IT Security

Jil Zerndt
FS 2025

01 - Intro

Business IT-Risks

- Ransom Demand
- Fraud
- Espionage
- Violation of Regulations
- Misuse of Computing Resources
- Reputation Loss
- System Outage
- Sabotage
- Data Loss
- Brand Misuse

Problems - Overview and their impact on data availability

Physisch

unabsichtlich

- Naturkatastrophen
- Feuer
- Ausfall
- Kaffee auf Server

absichtlich/bösartig

- Feuer
- Vandalismus
- Garantie läuft aus -> absichtlich langsamer
- Social Engineering

Virtuell

unabsichtlich

- Bitflip
- Config Fehler
- Bugs im SW
- Phishing klicken

absichtlich/bösartig

- DDoS
- Malware
- Ransomware
- Phishing senden
- Trojaner

Countermeasures - Overview

Disaster Recovery

- Offline backup solutions
- Restoring from images

Access Control

- Restricted Access Rights
- Multi-Factor Authentication
- Firewalls
- Traffic Management Solutions

Physical Protection

- Physical Access Control (locks, fences, etc.)
- Fire Protection (extinguishers, alarms, etc.)
- Monitoring (CCTV, Guards etc.)

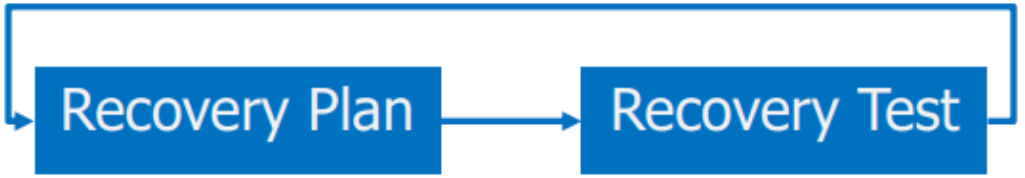
Training Processes

- Employee Training
- Four eyes principle
- Automation of routine processes
- Monitoring
- Preventive maintenance

Redundancy

- Uninterruptable Power Supplies
- High Availability setups
- Load Balancing
- Redundant data center
- Redundant network connections

Recovery Plan and Test



Recovery Plan - description of what to do if something goes wrong

- Roles and responsibilities
- Processes
- Contact details
- Technical instructions

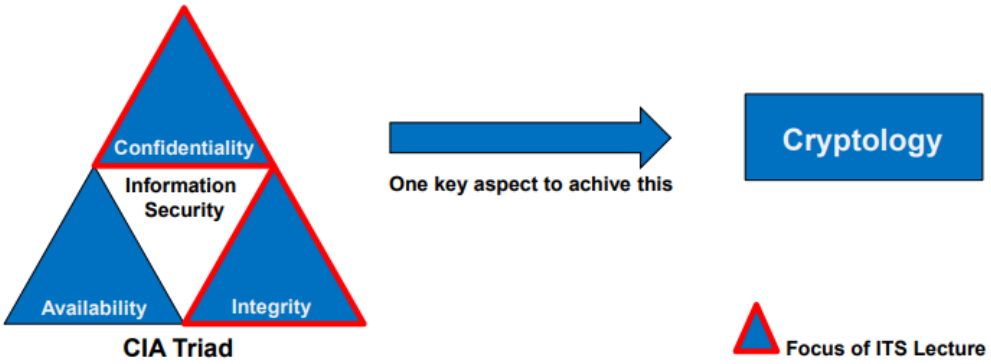
Recovery Test - testing the recovery plan

- Theoretical dry run
- Practical tests
  - turn off a server or DC
  - restore data from backup

Goals of IT Security

Most measures in Information Security have one of the three following high-level goals:

- Ensure data is confidential
- Ensure data is not corrupted
- Ensure data and systems are available



Overview of IT Security

Introduction to IT Security

Key IT Security Goals

Information security is based on three fundamental principles, commonly known as the CIA triad:

- **Confidentiality** - Ensuring data is only accessible to authorized users
- **Integrity** - Ensuring data is not modified in an unauthorized way
- **Availability** - Ensuring systems and data are accessible when needed

Business IT Risks

Organizations face numerous security threats that can impact business operations:

- Data loss
- System outage
- Espionage
- Sabotage
- Reputation loss
- Misuse of computing resources
- Violation of regulations
- Fraud
- Brand misuse
- Ransom demands

These risks can have significant financial, operational, and reputational impacts.

Security Frameworks and Controls

Security Control Frameworks

Security frameworks provide structured approaches to implementing security controls:

- **CIS Controls** - Prioritized set of actions to protect organizations
- Controls are typically organized in implementation groups based on difficulty and impact
- Focus on preventing the most common attack vectors first

Types of Security Measures

Security measures can be categorized based on their focus:

- **Preventive** - Block threats before they occur (firewalls, access controls)
- **Detective** - Identify when a breach has occurred (IDS, audit logs)
- **Corrective** - Mitigate damage after an incident (backups, incident response)

Disaster Recovery

Business Continuity Management

Disaster recovery and business continuity planning are essential for maintaining availability:

- **Recovery Plan** - Detailed procedures for recovering from incidents
- **Recovery Tests** - Regular testing of recovery procedures
- **Redundancy** - Duplicate systems, power supplies, and network connections
- **Offline backups** - Protection against ransomware and other threats

Disaster Recovery Planning

Initial Assessment

- Identify critical systems and data
- Determine acceptable recovery time objectives (RTO)
- Determine acceptable recovery point objectives (RPO)

Plan Development

- Document recovery procedures
- Assign roles and responsibilities
- Include contact details for all relevant parties
- Develop technical instructions for restoration

Testing

- Conduct regular theoretical dry runs
- Perform practical tests (e.g., server shutdown, data restoration)
- Update procedures based on test results

Regular Review

- Review and update plans regularly
- Consider changes in infrastructure, personnel, and threats

A medium-sized company implements a disaster recovery plan for their customer database. They define an RTO of 4 hours and an RPO of 15 minutes, meaning they need to restore service within 4 hours with no more than 15 minutes of data loss. To achieve this, they implement a combination of hourly differential backups with continuous transaction log shipping to a standby site. Regular recovery tests are scheduled quarterly to ensure the plan remains effective.

Kryptographie

Einführung in die Kryptographie

In diesem Kapitel schauen wir uns die Ziele, Grundbegriffe und Modelle der Kryptologie an.

Zweige der Kryptologie

Die **Kryptologie** ist ein Teil der Mathematik, welcher sich mit dem sicheren Übertragen und Speichern von Nachrichten beschäftigt.

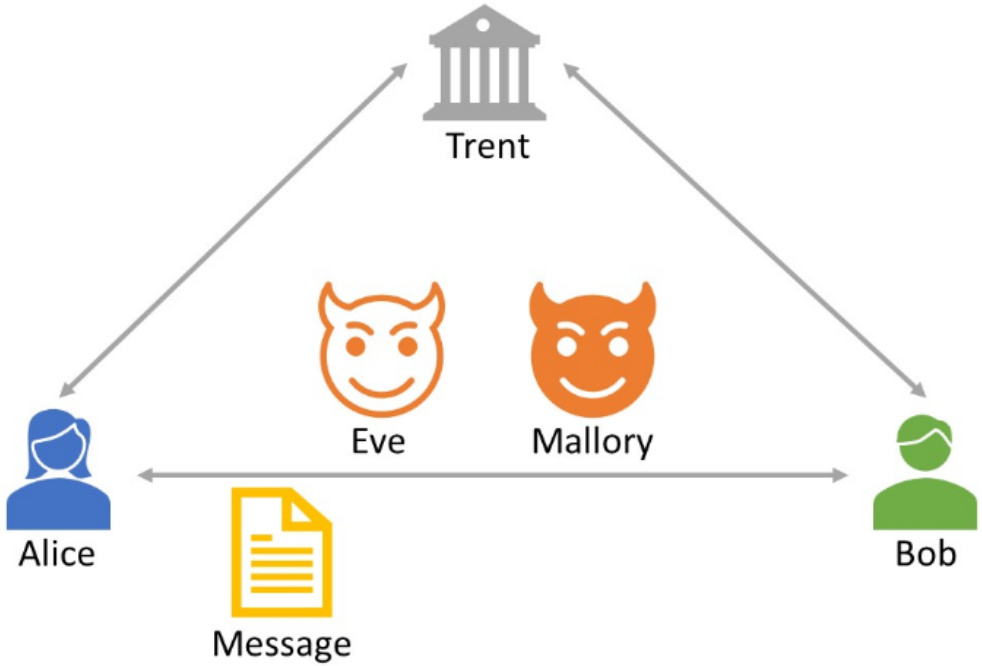
- **Kryptographie**: Die Wissenschaft der Verschlüsselung von Nachrichten (sicheres Speichern und Übertragen) → sichere Protokolle und deren Aufbau
- **Kryptoanalyse**: Die Wissenschaft des Entschlüsselns von Nachrichten (wie können Mechanismen der Kryptographie gebrochen werden?) → alte, unsichere Protokolle und deren Schwachstellen

Ziele der Kryptographie Nachrichten sicher übertragen und speichern. *Sicherheit* hat dabei verschiedene Aspekte:

- **Confidentiality:** Nur berechtigte Personen können eine Nachricht lesen. Unberechtigte Personen können die Nachricht zwar sehen, sie aber nicht entziffern, da die Nachricht für sie nur aus einer zufälligen Zeichenfolge zu bestehen scheint.
- **Integrity:** Eine Nachricht wird vom Empfänger so empfangen, wie sie vom Sender geschickt wurde. Das heisst, eine unberechtigte Person könnte zwar eine Nachricht abfangen, verändern, und dem Empfänger zustellen. Dieser würde dann aber merken, dass die Nachricht nicht der ursprünglichen Nachricht entspricht und sie daher verwerfen.
  → Der Empfänger kann sicher sein, dass die Nachricht nicht verändert wurde.
- **Authenticity:** Der Empfänger kann sicher sein, dass eine Nachricht auch wirklich von der Person stammt, von welcher die Nachricht zu kommen scheint. Das heisst, er kann überprüfen, ob der angegebene Absender auch dem tatsächlichen Absender entspricht.
- **Non-repudiation:** Wenn eine Person eine Nachricht bekommen hat, kann diese Person nicht abstreiten, dass sie die Nachricht erhalten hat. Das heisst, es kann bewiesen werden, dass sie die Nachricht erhalten hat.
- **Freshness:** Alle erhaltenen Nachrichten sind aktuell. Das heisst, ein Attacker könnte zwar eine Nachricht zurückbehalten und später senden, oder eine abgefangene Nachricht duplizieren und zu einem späteren Zeitpunkt noch einmal senden, aber dies würde vom Empfänger bemerkt.
- **Anonymity:** Der Absender und/oder Empfänger einer Nachricht bleiben unbekannt.

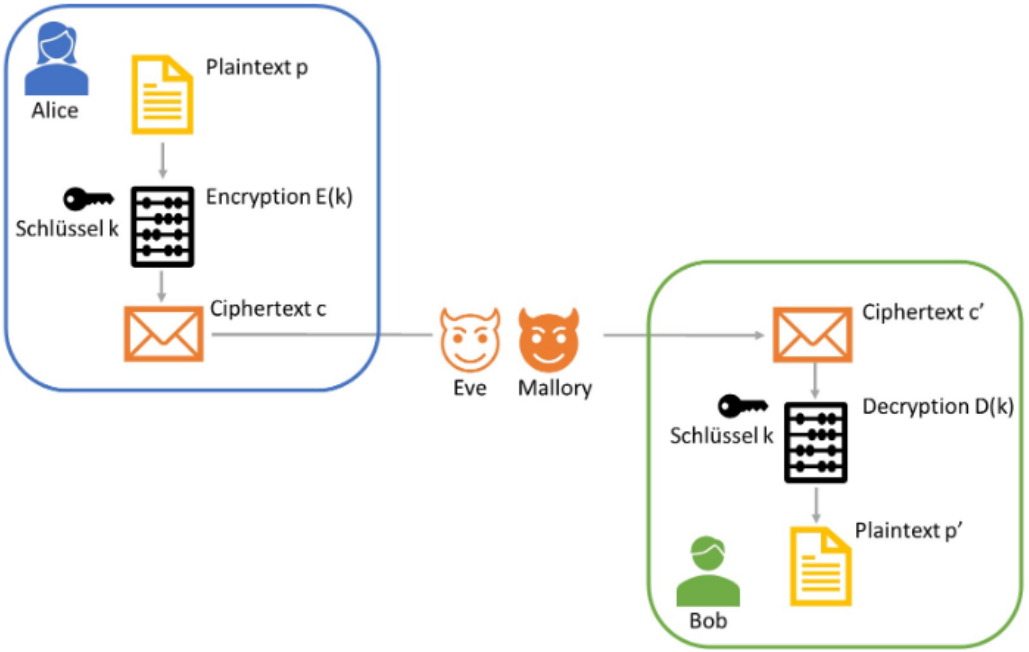
Model der Kommunikation und Verschlüsselung

Kommunikationsmodel



- **Alice** sendet eine Nachricht an Bob. Dabei sollen die oben genannten Ziele, confidentiality, integrity, authenticity, non-repudiation und freshness erfüllt werden.
- **Bob** empfängt die Nachrichten von Allice. Er will überprüfen können, dass die Ziele eingehalten wurden.
- **Eve** ist ein Attacker. Sie kann Nachrichten mitlesen, aber sie nicht verändern.
- **Mallory** ist ein anderer Attacker. Er kann Daten sowohl mitlesen als auch verändern. Er kann auch Nachrichten abfangen und später weitersenden, oder ganz verwerfen. Er kann auch neue Nachrichten generieren.
- **Trent** ist eine Drittperson/Instanz, welcher sowohl Alice und Bob vertrauen. Trent unterstützt Alice und Bob bei der sicheren Kommunikation.

