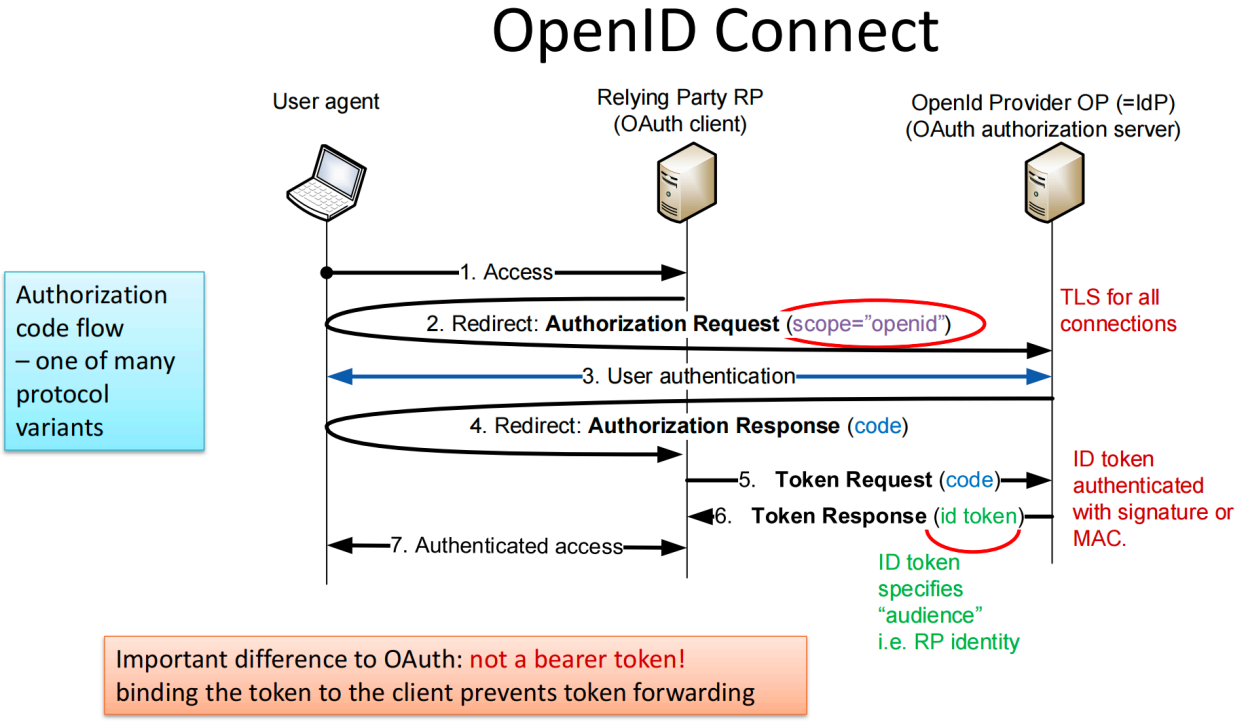
9.3 Web single sign-on on the Internet

- OAuth 2.0
    - OAuth was designed for authorization (i.e., delegation)
        - User authorizes a web app, mobile app, or another online service
        - service to access their data in an online service
    - Examples:
        - Authorize a website or app to update Facebook for you
        - Authorize continuous integration tool to monitor a GitHub repository

- Authorization



- OPENID CONNECT



  - Authentication built on OAuth 2.0, JavaScript, REST APIs, JSON data formats
    - OAuth access token is replaced with id token, which is not a bearer token;
    - the token is bound to the specific relying party (OAuth client)
  - Implementations are usually not interoperable

        - Many options: MAC with pre-shared key between OP and RP vs. JSON Web Signature on messages
        - OP provides both server-side RP code and client-side JavaScript

    > → no need for interoperability

9.4 Authentication to government services

- Strong authentication regulation
    - Two-method authentication, any two of:
        - something you know (password or PIN code)
        - something you have (physical token)
        - something you are (biometrics)
    - Identity proofing in person or electronically