Model der Verschlüsselung Um eine vertrauliche Kommunikation zu erreichen, werden Nachrichten vor dem Senden verschlüsselt und nach dem Empfangen wieder entschlüsselt.



- **Plaintext (Klartext):** Der Plaintext ist der Text, so wie er geschrieben, respektive gelesen werden kann. Er wird mit dem Buchstaben "päbgekürzt.
- **Ciphertext (Verschlüsselter Text):** Der Ciphertext ist der Text, welcher durch die Verschlüsselung entsteht. Er wird mit dem Buchstaben "cäbgekürzt.
- **Encryption (Verschlüsselung):** Die Verschlüsselung macht aus dem Plaintext den dazugehörenden Ciphertext. Dazu wird ein Schlüssel verwendet. Die Verschlüsselung kann wie folgt angegeben werden: $c = E[k](p)$. Der Verschlüsselungsalgorithmus selbst ist öffentlich bekannt und kann von allen analysiert werden, um mögliche Schwachstellen zu finden.
- **Decryption (Entschlüsselung):** Die Entschlüsselung macht aus einem Ciphertext den dazugehörenden Plaintext. Dazu wird ein Schlüssel verwendet. Die Entschlüsselung kann wie folgt angegeben werden p' = D[k](c'). Der Entschlüsselungsmechanismus ist öffentlich bekannt und kann von allen analysiert werden, um mögliche Schwachstellen zu finden.
- **Key (Schlüssel):** Nur mit dem richtigen Schlüssel, kann eine Nachricht richtig entschlüsselt werden. Je nach Art der Verschlüsselung wird derselbe Key für die Verschlüsselung und die Entschlüsselung verwendet (Secret Key Kryptographie) oder es werden unterschiedliche Schlüssel verwendet (Public Key Kryptographie). Damit die Verschlüsselung sicher ist, muss der Schlüssel, welcher für die Entschlüsselung gebraucht wird, geheim bleiben.

Confidentiality ist erreicht, wenn Eve (und Mallory) den Ciphertext c nicht lesen können. **Integrity** ist erreicht wenn der von Bob empfangene Plaintext p' dem von Alice gesendeten Plaintext p entspricht, also p' = p ist. Authenticity ist erreicht, wenn Bob sicher sein kann, dass die Nachricht von Alice stammt. Non-repudiation ist erreicht, wenn Alice nicht abstreiten kann, dass sie die Nachricht gesendet hat. Freshness ist erreicht, wenn Bob sicher sein kann, dass die Nachricht aktuell ist.

Attacktypen auf Kryptosysteme

Die Unterschiede zwischen den Attacken bestehen daraus, auf was der Attacker Zugriff hat.

Ciphertext-only attack Der Attacker kann vom Ciphertext alleine Rückschlüsse auf den Plaintext oder den verwendeten Schlüssel ziehen.

Chosen-ciphertext attack Der Attacker kann Ciphertexte generieren und diese vom System entschlüsseln lassen. Der Attacker bekommt entweder den zum gewählten Ciphertext gehörenden Plaintext (und kann daraus potenziell Rückschlüsse auf den verwendeten Schlüssel machen) oder er bekommt nur Teilinformationen, wie zum Beispiel "Die Entschlüsselung konnte / konnte nicht durchgeführt werden".

Known-plaintext attack Der Attacker kennt sowohl Teile des Plaintext als auch den dazugehörenden Ciphertext (oder zumindest Teile davon). Er kann daraus Rückschlüsse auf andere Plaintexte oder gar den Schlüssel machen.

Chosen-plaintext attack Der Attacker kann Plaintexte wählen, welche er vom System verschlüsseln lassen will. Er erhält dann den dazugehörenden Ciphertext und kann daraus Rückschlüsse auf andere Plaintexte oder gar den Schlüssel machen.

Brute-force attack Der Attacker probiert alle möglichen Schlüssel aus, bis er den richtigen gefunden hat. Dass er den richtigen gefunden hat, erkennt er daran, dass der erhaltene Plaintext sinnvoll erscheint.

Grundsätzlich können alle Verschlüsselungsalgorithmen mittels brute-force Attacken geknackt werden. Ein wichtiges Evaluationskriterium eines Verschlüsselungsalgorithmus ist also, wie lange eine solche brute-force Attacke im Durchschnitt benötigt. Dieser Anzahl sagt man auch kryptographischer Work Factor.

Kryptographischer Work Factor

Work Factor Durchschnittliche Anzahl Versuche, bis der richtige Schlüssel gefunden wird.

Der kryptographische Work Faktor kann als Mass der Stärke für eine Verschlüsselung herangezogen werden. Der Work Faktor bezeichnet die durchschnittliche Anzahl Versuche, bis man den richtigen Schlüssel bei einer Brute Force Attacke gefunden hat.

Ein Algorithmus hat dann einen genügend grossen Work Faktor wenn es unrealistisch ist, den richtigen Schlüssel per brute force zu erraten.

Einflussfaktoren auf den Work-Factor

- **Verschlüsselungsalgorithmus:** Je nach verwendetem Algorithmus ist die Berechnung des Work Faktors anders. Wir werden in den entsprechenden Kapiteln jeweils den dazugehörenden Work Faktor auflisten.
- **Schlüssellänge:** Grundsätzlich gilt: Je länger der Schlüssel, desto höher der Work Faktor
- **Zufälligkeit des Schlüssels:** Der Work Faktor von einem Algorithmus mit einem Schlüssel fixer länge, ist dann maximal, wenn alle Schlüssel gleich wahrscheinlich sind. Sind die Schlüssel nicht gleich wahrscheinlich, wird der Attacker zuerst die wahrscheinlicheren Schlüssel ausprobieren und so durchschnittlich weniger Schlüssel ausprobieren müssen, bis er den richtigen Schlüssel gefunden hat. Daher ist es sehr wichtig, dass die gewählten Schlüssel immer zufällig sind.

Berechnung des Work-Factors als Hilfe zur Berechnung erfindet man ein unrealistisch schnelles System, welches 10^9 Chips hat, welcher jeder in $10^{-12}s$ einen Schlüssel ausprobieren kann.

durchschn. Zeit je nach grösse Work-Factor (Schlüssellänge):

- 2^{64}: $1.8 \cdot 10^{-2}$ Sekunden
- 2^{96}: $7.9 \cdot 10^{7}$ s, ca. 2.5 Jahre
- 2^{128}: $3.4 \cdot 10^{17}$ s, ca. $10^{1}0$ Jahre
- 2^{256}: $1.2 \cdot 10^{56}$ s, ca. 10^{48} Jahre

Aus der Tabelle ersichtlich: Algorithmen mit Work-Factor $> 2^{128}$ als sicher betrachtet werden.

Den Work-Factor kann man auch in **bits** darstellen. Ein Work-Factor von 2^{128} entspricht einem Work-Factor von 128 bits. (2^n entspricht n bits)

Perfect Secrecy

Unter einem Algorithmus welcher Informationstheoretisch sicher ist, versteht man einen Algorithmus, bei dem man den ursprünglichen Plaintext nicht kennt, auch wenn man alle Schlüssel ausprobiert hat. Dazu sagt man auch der Algorithmus habe perfect secrecy. Aktuell gibt es nur einen einzigen Algorithmus, welcher diese Eigenschaft erfüllt, der One-Time-Pad (auch bekannt unter dem Namen Vernam Cipher).

One-Time-Pad

- **Voraussetzung:** Der Schlüssel ist komplett zufällig und genau gleich lang, wie die zu verschlüsselnde Nachricht.
- **Verschlüsselung:** Der Plaintext wird mit dem Schlüssel bitweise ver-xor-ed $c_j = p_j \oplus k_j \forall j \in \{1, \ldots, n\}$

Zur Erinnerung: Der XOR Operator ist definiert als:

p	k	$p \oplus k$
0	0	0
0	1	1
1	0	1
1	1	0

Perfect Secrecy Ein Verschlüsselungsalgorithmus hat perfect secrecy, wenn für alle möglichen Plaintexte p und Ciphertexte c und für alle Schlüssel k gilt:

$$P[p|c] = P[p] \tag{1}$$

Das heisst, dass die Wahrscheinlichkeit, dass ein bestimmter Plaintext p verschlüsselt wurde, gleich gross ist, wie die Wahrscheinlichkeit, dass ein beliebiger Plaintext p verschlüsselt wurde.

W enn ein Attacker jetzt alle Schlüssel ausprobieren würde, würde er neben vielen offensichtlich falschen Plaintexten auch viele plausible Plaintexte erhalten und wüsste daher nicht, welcher der plausiblen Plaintexte der korrekte ist.

Obwohl dies also genau das Verhalten ist, was wir möchten, kommt der One-Time-Pad nur in sehr seltenen Fällen zum Einsatz. Dies liegt daran, dass der Schlüssel

- genau gleich lang wie der zu verschlüsselnde Text sein muss
- komplett zufällig sein muss (vollständig zufällige Bitfolge)
- vorgängig geheim zwischen Sender und Empfänger ausgetauscht werden muss
- nur ein einziges mal verwendet werden darf (sonst ist er nicht mehr vollkommen zufällig und die ganze Sicherheit geht verloren)

Eigenschaften sicherer kryptographischer Algorithmen Da heute ausser dem One-Time-Pad kein Algorithmus bekannt ist, welcher als informationstheoretisch sicher gilt, werden folgende Eigenschaften für sichere Algorithmen definiert:

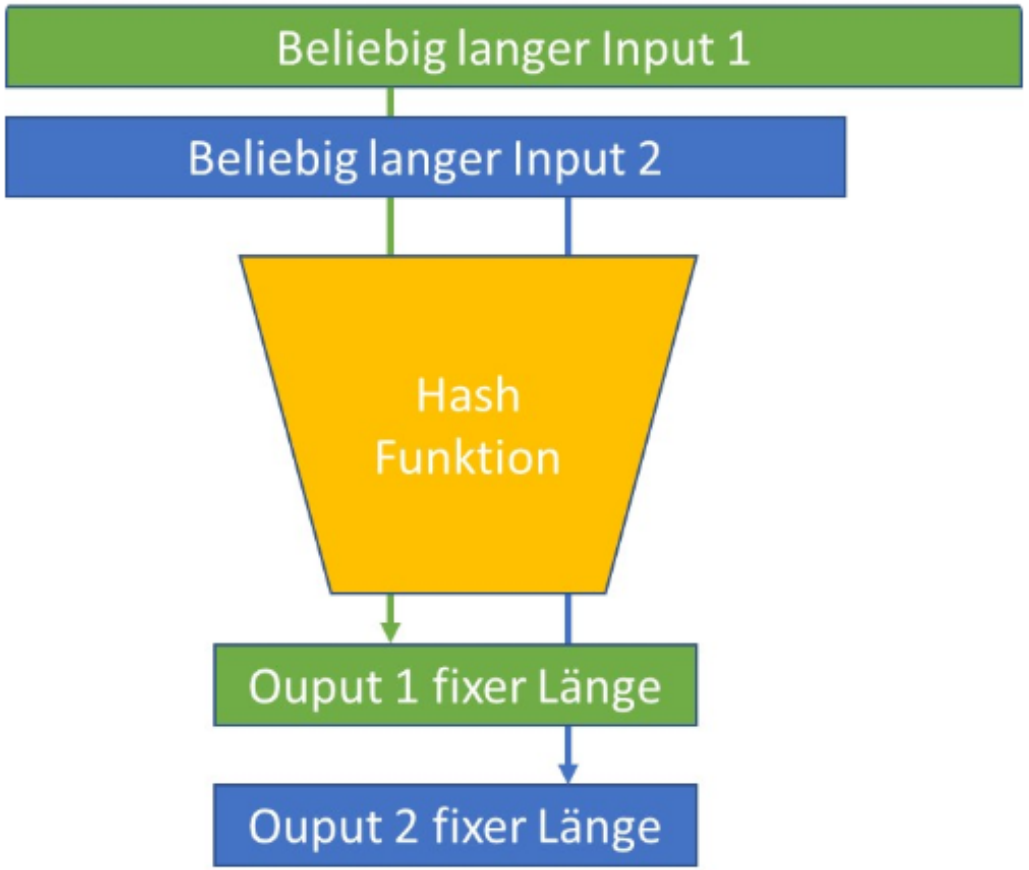
- Allgemein bekannt
- Keine Fehler bekannt
- Work Factor $> 2^{128}$

Daraus lässt sich auch ableiten, dass wir als Noobs keine kryptographischen Algorithmen selber entwickeln/implementieren sollten. Stattdessen soll auf standardisierte Algorithmen zurückgegriffen werden. (Öffentlich verfügbare Libraries)

Kryptographische Hash Funktionen

Kryptographische Hash Funktion ist eine mathematische Funktion mit folgenden Eigenschaften:

- aus einem beliebig langen Input wird Output mit konstanter Länge generiert
- es gibt keine Möglichkeit aus dem Output den Input wieder herzuleiten
- unterschiedliche Inputs ergeben mit sehr hoher Wahrscheinlichkeit völlig unterschiedliche Outputs, auch wenn sich die Inputs nur wenig unterscheiden
- es ist nicht möglich innert nützlicher Zeit zwei unterschiedliche Inputs zu generieren, welche denselben Output haben



Work Factor Da bei Hash Funktionen der Output weniger lang ist als der Input gibt es keinen Algorithmus, welcher für alle Inputs und Outputs die Eigenschaften 3 und 4 erreicht. Das heisst, es wird immer verschiedene Inputs geben, welche denselben Output generieren. Mit dem Work Faktor kann angegeben werden, wie gross der Aufwand ist, um diese Hash Collisions zu berechnen.
Der Work Faktor in bits einer kryptographischen Funktion entspricht der Hälfte der bits des generierten Outputs. Auch hier gilt: Für eine sichere Hash Funktion sollte der Work Faktor mindestens 128 bit betragen.

Digital Certificates

Introduction to Digital Certificates

Digital Certificates

A digital certificate is a signed statement by a trusted third party (Certificate Authority) that a certain public key belongs to a certain entity (person, organization, or system). Certificates are used to authenticate public keys, not the entities themselves.

Authentication of Public Keys

The fundamental problem certificates solve is how to reliably distribute public keys in an insecure network:

- Without certificates, it's difficult to know if a public key truly belongs to the claimed entity
- Certificates bind identities to public keys through the endorsement of a trusted third party
- The binding is authenticated through cryptographic signatures

Certificate Types

Different certificate types serve different purposes:

- **TLS Certificates**:
  - Domain Validation (DV) - Validates domain ownership only
  - Organization Validation (OV) - Validates domain ownership and organization details
  - Extended Validation (EV) - Provides the highest level of validation
- **Code signing certificates** - Authenticate software and files
- **Client certificates** - Identify individual users or machines

X.509 Certificate Structure

X.509 Standard

X.509 is the dominant certificate standard, developed by the International Telecommunication Union (ITU). X.509 certificates are described in ASN.1 notation and typically encoded using DER (Distinguished Encoding Rules).

X.509 Certificate Fields

Key components of an X.509 certificate include:

- **Version** - Certificate format version (v1, v2, or v3)
- **Serial Number** - Unique identifier assigned by the issuing CA
- **Signature Algorithm ID** - Algorithm used to sign the certificate
- **Issuer Name** - X.500 Distinguished Name of the issuing CA
- **Validity Period** - Start and end dates for certificate validity
- **Subject Name** - X.500 Distinguished Name of the certificate owner
- **Subject Public Key Info** - The public key and algorithm identifier
- **Extensions** - Additional fields (e.g., key usage, alternative names)
- **Certificate Signature** - The digital signature created by the CA

Certificate Chains and Trust

Certificate Chains

Certificates form chains of trust:

- **End entity certificates** - Used by servers, clients, or devices
- **Intermediate CA certificates** - Issued by root CAs to intermediate CAs
- **Root CA certificates** - Self-signed certificates from trusted root authorities

To verify a certificate, each certificate in the chain must be validated up to a trusted root.

Certificate Validation Process

To validate a certificate chain of length n with certificates C[1], ..., C[n], where C[1] is the end entity certificate:

1. Check that C[1].name matches the expected entity name
2. For i = 1 to n - 1:
   - Check that C[i].issuer = C[i+1].subject
   - Verify C[i].signature using C[i+1].publicKey
3. Locate root cert R with C[n].issuer = R.subject
4. Verify C[n].signature using R.publicKey
5. Verify R.signature using R.publicKey (self-verification)

Root Certificate Trust

Root certificates are the foundation of trust in PKI:

- Root certificates are pre-installed in operating systems and browsers
- They're the ultimate trust anchors that enable the validation of all other certificates
- The security of PKI depends on the protection of root CA private keys

Certificate Revocation

Certificate Revocation

Certificate revocation is the process of invalidating a certificate before its expiration date, typically due to:

- Compromise of the associated private key
- Change in the certificate owner's status
- Cessation of operation by the certificate owner

Certificate Revocation Mechanisms

Several mechanisms exist for checking certificate revocation status:

- **Certificate Revocation Lists (CRLs)** - Lists of revoked certificates published by CAs
- **Online Certificate Status Protocol (OCSP)** - Protocol for real-time certificate status checking
- **OCSP Stapling** - Server includes pre-fetched OCSP response during TLS handshake
- **OCSP Must-Staple** - Requires OCSP stapling for a certificate to be considered valid
- **Browser-Summarized CRLs** - Browser vendors compile and compress CRLs for distribution to browser instances

Checking Certificate Revocation

Using CRLs

- Retrieve CRL from the URL specified in the certificate
- Verify CRL signature using the issuing CA's public key
- Check if the certificate's serial number appears in the CRL

Using OCSP

- Send HTTP request to the OCSP responder URL in the certificate
- Include certificate serial number in the request
- Receive signed response indicating certificate status (good, revoked, unknown)
- Verify signature on the response using the issuing CA's public key

Using OCSP Stapling

- Server periodically requests OCSP status from the CA
- Server caches the signed OCSP response
- Server includes the response in the TLS handshake
- Client verifies the OCSP response signature using the CA's public key

Certificate Transparency

Certificate Transparency

Certificate Transparency (CT) is a framework designed to detect and prevent the fraudulent issuance of certificates:

- Publicly auditable logs of all issued certificates
- Monitors check logs for suspicious certificates
- Browsers require evidence that certificates are logged
- Helps detect malicious or mistakenly issued certificates

Let's Encrypt, a free and automated Certificate Authority, has become the largest CA with over 234 million active certificates. It uses the ACME (Automatic Certificate Management Environment) protocol to automate the validation, issuance, and renewal of certificates. Let's Encrypt provides only Domain Validation (DV) certificates with a validity period of 90 days, encouraging frequent renewal and automation.

Transport Layer Security (TLS)

Introduction to TLS

Transport Layer Security (TLS)

TLS is a cryptographic protocol designed to provide secure communication over a computer network. It provides:

- **Confidentiality** - Protection against eavesdropping
- **Integrity** - Detection of message tampering
- **Authentication** - Verification of communicating parties
- **Protection against replay attacks** - Prevention of message reuse

TLS is the successor to Secure Sockets Layer (SSL).

TLS Evolution

TLS has evolved through several versions:

- **SSL 2.0** (1995) - First publicly released version (deprecated)
- **SSL 3.0** (1996) - Fixed major security issues in 2.0 (deprecated)
- **TLS 1.0** (1999, RFC 2246) - First IETF standard version (legacy)
- **TLS 1.1** (2006, RFC 4346) - Added protection against CBC attacks (legacy)
- **TLS 1.2** (2008, RFC 5246) - Added authenticated encryption, improved flexibility (current)
- **TLS 1.3** (2018, RFC 8446) - Major redesign with improved security and performance (current)

TLS Building Blocks

TLS 1.3 relies on several cryptographic building blocks:

- Strong block ciphers (AES, ChaCha20)
- Authenticated encryption modes (GCM, CCM, Poly1305)
- Diffie-Hellman key exchange (including elliptic curve variants)
- Public key authentication with certificates
- Cryptographic hash functions for key derivation (HKDF)

Cipher Suites

A cipher suite is a combination of cryptographic algorithms used in TLS. In TLS 1.3, cipher suites are simplified and specify only symmetric encryption algorithms:

- **TLS_AES_128_GCM_SHA256** - AES-128 in GCM mode with SHA-256 for key derivation
- **TLS_AES_256_GCM_SHA384** - AES-256 in GCM mode with SHA-384 for key derivation
- **TLS_CHACHA20_POLY1305_SHA256** - ChaCha20 with Poly1305 and SHA-256

The key exchange algorithms and certificate verification methods are negotiated separately.

TLS Protocol Stack

TLS Protocol Layers

TLS consists of multiple protocol layers:

- **TLS Record Protocol** - The base layer that defines packet format
- **Handshake Protocol** - Negotiates cryptographic parameters and authenticates parties
- **Alert Protocol** - Communicates errors and warnings
- **Application Data Protocol** - Transmits encrypted application data
- **Change Cipher Spec Protocol** - (Deprecated in TLS 1.3, retained for backward compatibility)

TLS Protocol Phases

TLS operates in three distinct phases:

- **Handshake** - Establishes cryptographic parameters and authenticates parties
- **Data Exchange** - Transfers protected application data
- **Connection Teardown** - Securely terminates the connection

TLS 1.3 Handshake

TLS 1.3 Handshake Goals

The TLS 1.3 handshake accomplishes several goals:

- Negotiate cryptographic algorithms
- Perform Diffie-Hellman key exchange
- Generate handshake keys for encryption of subsequent messages
- Authenticate the server (and optionally the client)
- Verify message integrity during the handshake
- Generate keys for data encryption

TLS 1.3 Handshake Process

Initial Exchange

- Client sends ClientHello message with:
  - Supported TLS versions
  - Supported cipher suites
  - Supported key exchange groups
  - Client's key shares for preferred groups
- Server responds with ServerHello message containing:
  - Selected TLS version
  - Selected cipher suite
  - Server's key share for selected group

Key Derivation

- Both parties compute shared secret using Diffie-Hellman
- Handshake keys are derived using HKDF with the shared secret
- All subsequent handshake messages are encrypted

Server Authentication

- Server sends Certificate message with certificate chain
- Server sends CertificateVerify with signature over handshake transcript
- Server sends Finished message with MAC over handshake transcript

Client Verification

- Client validates server certificate
- Client verifies server's signature in CertificateVerify
- Client verifies server's Finished message
- Client sends its own Finished message

Application Data

- Both parties derive application data keys
- Encrypted application data can now be exchanged

TLS 1.3 Improvements

TLS 1.3 provides several improvements over previous versions:

- **Reduced Handshake Latency** - Requires only one round-trip (1-RTT) instead of two
- **Zero Round-Trip Time (0-RTT)** - Optional resumption mode for repeat connections
- **Simplified Cryptography** - Removed support for weak algorithms
- **Forward Secrecy** - Mandatory for all cipher suites
- **Encrypted Handshake** - Better protection against traffic analysis

TLS Data Exchange

TLS Record Protection

Application data is protected within TLS records:

- Data is fragmented into manageable chunks
- Each fragment is encrypted and authenticated using AEAD ciphers
- A sequence number is included in the authentication to prevent replay attacks
- TLS records include headers that are not encrypted but are authenticated

TLS Session Resumption

TLS Session Resumption

Session resumption allows clients to reconnect to servers more efficiently:

- Avoids the computational cost of public key operations
- Reduces the number of round-trips required
- In TLS 1.3, uses pre-shared keys (PSKs) derived from previous connections
- Server provides NewSessionTicket messages containing ticket data
- Client can use this ticket in future connections

TLS Session Teardown

TLS Connection Termination

TLS provides a secure mechanism for terminating connections:

- Uses close_notify alerts to signal completion of data transmission
- Prevents truncation attacks where an attacker prematurely terminates a connection
- Both parties should send close_notify alerts before closing the underlying TCP connection

Datagram Transport Layer Security (DTLS)

Datagram Transport Layer Security (DTLS)

DTLS is an adaptation of TLS for datagram transport protocols like UDP:

- Based on TLS with minimal modifications for unreliable transport
- Uses explicit sequence numbers to handle packet reordering
- Implements message loss detection and retransmission for handshake messages
- Provides optional replay detection for application data
- Current version is DTLS 1.3 (RFC 9147)

A web browser connecting to a secure website (https://example.com) initiates a TLS 1.3 handshake. The browser sends a ClientHello message including support for TLS 1.3 and key shares for X25519 and P-256 curves. The server selects TLS 1.3 with X25519 and responds with a ServerHello message. Both compute a shared secret and derive handshake keys. The server sends its certificate chain, a signature over the handshake, and a Finished message. After verifying these, the browser sends its own Finished message. Both derive application data keys and begin encrypted communication, completing the handshake in just one round-trip.

TLS 1.3 Session Resumption

```python
# Client-side pseudocode for TLS 1.3 session resumption
def resume_tls_session(server, port, psk_identity, psk):
    # Create ClientHello with PSK
    client_hello = create_client_hello(
        supported_versions=[TLS_1_3],
        cipher_suites=[TLS_AES_128_GCM_SHA256, TLS_CHACHA20_POLY1305_SHA256],
        psk_identity=psk_identity
    )

    # Send ClientHello to server
    send_message(server, port, client_hello)

    # Receive ServerHello
    server_hello = receive_message(server, port)

    if server_hello.accepts_psk:
        # Derive traffic keys from PSK
        traffic_keys = derive_keys_from_psk(psk, client_hello, server_hello)

        # Send Finished message
        finished = create_finished_message(traffic_keys.client_handshake_key)
        send_encrypted_message(server, port, finished,
            traffic_keys.client_handshake_key)

        # Receive server Finished
        server_finished = receive_encrypted_message(server, port,
            traffic_keys.server_handshake_key)

        # Ready for application data
        return Connection(traffic_keys.application_keys)
    else:
        # Fall back to full handshake
        return perform_full_handshake(server, port, client_hello, server_hello)
```

Layer 2 Security

Introduction to Secure Communication

Security at Different OSI Layers

Secure communication can be implemented at different layers of the OSI model:

- **Physical Layer (Layer 1)** - Quantum cryptography, physical isolation
- **Data Link Layer (Layer 2)** - IEEE 802.1X, WLAN security (WEP, WPA, WPA2, WPA3), MACsec
- **Network Layer (Layer 3)** - IPsec
- **Transport Layer (Layer 4)** - TLS, QUIC
- **Application Layer (Layer 7)** - PGP, S/MIME, Signal, WhatsApp

Layer Selection Tradeoffs

The choice of which layer to implement security has various tradeoffs:

- **Higher Layer Security**:
  - Easier to deploy (often included in applications)
  - Typically provides end-to-end protection
  - Less generally applicable (often optimized for specific applications)
- **Lower Layer Security**:
  - More difficult to deploy (may require kernel, router, or switch modifications)
  - Often provides only hop-to-hop protection at layer 2
  - More generally applicable (protects all protocols above it)

Extensible Authentication Protocol (EAP)

Extensible Authentication Protocol (EAP)

EAP is a framework for authentication that supports multiple authentication methods:

- Provides a flexible authentication mechanism
- Designed to be transport-independent (can run over various protocols)
- Allows network access control without knowledge of each individual user
- Supports numerous authentication methods via different EAP types

EAP Packet Format

EAP packets have a simple structure:

```
struct eap_packet {
    uint8_t code;        // Request (1), Response (2), Success (3), Failure (4)
    uint8_t id;          // Used to match requests and responses
    uint16_t length;     // Total length of the EAP packet
    uint8_t type;        // When code=1,2: authentication method type
    uint8_t data[];      // Type-specific data
};
```

Common EAP types include:

- Type 4: MD5-Challenge (insecure!)
- Type 13: EAP-TLS (recommended)
- Type 21: EAP-TTLS
- Type 25: PEAP (Protected EAP)

EAP Authentication Methods

EAP supports numerous authentication methods with varying security properties:

- **EAP-MD5** - Simple challenge-response mechanism (insecure, no mutual authentication)
- **EAP-TLS** - Uses TLS protocol with client certificates (strong security)
- **EAP-TTLS** - Tunneled TLS without requiring client certificates
- **PEAP** - Microsoft's Protected EAP, similar to EAP-TTLS
- **EAP-FAST** - Cisco's Flexible Authentication via Secure Tunneling

Current recommendation is to use EAP-TLS when possible.

IEEE 802.1X Port-Based Access Control

IEEE 802.1X

IEEE 802.1X is a standard for port-based network access control:

- Controls access to a network by authenticating devices before allowing network access
- Prevents unauthorized access through publicly accessible network ports
- Uses EAP for authentication
- Provides centralized VLAN assignment based on authentication

802.1X Components

IEEE 802.1X involves three key components:

- **Supplicant** - The client device requesting network access
- **Authenticator** - The network device (switch, access point) controlling port access
- **Authentication Server** - Typically a RADIUS server that validates user credentials

IEEE 802.1X Authentication Process

Initial Connection

- Client connects to a port on the switch/access point
- Port is initially blocked for all traffic except EAP
- Communication begins with EAPOL (EAP over LAN) protocol

Authentication Exchange

- Supplicant sends EAPOL-Start message or authenticator sends EAP-Request Identity
- Supplicant responds with identity
- Authenticator forwards identity to authentication server using RADIUS
- Authentication server and supplicant perform EAP authentication exchange
- Authenticator relays EAP messages between supplicant and authentication server

Authorization

- Authentication server sends Access-Accept or Access-Reject message
- On success, authenticator can include VLAN assignment and other parameters
- Authenticator opens port for network access based on authorization parameters

In an IEEE 802.1X deployment, a user connects their laptop to a corporate network. The switch port is initially closed, blocking all non-authentication traffic. The user's laptop (supplicant) provides credentials through EAP-TLS, and the switch (authenticator) forwards these to a RADIUS server. Upon successful authentication, the RADIUS server instructs the switch to place the user's connection in the appropriate VLAN based on their role. The port is then opened for normal network traffic.

Security Limitations of 802.1X

While 802.1X provides significant security benefits, it has limitations:

- Man-in-the-middle attacks are possible when an attacker inserts themselves between a legitimate device and the switch
- An attacker could monitor traffic by creating a bridge between the legitimate device and the switch
- Once initial authentication is complete, the ongoing session is not continuously verified
- These limitations are addressed in 802.1X-2010 with the use of MACsec

MACsec (IEEE 802.1AE)

MACsec

IEEE 802.1AE (Media Access Control Security) provides:

- Data confidentiality, integrity, and authenticity on layer 2 (Ethernet)
- Protection for all data including DHCP, ARP, and higher layer protocols
- Security for both physical and virtual links between devices
- Line-rate encryption in hardware implementations

MACsec Operation

MACsec operates by:

- Encrypting the payload of Ethernet frames between adjacent devices
- Adding a SecTAG to identify the keys and provide packet numbering
- Adding an Integrity Check Value (ICV) based on GCM-AES
- Providing hop-by-hop protection where each device on the path decrypts and re-encrypts the data

WLAN Security

WLAN Security Challenges

Wireless networks face unique security challenges:

- No physical cable requirement for packet sniffing
- Signals extend beyond physical boundaries
- Difficult to control who can attempt to connect
- Need for strong encryption and authentication

Evolution of WLAN Security

WLAN security has evolved through several generations:

- **WEP (Wired Equivalent Privacy)** - Original security standard with major design flaws
- **WPA (Wi-Fi Protected Access)** - Intermediate solution while developing 802.11i
- **WPA2 (IEEE 802.11i)** - Current standard with strong security
- **WPA3** - Newest standard with improved security measures

WEP Security

Wired Equivalent Privacy (WEP)

WEP was the original security standard for 802.11 networks:

- Shared key between all clients and the access point
- Uses RC4 stream cipher with 40 or 104-bit keys
- Uses a 24-bit Initialization Vector (IV)
- Includes a CRC-32 checksum for integrity checking

WEP Security Flaws

WEP has several critical design flaws:

- Small IV space (24 bits) leads to inevitable IV reuse
- Weak key scheduling algorithm in RC4
- CRC-32 is not cryptographically secure for integrity protection
- No protection against bit-flipping attacks
- No replay protection
- No per-user/per-session keys

To demonstrate the weakness of WEP, tools like Aircrack-ng can typically crack a WEP key after capturing approximately one million encrypted frames, a process that takes just minutes to hours depending on network traffic volume. The attack exploits weaknesses in the RC4 key scheduling algorithm, using the predictable IVs to determine key bits.

WPA and WPA2

Wi-Fi Protected Access (WPA)

WPA was developed as an interim solution while waiting for 802.11i (WPA2):

- Introduced the Temporal Key Integrity Protocol (TKIP)
- Provided per-packet key mixing
- Added message integrity code (Michael)
- Implemented a frame counter to prevent replay attacks
- Available in personal (PSK) and enterprise (802.1X) modes

WPA2 (IEEE 802.11i)

WPA2 is the complete implementation of the IEEE 802.11i standard:

- Introduced Counter Mode with CBC-MAC Protocol (CCMP)
- Based on AES encryption (replacing RC4)
- Provides stronger authentication, integrity, and confidentiality
- Retained TKIP only for backward compatibility
- Available in personal (PSK) and enterprise (802.1X) modes

WPA/WPA2 Operation

WPA and WPA2 operate in two distinct phases:

- **Authentication Phase**:
  - WPA-Personal: Uses pre-shared key (PSK)
  - WPA-Enterprise: Uses 802.1X and EAP with a RADIUS server
- **Key Exchange Phase**:
  - Establishes pairwise transient keys for unicast traffic
  - Establishes group keys for broadcast/multicast traffic
  - Implements key rotation to prevent attacks

WPA3

WPA3

WPA3 is the latest Wi-Fi security standard (released in 2018):

- Introduces Simultaneous Authentication of Equals (SAE) to replace WPA2-Personal PSK
- Provides forward secrecy
- Protects against offline dictionary attacks
- Includes enhanced protection for enterprise networks
- Improves encryption for public networks with Opportunistic Wireless Encryption (OWE)

Best Practices for WLAN Security

Network Design

- Use WPA2 or WPA3 with CCMP/AES encryption
- Implement Enterprise mode with 802.1X authentication when possible
- If using Personal mode, use complex pre-shared keys (PSKs)
- Use separate SSIDs for different security levels

Authentication

- Use EAP-TLS for strongest security
- If client certificates aren't feasible, use PEAP or EAP-TTLS
- Avoid EAP-MD5 and other weak authentication methods

Management

- Change default administrator credentials
- Disable unnecessary services like WPS
- Update firmware regularly
- Use a strong encryption passphrase

TKIP vs CCMP Comparison

```
// TKIP (Temporal Key Integrity Protocol)
process_with_tkip(packet, key) {
    // For each packet:
    per_packet_key = mix_key(key, IV, MAC_address);  // Key mixing function
    // Use RC4 for encryption
    encrypted_data = rc4_encrypt(per_packet_key, data);
    // Calculate Michael MIC for integrity
    mic = michael_algorithm(key, header, data);
    // Output packet with IV, encrypted data, and MIC
    return format_packet(IV, encrypted_data, mic);
}

// CCMP (Counter Mode with CBC-MAC Protocol)
process_with_ccmp(packet, key) {
    // For each packet:
    // Construct nonce from packet number, addresses, and priority
    nonce = construct_nonce(packet_number, addr, priority);
    // Use AES in counter mode for encryption
    encrypted_data = aes_ctr_encrypt(key, nonce, data);
    // Calculate CBC-MAC for authentication/integrity
    mac = aes_cbc_mac(key, header, data);
    // Output packet with packet number, encrypted data, and MAC
    return format_packet(packet_number, encrypted_data, mac);
}
```

Network Security

Network Segmentation

Network Segmentation

Network segmentation is the practice of dividing a computer network into smaller, isolated segments:

- Increases operational performance by containing traffic
- Limits damage from cyber attacks to specific segments
- Protects vulnerable devices by isolating them
- Reduces the scope of compliance requirements
- Protects against insider threats

Typical Network Segments

Common network segments in organizations include:

- **DMZ (Demilitarized Zone)** - Hosts public-facing services
- **Server Network** - Hosts internal servers and applications
- **Client Network** - Hosts employee workstations
- **Guest Network** - Hosts visitor devices with limited access
- **IoT Network** - Hosts Internet of Things devices
- **Industrial Network** - Hosts industrial control systems
- **Admin Network** - Hosts administrative systems and tools

Packet Filtering Firewalls

Packet Filtering Firewalls

Packet filtering firewalls control traffic flow between network segments:

- Examine packet headers to make filtering decisions
- Apply rules based on source/destination addresses, ports, and protocols
- Operate at the network and transport layers of the OSI model
- Form the foundation of network security architecture

Firewall Rule Management

Effective firewall management requires structured processes:

- **New Request** - Submission, approval, design, testing, deployment, validation
- **Operation** - Monitoring, audits, reports
- **Recertification** - Regular review of existing rules
- **Decommissioning** - Removal of unnecessary rules

Without proper management processes, firewall rule sets tend to grow uncontrolled and become difficult to understand.

Firewall Benefits and Limitations

Firewalls provide significant security benefits but have limitations:

- **Benefits**:
  - Block unwanted traffic before it enters protected networks
  - Centralize access control
  - Hide internal network structure
- **Limitations**:
  - Primarily perimeter protection (assumes threats are external)
  - Cannot prevent internal threats
  - Basic packet filtering cannot detect application-layer attacks
  - Cannot protect against allowed protocols misuse

Next Generation Firewalls (NGFW)

Next Generation Firewalls

NGFWs extend traditional firewall capabilities with additional features:

- **Deep Packet Inspection** - Examines packet contents, not just headers
- **Application Awareness** - Identifies and controls traffic by application
- **Intrusion Prevention** - Detects and blocks network attacks
- **Antivirus/Anti-malware** - Scans traffic for malicious content
- **Sandboxing** - Executes suspicious files in isolated environments
- **Threat Intelligence** - Incorporates external information about threats

Microsegmentation

Microsegmentation

Microsegmentation takes network segmentation to a more granular level:

- Segments as small as individual workloads or applications
- Implemented through network virtualization or host-based firewalls
- Limits lateral movement within a traditional network segment
- Reduces the attack surface and blast radius
- Balances security with management complexity

Zero Trust Architecture

Zero Trust is a security model that assumes no implicit trust within or outside the network:

- **Core Principles**:
  - Verify explicitly - Always authenticate and authorize
  - Use least privilege access - Provide minimal necessary access
  - Assume breach - Operate as if attackers are already present
- **Components**:
  - Policy Enforcement Point (PEP) - Enforces access decisions
  - Policy Decision Point (PDP) - Makes access decisions
  - Policy Administration Point (PAP) - Manages policies

Zero Trust Challenges

While Zero Trust offers significant security benefits, it presents several challenges:

- Single points of failure in policy enforcement
- Potential for accidental or malicious misconfigurations
- Vulnerability to denial-of-service attacks
- Difficulties with encrypted traffic visibility
- Lack of protection against credential theft

Host-Based Security

Host-Based Firewalls

Host-based firewalls provide an additional layer of protection:

- Operate directly on individual devices
- Protect against threats that bypass network firewalls
- Filter traffic based on local policies
- Examples include Windows Defender Firewall, Linux nftables, and macOS firewall

Endpoint Detection and Response (EDR)

EDR systems combine several capabilities to secure endpoints:

- **Monitoring**:
  - Anomaly detection
  - Vulnerability scanning
  - Integrity checks
- **Protection**:
  - Host-based firewall
  - Antivirus/anti-malware
  - Application control
- **Investigation and Response**:
  - Device isolation
  - User session termination
  - Change rollback
  - Evidence collection

Application Protection

Web Application Firewalls (WAF)

WAFs protect web applications from attacks:

- Examine HTTP/HTTPS traffic at the application layer
- Protect against OWASP Top 10 vulnerabilities:
  - Cross-Site Scripting (XSS)
  - SQL Injection
  - Cross-Site Request Forgery (CSRF)
  - Others
- Require TLS termination to inspect encrypted traffic
- Deploy in front of web servers

Secure Web Gateways and Cloud Security

Secure Web Gateway (SWG)

SWGs protect users from web-based threats:

- URL filtering to block malicious sites
- Data Loss Prevention (DLP) to prevent sensitive data exfiltration
- TLS inspection to examine encrypted traffic
- User and application control

Cloud Access Security Broker (CASB)

CASBs provide security controls for cloud services:

- **Shadow IT discovery** - Identify unauthorized cloud service usage
- **Cloud usage control** - Set access rights to cloud services
- **Data leakage prevention** - Control data sharing in cloud services
- **Anomaly detection** - Alert on unusual behavior patterns
- **Implementation methods**:
  - API scanning - Directly interfaces with cloud providers
  - Forward proxy - Controls outbound access to cloud services
  - Reverse proxy - Intermediates between users and cloud services

Network Detection and Response

Network Detection and Response (NDR)

NDR systems monitor network traffic to detect and respond to threats:

- Establish baseline network behavior
- Detect anomalies that may indicate attacks
- Analyze potential incidents to identify true positives
- Automatically respond to confirmed threats

Security Information and Event Management (SIEM)

SIEM systems collect and analyze security events:

- Aggregate logs from multiple sources
- Correlate events to identify attack patterns
- Provide a centralized dashboard for security monitoring
- Generate reports for compliance and security analysis

Security Orchestration Automation and Response (SOAR)

SOAR platforms extend SIEM capabilities:

- Automate security response with playbooks
- Integrate with multiple security tools
- Orchestrate complex security workflows
- Enhance incident response with automation

Linux Firewall with nftables

nftables Framework

nftables is a packet classification and filtering framework in Linux:

- Replaces the legacy iptables framework
- Part of the Linux kernel since version 2.4
- Provides packet filtering, network address translation, and packet mangling
- Configured using the nft command-line tool

nftables Architecture

nftables uses a hierarchical structure:

- **Tables** - Group chains for a specific packet type (address family)
- **Chains** - Group rules and attach to hooks in the network stack
- **Rules** - Define matching criteria and actions for packets
- **Expressions** - Match packet properties (addresses, ports, etc.)
- **Actions** - Determine what happens to matching packets (accept, drop, reject, etc.)

Basic nftables Commands

```
# Create a table
nft add table inet filter

# Create a chain in the table
nft add chain inet filter input { type filter hook input priority 0 \; policy drop \;
    }

# Add a rule to the chain
nft add rule inet filter input tcp dport 22 accept

# List the ruleset
nft list ruleset

# Delete a rule (using its handle)
nft delete rule inet filter input handle 4

# Flush a table (delete all chains and rules)
nft flush table inet filter
```

Creating a Basic Firewall with nftables

Setup Tables and Chains

- Create table for IPv4 and IPv6 traffic
- Create input, forward, and output chains
- Set default policies (usually drop for input and forward, accept for output)

Add Basic Rules

- Allow established and related connections
- Allow loopback traffic
- Allow specific services (SSH, HTTP, etc.)
- Allow ICMP/ICMPv6 for diagnostics

Add Protection Rules

- Drop invalid packets
- Implement rate limiting for certain traffic
- Log suspicious traffic

Test and Save

- Test connectivity from various sources
- Save ruleset to persist across reboots
- Implement monitoring to detect issues

Port Scanning

Port Scanning

Port scanning is a technique to discover available services on a network:

- Identifies open ports on target systems
- Helps determine running services and potential vulnerabilities
- Used by both attackers for reconnaissance and administrators for security testing
- Most commonly performed using tools like nmap

Basic Port Scanning with nmap

```
# Scan a single host for common ports
nmap 192.168.1.1

# Scan specific ports
nmap -p 22,80,443 192.168.1.1

# Scan a range of ports
nmap -p 1-1000 192.168.1.1

# Scan an entire subnet
nmap 192.168.1.0/24

# Perform a service version scan
nmap -sV 192.168.1.1

# Perform an OS detection scan
nmap -O 192.168.1.1

# Perform a comprehensive scan
nmap -A 192.168.1.1

# Scan without ping (useful if ICMP is blocked)
nmap -Pn 192.168.1.1
```

A security administrator needs to verify that a newly implemented firewall is correctly filtering traffic between network segments. They use nmap to scan hosts in the protected segment from an external testing machine:
`nmap -Pn -sT -p 1-1024 10.0.1.10`
The scan reveals that only ports 80 (HTTP) and 443 (HTTPS) are accessible, while all other ports show as "filtered,"indicating that the firewall is properly blocking the traffic. Additional scans from authorized internal segments confirm that administrative ports like 22 (SSH) are only accessible from the management network.

Virtual Private Networks

Introduction to VPNs

Virtual Private Network (VPN)

A VPN is a private network created within a public network infrastructure:

- **Private** - External parties cannot read or modify transmitted data
- **Virtual** - Privacy is achieved through cryptography, not dedicated links
- Creates an encrypted tunnel between endpoints
- Allows secure access to resources across untrusted networks

VPN Use Cases

Common VPN use cases include:

- Connecting remote offices or branches to a main corporate network
- Allowing partner organizations limited access to internal resources
- Enabling remote employees to securely access company resources
- Protecting privacy when using public Wi-Fi networks
- Bypassing geographical restrictions on content

VPN Core Concepts

Key VPN characteristics include:

- VPNs connect networks (or a host to a network), not just individual hosts
- VPN endpoints (gateways) establish and maintain the secure tunnel
- Internal hosts typically require no special configuration
- Traffic is encrypted between VPN endpoints
- VPNs often hide internal addressing with NAT

VPN Protocols

VPN Protocol Types

Several protocols are commonly used to implement VPNs:

- **IPsec** - IP Security protocol suite operating at the network layer
- **OpenVPN** - SSL/TLS-based solution running at the application layer
- **WireGuard** - Modern, high-performance VPN protocol
- **Proprietary protocols** - Vendor-specific implementations

IPsec VPNs

Internet Protocol Security (IPsec)

IPsec is an IETF standard protocol suite for secure IP communications:

- Operates at the network layer (Layer 3)
- Provides authentication, integrity, and confidentiality
- Includes key exchange mechanism (Internet Key Exchange, IKE)
- Supported by most operating systems and network devices

IPsec Components

IPsec consists of several components:

- **Authentication Header (AH)** - Provides authentication and integrity (rarely used)
- **Encapsulating Security Payload (ESP)** - Provides encryption, authentication, and integrity
- **Internet Key Exchange (IKE)** - Handles key exchange and security association negotiation
- **Security Associations (SA)** - Parameters for secure communication

IPsec vs. TLS Comparison

IPsec and TLS differ in several key aspects:

- **Layer** - IPsec works at the network layer, TLS at the transport/application layer
- **Protection scope** - IPsec protects all IP traffic between hosts, TLS protects specific application connections
- **Implementation** - IPsec requires kernel integration, TLS runs in user space
- **Configuration** - IPsec typically requires more complex configuration

IPsec Modes

IPsec operates in two primary modes:

- **Transport Mode**:
  - Protects the payload of the IP packet
  - Original IP header remains intact
  - Typically used for host-to-host communications
- **Tunnel Mode**:
  - Protects the entire original IP packet
  - Encapsulates the original packet in a new IP packet
  - Typically used for network-to-network (gateway-to-gateway) VPNs
  - Hides internal IP addressing from eavesdroppers

IPsec ESP Packet Structure in Tunnel Mode

```c
/* Original IP Packet */
struct original_ip_packet {
    struct ip_header original_ip_header;
    struct tcp_header tcp_header;        // Or other protocol
    uint8_t data[VARIABLE_LENGTH];
};

/* IPsec ESP Packet in Tunnel Mode */
struct ipsec_esp_tunnel_packet {
    struct ip_header new_ip_header;      // New header for routing between gateways
    struct esp_header {                  // ESP Header
        uint32_t spi;                    // Security Parameters Index
        uint32_t sequence_number;        // Sequence number for replay protection
    } esp_header;

    /* Encrypted portion begins */
    struct original_ip_packet original_packet;  // The entire original packet

    uint8_t padding[0-255];              // Padding to block size
    uint8_t pad_length;                  // Number of padding bytes
    uint8_t next_header;                 // Type of the encapsulated header
    /* Encrypted portion ends */

    uint8_t integrity_check_value[VARIABLE_LENGTH];  // Authentication data
};
```

IPsec VPN Setup Process

Phase 1: IKE Security Association

- Exchange cryptographic algorithms and parameters
- Authenticate peers (pre-shared keys or certificates)
- Establish a secure channel for Phase 2
- Generate keying material

Phase 2: IPsec Security Association

- Negotiate IPsec security parameters
- Establish IPsec SAs for data protection
- Define traffic selectors (which traffic to protect)

Data Exchange

- Encrypt and authenticate data according to SA parameters
- Encapsulate traffic based on tunnel or transport mode
- Process incoming encrypted traffic

SA Maintenance

- Renew SAs before expiration
- Handle dead peer detection
- Manage rekeying

OpenVPN

OpenVPN

OpenVPN is an open-source VPN solution:

- Started as an open-source project in 2002
- Uses SSL/TLS for security and key exchange
- Operates at the application layer
- Creates a virtual network interface in user space
- Available for all major operating systems

OpenVPN Architecture

OpenVPN's architecture includes several key components:

- **Virtual network interfaces** (TUN/TAP) to capture and inject network traffic
- **SSL/TLS library** for authentication and key exchange
- **Control channel** for management communication
- **Data channel** for encrypted user traffic
- **Configuration system** for defining connection parameters

OpenVPN Protocol Features

OpenVPN offers several protocol features:

- Can operate over UDP (default, port 1194) or TCP
- Uses SSL/TLS for authentication and key exchange
- Implements reliability mechanisms when using UDP
- Employs OpenSSL for cryptographic operations
- Supports various authentication methods:
  - Pre-shared static keys
  - Certificate-based authentication
  - Username/password authentication
  - Multi-factor authentication

OpenVPN Packet Format

```c
/* OpenVPN Packet Structure */
struct openvpn_packet {
    uint16_t packet_length;       // Length of the entire packet
    uint8_t opcode;               // Operation code (message type + key ID)
    /*
     * Opcode bits:
     * - First 5 bits: Message type (control, ACK, data)
     * - Last 3 bits: Key ID to identify encryption key set
     */

    /* Payload (varies by message type) */
    union {
        /* For control messages (handshake) */
        struct {
            uint8_t session_id[8];      // Session identifier
            uint32_t packet_id;         // Packet identifier for replay protection
            uint8_t hmac[20];           // HMAC for message authentication
            uint8_t tls_payload[];      // TLS handshake data
        } control_message;

        /* For data messages */
        struct {
            uint32_t sequence_number;   // Anti-replay sequence number
            uint8_t encrypted_payload[]; // Encrypted IP packet
            uint8_t hmac[];             // HMAC for message authentication
        } data_message;

        /* For ACK messages */
        struct {
            uint8_t session_id[8];      // Session identifier
            uint32_t packet_id_array[]; // Array of acknowledged packet IDs
        } ack_message;
    } payload;
};
```

WireGuard

WireGuard

WireGuard is a modern VPN protocol designed for simplicity and performance:

- Developed in 2015 and integrated into Linux kernel in 2020
- Uses state-of-the-art cryptography (ChaCha20, Poly1305, Curve25519)
- Minimalist design with a small codebase (approximately 4,000 lines)
- Provides high performance with lower overhead
- Offers better roaming capabilities

WireGuard Architecture

WireGuard's architecture focuses on simplicity:

- Operates at the network layer (Layer 3)
- Implements a "cryptokey routing"concept
- Uses public key authentication
- Always employs perfect forward secrecy
- Maintains a simple, stateless design

WireGuard Features

Key features of WireGuard include:

- **Simplified cryptography** - No cipher negotiation, uses fixed set of modern algorithms
- **Fast handshake** - 1-RTT handshake under normal circumstances
- **Clean, minimal codebase** - Easier to audit and more secure
- **Connection-less design** - No persistent connection state
- **Stealth operation** - No response to unauthenticated packets
- **DoS mitigation** - Cookie challenge mechanism
- **Seamless roaming** - Maintains connections across network changes

Basic WireGuard Configuration

Interface Setup

- Generate public/private key pair
- Create WireGuard network interface
- Assign IP address to interface
- Configure listening port

Peer Configuration

- Add peer's public key
- Configure allowed IP ranges for cryptokey routing
- Set endpoint address if known
- Configure optional parameters (keepalive, preshared key)

Routing Configuration

- Add routes for traffic to use the WireGuard interface
- Configure firewall rules if necessary
- Enable packet forwarding if needed

Activation

- Bring up the WireGuard interface
- Test connectivity
- Configure for automatic startup

Basic WireGuard Configuration

```
# Generate private and public keys
wg genkey | tee privatekey | wg pubkey > publickey

# Create WireGuard interface
ip link add dev wg0 type wireguard

# Configure interface
ip address add 10.0.0.1/24 dev wg0
wg set wg0 \
    private-key ./privatekey \
    listen-port 51820

# Add peer
wg set wg0 peer PEER_PUBLIC_KEY \
    allowed-ips 10.0.0.2/32 \
    endpoint example.com:51820 \
    persistent-keepalive 25

# Activate interface
ip link set up dev wg0

# Add route (if needed)
ip route add 192.168.0.0/24 via 10.0.0.2

# Save configuration to file
wg showconf wg0 > wg0.conf
```

Comparing VPN Protocols

Protocol Comparison

IPsec, OpenVPN, and WireGuard have different characteristics:

- **IPsec**:
  - Pro: Widely supported, mature standard
  - Pro: Hardware acceleration in many devices
  - Con: Complex implementation and configuration
  - Con: May be blocked by restrictive firewalls
- **OpenVPN**:
  - Pro: Cross-platform compatibility
  - Pro: Runs in user space
  - Pro: Works well through firewalls (can use TCP port 443)
  - Con: Slower performance than IPsec and WireGuard
- **WireGuard**:
  - Pro: Superior performance and simplicity
  - Pro: Modern cryptography by default
  - Pro: Small, auditable codebase
  - Con: Less mature than alternatives
  - Con: Fixed cryptographic algorithms

VPN Deployment Scenarios

Site-to-Site VPN

Site-to-site VPNs connect entire networks:

- Connect branch offices to headquarters
- Connect partner organizations' networks
- Implemented using VPN gateways at each location
- Transparent to end users
- Typically always-on connections

Remote Access VPN

Remote access VPNs connect individual users to a network:

- Enable employees to work remotely
- Allow access to internal resources from outside
- Implemented using VPN client software on user devices
- Often use dynamic IP addressing with virtual addresses
- Typically on-demand connections

A multinational company uses a hybrid VPN setup to secure its communications. For connecting its headquarters to branch offices in different countries, it deploys site-to-site IPsec VPNs with AES-256 encryption and certificate-based authentication. These connections are always active and transparent to users.
For its remote employees, the company provides OpenVPN access using two-factor authentication. When traveling employees connect from hotel Wi-Fi networks, the OpenVPN client encrypts all their traffic and tunnels it to the corporate network. The system assigns each connected user a virtual IP address (10.3.0.x) from a dedicated pool, enabling internal systems to apply appropriate access controls based on these addresses.

User Authentication

Access Control Components

Access Control Framework

Access control systems typically involve four key components:

- **Identification** - Claiming an identity (e.g., username)
- **Authentication** - Proving the claimed identity (e.g., password)
- **Authorization** - Determining what the authenticated identity may do
- **Accounting/Auditing** - Recording what actions were performed

Authentication Fundamentals

Authentication is the process of verifying that a claimed identity is genuine:

- Establishes trust in the identity claim
- Serves as the foundation for access control
- Must be completed before authorization
- Should be resistant to impersonation attacks

Authentication Factors

Authentication Factors

Authentication is based on three fundamental factors:

- **Something you know** - Knowledge factors (passwords, PINs, security questions)
- **Something you have** - Possession factors (physical tokens, smart cards, mobile devices)
- **Something you are** - Inherence factors (biometrics - fingerprints, face, voice)

Additional factors sometimes considered include:

- **Somewhere you are** - Location factors (GPS, network location)
- **Something you do** - Behavior factors (typing patterns, gestures)

Password-Based Authentication

Password Security Challenges

Password-based authentication faces several significant challenges:

- **Human memory limitations** - Difficult to remember strong, unique passwords
- **Password reuse** - Users often use the same password across multiple sites
- **Password sharing** - Accounts accessed by multiple people
- **Password theft** - Phishing, keylogging, and data breaches
- **Brute force attacks** - Systematic guessing of passwords

Password Best Practices

For Users

- Use unique passwords for different accounts
- Create strong passwords (length over complexity)
- Use a password manager
- Enable multi-factor authentication when available
- Change passwords if compromise is suspected

For System Administrators

- Never store passwords in plaintext
- Use strong, salted hash functions with key stretching
- Implement rate limiting for login attempts
- Maintain a blocklist of commonly used passwords
- Require multi-factor authentication for sensitive systems

For Developers

- Use established password hashing libraries
- Implement secure password reset mechanisms
- Enforce minimum password strength requirements
- Design for multi-factor authentication from the start
- Follow current best practices (NIST, BSI guidelines)

Secure Password Storage

Password Hashing

Password hashing is a one-way transformation to securely store passwords:

- Converts passwords into fixed-length strings that cannot be reversed
- Allows verification without storing the actual password
- Must use cryptographic hash functions designed for security
- Should be combined with salting and key stretching

Salting Passwords

Salting is adding random data to passwords before hashing:

- Prevents use of precomputed tables (rainbow tables) for cracking
- Forces attackers to crack each password individually
- Salt should be unique per user and sufficiently long (64-128 bits)
- Salt is stored alongside the hash in the password database

Key Stretching

Key stretching increases the computational work needed to compute password hashes:

- Applies the hash function repeatedly (thousands or millions of times)
- Increases the time needed for brute force attacks
- Can be adjusted as hardware becomes faster
- Implemented in specialized password hashing functions

Password Hashing Functions

Modern password hashing functions are designed specifically for password storage:

- **bcrypt** - Based on Blowfish cipher, includes salt and cost factor
- **PBKDF2** - Password-Based Key Derivation Function, configurable iterations
- **Argon2** - Winner of the Password Hashing Competition, memory-hard function
- **scrypt** - Memory-hard function designed to be resistant to hardware acceleration

These functions are preferable to general-purpose cryptographic hash functions for password storage.

Secure Password Storage with Argon2

```python
import argon2
import os
from argon2 import PasswordHasher

# Configure Argon2 parameters
# - time_cost: number of iterations
# - memory_cost: memory usage in kibibytes
# - parallelism: number of parallel threads
# - hash_len: length of the hash in bytes
# - salt_len: length of the salt in bytes
ph = PasswordHasher(
    time_cost=3,            # Iterations
    memory_cost=65536,      # 64 MB
    parallelism=4,          # 4 threads
    hash_len=32,            # 32 bytes output
    salt_len=16             # 16 bytes salt
)

def register_user(username, password):
    # Hash the password
    hash = ph.hash(password)

    # In a real application, store username and hash in database
    # db.execute("INSERT INTO users (username, password_hash) VALUES (?, ?)",
    #            (username, hash))

    return hash

def verify_password(stored_hash, provided_password):
    try:
        # Verify password against stored hash
        ph.verify(stored_hash, provided_password)

        # Check if the hash needs to be rehashed (e.g., parameters changed)
        if ph.check_needs_rehash(stored_hash):
            new_hash = ph.hash(provided_password)
            # Update the stored hash
            # db.execute("UPDATE users SET password_hash = ? WHERE password_hash = ?",
            #            (new_hash, stored_hash))

        return True
    except argon2.exceptions.VerifyMismatchError:
        # Password doesn't match
        return False

# Example usage
hash = register_user("alice", "correct-horse-battery-staple")
print(f"Stored hash: {hash}")

# Verification
print(f"Correct password: {verify_password(hash, 'correct-horse-battery-staple')}")
print(f"Wrong password: {verify_password(hash, 'incorrect-password')}")
```

Multi-Factor Authentication

Multi-Factor Authentication (MFA)

Multi-factor authentication combines at least two different authentication factors:

- Requires factors from different categories (know, have, are)
- Significantly increases security compared to single-factor authentication
- Mitigates risks of compromised passwords
- Widely recommended for sensitive systems and accounts

Common MFA Methods

Popular implementations of multi-factor authentication include:

- **One-Time Passwords (OTP)**:
  - Time-based (TOTP) - Generated from a shared secret and current time
  - HMAC-based (HOTP) - Generated from a shared secret and counter
  - SMS-based - Codes sent via text message (less secure)
- **Push Notifications** - Authentication requests sent to mobile applications
- **Hardware Security Keys** - Physical devices using FIDO/U2F standards
- **Biometric Verification** - Fingerprint, face, or voice recognition

MFA Attack Vectors

Despite its strength, MFA can still be vulnerable to certain attacks:

- **Real-time phishing** - Intercepting and forwarding authentication in real-time
- **SIM swapping** - Taking control of a victim's phone number for SMS-based MFA
- **MFA fatigue** - Bombarding users with authentication requests until they approve
- **Malware on endpoint devices** - Capturing authentication data locally
- **Social engineering** - Tricking users into bypassing MFA protections

MFA Fatigue Countermeasures

To counter MFA fatigue attacks, several approaches can be used:

- **Number matching** - Displaying a number on login screen that must be entered in authenticator
- **Geographic context** - Showing location information of the login attempt
- **Device context** - Showing device information of the login attempt
- **Request limiting** - Restricting the number of authentication requests
- **Suspicious activity detection** - Identifying unusual patterns

Passwordless Authentication

Passwordless Authentication

Passwordless authentication eliminates passwords in favor of stronger alternatives:

- Based on public key cryptography and digital certificates
- Typically combines possession and inherence factors
- Eliminates password-related vulnerabilities
- Improves user experience by removing password entry

FIDO2/WebAuthn

FIDO2 is a set of standards for passwordless authentication:

- Combines WebAuthn (web authentication standard) and CTAP (client-to-authenticator protocol)
- Uses public key cryptography for authentication
- Supports various authenticator types:
  - Platform authenticators (built into devices)
  - Roaming authenticators (external security keys)
- Protects against phishing through origin binding
- Private keys never leave the authenticator

FIDO2 Authentication Flow

Registration

- User initiates registration on website/application
- Relying party (website) requests credential creation
- Authenticator generates key pair and prompts user for consent
- Private key stays in authenticator, public key sent to relying party
- Relying party associates public key with user account

Authentication

- User initiates login on website/application
- Relying party sends challenge and requests assertion
- Authenticator prompts user for presence/verification
- Authenticator signs challenge with private key
- Relying party verifies signature using stored public key
- Authentication succeeds if signature is valid

Authentication Protocols

Authentication Protocols

Authentication protocols define the messages and procedures for user authentication:

- Standardize authentication across different systems
- Can be direct (authenticating directly to service) or indirect (using a third party)
- Often establish session keys for secure communication
- May support various authentication methods

Direct vs. Indirect Authentication

There are two fundamental approaches to authentication:

- **Direct Authentication**:
  - User authenticates directly to the service
  - Service has all information needed to authenticate users
  - Credentials are configured on each service
  - Example: Local login on a server
- **Indirect Authentication**:
  - Authentication delegated to a central authority
  - Service trusts the authentication decisions of this authority
  - Credentials are managed centrally
  - Examples: RADIUS, Kerberos, SAML, OpenID Connect

Single Sign-On (SSO)

Single Sign-On (SSO)

Single sign-on enables users to authenticate once and access multiple services:

- Reduces the number of authentication events
- Centralizes credential management
- Improves user experience
- Enhances security by reducing password fatigue
- Can implement consistent security policies across services

SSO Implementation Approaches

SSO can be implemented using various protocols:

- **Kerberos** - Primarily for on-premises environments (e.g., Active Directory)
- **SAML** - Web-based SSO between different organizations
- **OpenID Connect** - Modern protocol for web and mobile applications
- **OAuth 2.0** - Authorization framework often used with OpenID Connect

OAuth 2.0 and OpenID Connect

OAuth 2.0

OAuth 2.0 is an authorization framework (not authentication):

- Enables third-party applications to access resources without sharing credentials
- Uses tokens to grant limited access to resources
- Defines various authorization flows for different scenarios
- Standardized in RFC 6749

OAuth 2.0 Roles

OAuth 2.0 defines four key roles:

- **Resource Owner** - Entity that grants access (typically the user)
- **Client** - Application requesting access to resources
- **Resource Server** - Server hosting the protected resources
- **Authorization Server** - Server issuing access tokens

OpenID Connect (OIDC)

OpenID Connect is an identity layer built on top of OAuth 2.0:

- Adds authentication functionality to OAuth 2.0
- Provides user profile information
- Uses JSON Web Tokens (JWT) for identity tokens
- Enables ßocial loginänd federated identity

OpenID Connect Terminology

OpenID Connect introduces specific terminology:

- **Relying Party (RP)** - Application that wants to verify user identity
- **OpenID Provider (OP)** - Service that authenticates users
- **ID Token** - JWT containing user identity information
- **UserInfo Endpoint** - API for retrieving additional user information
- **Claims** - Pieces of information about the user

OpenID Connect Flow

Authorization Code Flow

- User initiates login to relying party
- Relying party redirects to OpenID provider with client ID and requested scopes
- User authenticates with OpenID provider
- OpenID provider redirects back with authorization code
- Relying party exchanges code for tokens (ID token, access token)
- Relying party validates ID token and creates session

Optional UserInfo Request

- Relying party may request additional user information
- Uses access token to query UserInfo endpoint
- Receives additional claims about the user

SAML

Security Assertion Markup Language (SAML)

SAML is an XML-based framework for exchanging authentication and authorization data:

- Primarily used for web-based SSO
- Enables cross-domain identity and access management
- Separates identity provider from service provider
- Current version is SAML 2.0

SAML Components

SAML defines several core components:

- **Assertions** - XML documents containing authentication statements
- **Protocols** - Rules for requesting and receiving assertions
- **Bindings** - Methods for transporting assertions over different protocols
- **Profiles** - Combinations of assertions, protocols, and bindings for specific use cases

SAML vs. OpenID Connect

SAML and OpenID Connect differ in several ways:

- **Format** - SAML uses XML, OIDC uses JSON
- **Platform support** - SAML for web only, OIDC for web, mobile, and APIs
- **Complexity** - SAML more complex, OIDC simpler to implement
- **Age** - SAML older and more established, OIDC newer
- **Token size** - SAML assertions larger than OIDC ID tokens

Kerberos

Kerberos

Kerberos is an authentication protocol for secure authentication in non-secure networks:

- Developed at MIT in 1983
- Uses tickets to authenticate users without sending passwords
- Built on symmetric key cryptography
- Forms the foundation of Windows domain authentication
- Current version is Kerberos V5 (RFC 4120)

Kerberos Components

Kerberos involves several key components:

- **Principals** - Users, services, or systems that participate in authentication
- **Key Distribution Center (KDC)** - Central authentication server consisting of:
  - Authentication Service (AS) - Verifies user identities
  - Ticket-Granting Service (TGS) - Issues service tickets
- **Realm** - Administrative domain of a KDC
- **Tickets** - Cryptographically protected credentials:
  - Ticket-Granting Ticket (TGT) - Used to request service tickets
  - Service Ticket - Used to access specific services

Kerberos Authentication Process

Initial Authentication

- User logs in and sends username to Authentication Service
- AS checks if user exists in database
- AS sends back data encrypted with user's key (derived from password)
- Client decrypts this using the user's password
- Client extracts Ticket-Granting Ticket (TGT) and session key

Service Ticket Acquisition

- When user needs to access a service, client contacts Ticket-Granting Service
- Client sends TGT and an authenticator encrypted with session key
- TGS verifies TGT and authenticator
- TGS issues service ticket and new session key for the client-service communication

Service Access

- Client presents service ticket to the service
- Client also sends a new authenticator encrypted with client-service session key
- Service decrypts ticket and authenticator
- Service verifies authenticator contents (timestamp within allowed skew)
- Service grants access if verification succeeds

Kerberos Security Properties

Kerberos provides several security properties:

- **No password transmission** - Passwords never sent over the network
- **Limited ticket lifetime** - Tickets expire after a defined period
- **Mutual authentication** - Both client and server can verify each other
- **Replay protection** - Timestamps prevent reuse of authentication messages
- **Key distribution** - Session keys established for secure communication

Cross-Realm Authentication

Kerberos supports authentication across different administrative domains:

- Enables federated identity across organizations
- Requires trust relationship between realms
- Uses a hierarchical trust model or direct cross-realm keys
- Allows users from one realm to access services in another

Federated Authentication

Federated Authentication

Federated authentication enables identity verification across organization boundaries:

- Allows users to authenticate with their home organization
- Enables access to resources in partner organizations
- Eliminates need for multiple accounts and credentials
- Maintains security boundaries between organizations
- Examples include academic federation (SWITCHaai) and enterprise federations

Shibboleth

Shibboleth is a system for federated identity management:

- Based on SAML
- Separates authentication from authorization
- Preserves privacy by limiting attribute sharing
- Widely used in academic and research communities
- Components include:
  - Identity Provider (IdP) - Authenticates users
  - Service Provider (SP) - Protects resources
  - Discovery Service - Helps users find their home organization

Shibboleth Authentication Flow

Resource Access Request

- User attempts to access a protected resource
- Service Provider determines resource requires authentication
- SP redirects user to Discovery Service

Identity Provider Selection and Authentication

- User selects their home organization from the Discovery Service
- User is redirected to their Identity Provider
- User authenticates with their home organization credentials

Assertion Generation and Validation

- IdP generates a SAML assertion containing authentication statement
- IdP may include additional user attributes (depending on configuration)
- Assertion is cryptographically signed and optionally encrypted
- User's browser posts the assertion to the Service Provider

Resource Access

- SP validates the assertion's signature
- SP extracts user identity and attributes
- SP makes authorization decision based on attributes
- SP grants access to the resource if authorized

SWITCHaai is a federation of Swiss universities and research institutions that implements Shibboleth for federated authentication. When a student from ZHAW accesses a learning resource at ETH Zurich, they are redirected to the SWITCH discovery service to select ZHAW as their home institution. After authenticating with ZHAW credentials, the student receives access to the ETH resource without needing a separate ETH account. The ETH service provider learns only the minimal information needed to authorize access, such as the student's affiliation and role, without storing or managing their credentials.

Authorization

Introduction to Authorization

Authorization

Authorization is the process of determining whether an authenticated entity is permitted to perform a requested action:

- Follows successful authentication
- Controls access to resources and operations
- Implements security policies
- Enforces principle of least privilege

Fundamental Building Blocks

Authorization systems consist of three core components:

- **Access Control Model** - Conceptual framework defining how access decisions are made
- **Security Policy** - Rules defining who can access what resources
- **Security Mechanism** - Technical implementation enforcing the security policy

Security Policy Drivers

Security policies are influenced by multiple factors:

- **Business Drivers** - Efficiency, usability, operational requirements
- **Security Drivers** - Risk management, threat mitigation, data protection
- **Regulatory Drivers** - Legal compliance, industry standards

Access Control Models

Major Access Control Models

Three primary access control models dominate modern systems:

- **Discretionary Access Control (DAC)** - Access decisions made by resource owners
- **Mandatory Access Control (MAC)** - Access decisions enforced by system policy
- **Role-Based Access Control (RBAC)** - Access based on user roles within an organization

Additional models include:

- **Attribute-Based Access Control (ABAC)** - Access based on attributes of users, resources, and environment
- **Rule-Based Access Control** - Access based on predefined rules
- **Relationship-Based Access Control (ReBAC)** - Access based on relationships between entities

Discretionary Access Control (DAC)

Discretionary Access Control

DAC is an access control model where the resource owner controls access rights:

- Resource owners determine who can access their resources
- Owners can delegate access control to others
- Access rights can be transferred between users
- Most common model in general-purpose operating systems

DAC Implementation Methods

DAC is commonly implemented through two main mechanisms:

- **Access Control Lists (ACLs)**:
  - Lists of permissions attached to objects
  - Each entry specifies subject and allowed operations
  - Efficiently answers "who can access this object?"
- **Capabilities**:
  - Unforgeable tokens owned by subjects
  - Each token grants specific access rights
  - Efficiently answers "what can this subject access?"

DAC Advantages and Disadvantages

DAC offers certain benefits and limitations:

- **Advantages**:
  - Flexible and intuitive for users
  - Simple to understand and administer
  - Enables fine-grained access control
- **Disadvantages**:
  - Vulnerable to Trojan horse attacks
  - Cannot enforce system-wide security policies
  - May lead to uncontrolled information flow
  - Prone to privilege creep over time

UNIX/Linux File Permissions

Standard UNIX Permissions

UNIX-style file permissions implement a simple form of DAC:

- Each file/directory has an owner and group
- Permissions divided into three categories:
  - Owner (user) permissions
  - Group permissions
  - Others (world) permissions
- Each category can have read (r), write (w), and execute (x) permissions
- Represented as 9 bits, often shown in octal notation (e.g., 755)

UNIX Permission Interpretation

Permission bits have different meanings for files and directories:

- **For Files**:
  - Read (r) - View file contents
  - Write (w) - Modify file contents
  - Execute (x) - Run file as a program
- **For Directories**:
  - Read (r) - List directory contents
  - Write (w) - Create, delete, or rename files within directory
  - Execute (x) - Access files and subdirectories (traverse)

Managing UNIX Permissions

```
# Display file permissions
ls -l myfile.txt
# -rw-r--r-- 1 alice users 2048 Apr 15 14:22 myfile.txt
#  ^ ^ ^  ^
#  | | |  |
#  | | |  +-- Others: read only
#  | | +---- Group: read only
#  | +------ Owner: read and write
#  +-------- File type (regular file)

# Change permissions using symbolic notation
chmod u+x myfile.txt      # Add execute permission for owner
chmod g+w myfile.txt      # Add write permission for group
chmod o-r myfile.txt      # Remove read permission for others
chmod a+x myfile.txt      # Add execute permission for all categories

# Change permissions using octal notation
chmod 755 myfile.txt      # rwxr-xr-x (Owner: rwx, Group: r-x, Others: r-x)
chmod 644 myfile.txt      # rw-r--r-- (Owner: rw-, Group: r--, Others: r--)
chmod 700 myfile.txt      # rwx------ (Owner: rwx, Group: ---, Others: ---)

# Change owner and group
chown alice:users myfile.txt
```

Special Permission Bits

UNIX systems have additional permission bits for special purposes:

- **Setuid bit (4000)** - When set on executable files, the program runs with the permissions of the file owner
- **Setgid bit (2000)**:
  - On executable files: Program runs with the permissions of the file's group
  - On directories: New files inherit the directory's group
- **Sticky bit (1000)** - On directories, files can only be deleted or renamed by their owner (commonly used on /tmp)

Effective UNIX Permission Management

File Ownership

- Assign appropriate owner and group to files
- Create groups based on access requirements
- Use chown and chgrp to modify ownership

Permission Assignment

- Follow principle of least privilege
- For directories, consider child file implications
- Remember execute permission is needed to traverse directories
- Use special bits cautiously

Default Permissions

- Configure umask to set default permissions
- Common umask values:
  - 022 - Files: 644, Directories: 755
  - 027 - Files: 640, Directories: 750

Permission Verification

- Regularly audit permissions
- Test access with different user accounts
- Check effective permissions with access control lists

Access Control Lists (ACLs)

POSIX ACLs

POSIX Access Control Lists extend standard UNIX permissions:

- Allow specifying permissions for multiple users and groups
- Maintain backward compatibility with standard permissions
- Include:
  - Minimal ACLs - Equivalent to standard permissions
  - Extended ACLs - Additional users/groups with specific permissions

ACL Components

POSIX ACLs consist of several entry types:

- **Owner entry** - Permissions for the file owner (user::rwx)
- **Named user entries** - Permissions for specific users (user:alice:rwx)
- **Group owner entry** - Permissions for the file's group (group::rwx)
- **Named group entries** - Permissions for specific groups (group:developers:rwx)
- **Other entry** - Permissions for everyone else (other::rwx)
- **Mask entry** - Maximum permissions for named users/groups and group owner

Managing POSIX ACLs

```
# Display ACLs
getfacl myfile.txt
# file: myfile.txt
# owner: alice
# group: users
# user::rw-
# group::r--
# other::r--

# Add an ACL entry for user bob with read and write permissions
setfacl -m user:bob:rw- myfile.txt

# Add an ACL entry for group developers with read permissions
setfacl -m group:developers:r-- myfile.txt

# View the updated ACLs
getfacl myfile.txt
# file: myfile.txt
# owner: alice
# group: users
# user::rw-
# user:bob:rw-
# group::r--
# group:developers:r--
# mask::rw-
# other::r--

# Remove an ACL entry
setfacl -x user:bob myfile.txt

# Remove all ACLs and revert to standard permissions
setfacl -b myfile.txt
```

A university department sets up a shared project directory for collaboration between faculty and students. They need different permission levels for different groups. They create the directory with standard permissions for the owner (rwx), but then use ACLs to provide tailored access:

```
# Create project directory
mkdir /projects/physics101
chmod 700 /projects/physics101

# Set ACLs
# Faculty get full access
setfacl -m group:faculty:rwx /projects/physics101
# Teaching assistants get read and execute
setfacl -m group:tas:r-x /projects/physics101
# Students get read-only
setfacl -m group:students:r-- /projects/physics101
# Department staff gets full access
setfacl -m group:staff:rwx /projects/physics101

# Create subdirectory for submissions with different permissions
mkdir /projects/physics101/submissions
# Students can write to submissions directory
setfacl -m group:students:r-x /projects/physics101/submissions
# Default ACL for new files in submissions directory
setfacl -d -m group:faculty:rwx /projects/physics101/submissions
setfacl -d -m group:tas:r-x /projects/physics101/submissions
setfacl -d -m group:students:r-- /projects/physics101/submissions
```

This setup allows faculty to manage all content, TAs to access but not modify most content, and students to read course materials while being able to navigate to the submissions directory where they can add their assignments.

Mandatory Access Control (MAC)

Mandatory Access Control

MAC is an access control model where a system-wide policy determines access rights:

- Access decisions enforced by the system, not resource owners
- Based on security labels assigned to subjects and objects
- Security policy defined by system administrators
- Users cannot override or modify access controls

MAC Implementations

Modern operating systems implement various forms of MAC:

- **SELinux (Security-Enhanced Linux)**:
  - Developed by NSA
  - Label-based access control
  - Fine-grained policy control
  - Used in Red Hat/Fedora distributions
- **AppArmor**:
  - Path-based access control
  - Simpler configuration than SELinux
  - Used in Ubuntu/SUSE distributions
- **Windows Mandatory Integrity Control**:
  - Introduced in Windows Vista
  - Based on integrity levels (Low, Medium, High, System)
  - Prevents lower integrity processes from modifying higher integrity resources

MAC vs. DAC Comparison

MAC and DAC differ in several key aspects:

- **Control**:
  - DAC - Resource owners control access
  - MAC - System policy controls access
- **Policy Management**:
  - DAC - Distributed management
  - MAC - Centralized management
- **Security Assurance**:
  - DAC - Lower, vulnerable to user errors
  - MAC - Higher, enforces system-wide policy
- **Complexity**:
  - DAC - Simpler to implement and understand
  - MAC - More complex, requires specialized knowledge

Role-Based Access Control (RBAC)

Role-Based Access Control

RBAC is an access control model based on user roles within an organization:

- Users are assigned to roles based on job responsibilities
- Permissions are assigned to roles, not directly to users
- Users acquire permissions through role membership
- Standardized in INCITS 359-2004

RBAC Components

RBAC consists of four main components:

- **Users** - Entities that need access to resources
- **Roles** - Collections of permissions representing job functions
- **Permissions** - Approved operations on protected resources
- **Sessions** - Mappings between users and their activated roles

RBAC Security Principles

RBAC supports several important security principles:

- **Principle of Least Privilege** - Users have only the permissions needed for their job
- **Separation of Duties** - Critical operations require multiple users
- **Data Abstraction** - Permissions defined at higher levels than individual objects

Implementing RBAC

Role Design

- Analyze job functions within the organization
- Identify required permissions for each function
- Create roles based on job functions
- Define role hierarchies if needed

Permission Assignment

- Identify protected resources
- Define operations on these resources
- Assign permissions to appropriate roles
- Review and refine permission assignments

User Management

- Assign users to appropriate roles
- Consider time and location constraints
- Implement role activation/deactivation
- Review user-role assignments regularly

Administration

- Define role administration model
- Establish processes for role changes
- Audit role and permission usage
- Regularly review the RBAC structure

Attribute-Based Access Control (ABAC)

Attribute-Based Access Control

ABAC is a flexible access control model based on attributes:

- Access decisions based on attributes of:
  - Subjects (users, applications)
  - Resources (data, services)
  - Actions (read, write, execute)
  - Environment (time, location, security level)
- Uses policies that combine attributes with logical rules
- More dynamic than traditional RBAC
- Can implement complex, context-aware policies

ABAC vs. RBAC

ABAC extends beyond traditional RBAC:

- **Flexibility**:
  - RBAC - Predetermined access based on roles
  - ABAC - Dynamic access based on multiple attributes
- **Granularity**:
  - RBAC - Role-level permissions
  - ABAC - Fine-grained, attribute-based decisions
- **Context Awareness**:
  - RBAC - Limited environmental context
  - ABAC - Rich environmental attributes (time, location, etc.)
- **Policy Expression**:
  - RBAC - Role-permission assignments
  - ABAC - Complex boolean logic combining attributes

A healthcare system implements ABAC to control access to patient records. Access decisions consider:

- **Subject attributes**: Role (doctor, nurse, administrator), department, specialty, care relationship
- **Resource attributes**: Patient ID, record type, sensitivity level, department
- **Action attributes**: Read, update, create, delete
- **Environmental attributes**: Time of day, access location, emergency status

A sample policy might be: "Doctors can view patient records if they are the patient's primary physician OR they are on-call AND it's outside regular hours OR an emergency has been declared."
This policy would be impossible to implement with simple RBAC but is straightforward with ABAC. The system can dynamically evaluate multiple attributes at access time to make appropriate decisions based on the current context.

Zero Trust Security Model

Zero Trust

Zero Trust is a security model that assumes no implicit trust for any entity:

- Eliminates the concept of trusted internal networks
- Requires continuous verification for all access requests
- Applies least privilege access controls
- Inspects and logs all traffic
- Focuses on protecting resources rather than network segments

Zero Trust Principles

Key principles of the Zero Trust model include:

- **Verify explicitly** - Always authenticate and authorize based on all available data points
- **Use least privilege access** - Limit user access with Just-In-Time and Just-Enough-Access
- **Assume breach** - Minimize blast radius and segment access
- **Identity-based security** - Identity becomes the primary security perimeter
- **Continuous monitoring** - Collect and analyze data continuously

Implementing Zero Trust

Identity and Access Management

- Implement strong authentication (MFA)
- Use identity as the primary security control
- Apply risk-based authentication
- Establish continuous validation

Micro-segmentation

- Create fine-grained security perimeters
- Implement least-privilege access controls
- Protect individual workloads and applications
- Enforce controls at the application layer

Continuous Monitoring

- Collect and analyze telemetry data
- Detect anomalous behavior
- Maintain comprehensive logging
- Implement behavior analytics

Automation and Orchestration

- Automate security policy enforcement
- Implement dynamic access controls
- Use security orchestration and response
- Regularly test security controls

Basic Security Concept Development

Creating a comprehensive security concept involves:

- **Architecture Diagram**:
  - Identify application components
  - Document interfaces and data flows
  - Specify security controls
- **Network Architecture**:
  - Define network segmentation
  - Specify encryption requirements
  - Document network controls
- **Role Model**:
  - Define user roles and permissions
  - Categorize protected data
  - Create access control matrix
- **Authentication**:
  - Specify authentication mechanisms
  - Document identity providers
  - Define authentication workflow
- **Backup and Recovery**:
  - Define backup strategy
  - Document recovery procedures
  - Specify testing approach

A university develops a security concept for its course management system. The concept addresses:

- **Architecture**: Web application with database backend, integration with student information system, grade export functionality
- **Network**: System components distributed across server and DMZ zones, all external connections use TLS
- **Roles**: Administrators, lecturers, teaching assistants, students, each with specific permissions for course content, enrollment, and grades
- **Authentication**: Federated authentication using Shibboleth with university identity provider
- **Backup**: Daily differential backups with weekly full backups, stored in separate physical location, tested monthly

The concept considers regulatory requirements for data protection and includes specific measures to protect grade information as sensitive personal data.