Question 1: Networking and Internetworking

Netværk intro + forskellige typer
Devices: computers, printers, routers, switches and other.
Forbindelser: kabel, wifi, etc.
Types: PersonalAreaN, LAN, MetropolitanAN (city), WideAN, Internet: interplanetary (SLIDE 7-8: Internet 1n3)

Idea: protocol hierarchies and the benefits
SLIDE 4: Protocol Layers (2): Each layer = more data/inf, layer = specific OSI comunication, layer has protocols like UDP TCP.

OSI and IP model: responsibility at each layer
SLIDE 5-6: OSI og TCP/IP: forklar hvert lag.

Connection oriented vs connectionless network
Conntection-oriented: setup and torn down: e.g. phone call. TCP
Connectionless: messages are handled separately, e.g., postal delivery. UDP

Question 2: Routing

Introduction
What is routing (SLIDE 8: Internet 3)
-> To send information between devices on different networks, involving a number of routers.

Routing table look for connectionless network
SLIDE 13: Connectionless Service - Datagrams:

- Første kolonne: pakke destination.
- Anden kolonne: næste hop mod destination.
- Fejl i A(later) anden kolonne D og D: kan kun send til forbundet routers.

Routing table look for connection oriented network
SLIDE 14: Connection-Oriented - virtual circuits:

- Pakker får tags
- Router tabeller viser hvilke tags sendes hvorhen.
- Samme tag, altid samme vej.
- FORSVARET TRUNKS

How to contruct and maintain routing tables in connectionless network
SLIDE 15: OPTIMALITY PRINCIPLE: Find best route, CHANGES OVER TIME: failures, turned off hardware, or new hardware.
SLIDE 18: Link State Routing - LSPs:

- Each node FLOODS information about neighbors in LINK STATE PACKETS
    - To learn full network graph
- Each node runs Dijkra's algorithm

LSPs har SEQ number og AGE, der gør at de dør. Internettet er træ/hiraki, så nettet ikke bliver flooded.

Representation of routing tables - compact form
TODO ??? At kun kende sin nabo????

Question 3: Transport Layer

Introduction

SLIDE 5: OSI: Lvl 4, TCP og UDP. First layer to handle data loss.
SLIDE 25: Services Proviced to the Upper Layers:

- Transport address = PORT NUMBER
- Network address = IP
- TPDU: Transport Protocol Data Unit: msg encapsulation format that adds several bytes of routing header.

SLIDE 26: Transport Service Primitives: Teorien bag, men kom. er baseret på pakker.

Three-way handshake

SLIDE 27: Connection Establishment: Forklar.
SLIDE 28: Connection Establishment:

- a: host1: SEQ passer ikke. Gammel eller ugyldig CR.
- b: host1: samme. host2: SEQ og ACK ugyldig.
- Indeholder også TIMEOUTS som backup.

SLIDE 29: Connection Release.
SLIDE 30: Connection Release: Fungerer godt hvis intet pakketab.
SLIDE 31: Connection Release: Forklar timers.

Sliding window protocol

Ideen er at man TESTER hvor mange PAKKER modtageren kan modtage, og så JUSTERER ud fra det.
SLIDE 39: TCP Sliding Window: Modtager fortæller: DATA MODTAGET, og FRI PLADS TILBAGE.
SLIDE 41: TCP Congestion Control: Forklar: amount *2 -> Threshold -> +1. Start fra 0.
SLIDE 42: TCP Congestion Control: Forklar: …
Sliding Window: TESTER hvor mange pakker MODATGEREN kan MODTAGE.
Congestion Window: TESTER hvor mange PAKKER der kan være i NETVÆRKET.

Question 4: RMI (Remote Method Invocation)

Introduction
Remote method invocation tillader at kalde metoder på objekter på tværs af et netværk, semantisk ækvivalent som at det var samme maskine.

Goals
- At benytte eksisterende netværks protokoller
- Generalisering af eksisterende primitive typer i programmerings sprog for at understøtte udvikling af distribuerede systemer.
- Distributed "garbage collection"
- At minimerer forskellen imellem at arbejde med lokale og "remote" objects.

Why is RMI suitable for distributed systems?
RMI for distribuerede systemer fordi det gør implementering nemmere ved at ABSTRAHERE NETVÆRKS DELEN og tillade implementering næsten som om det ikke var på tværs af netværk.

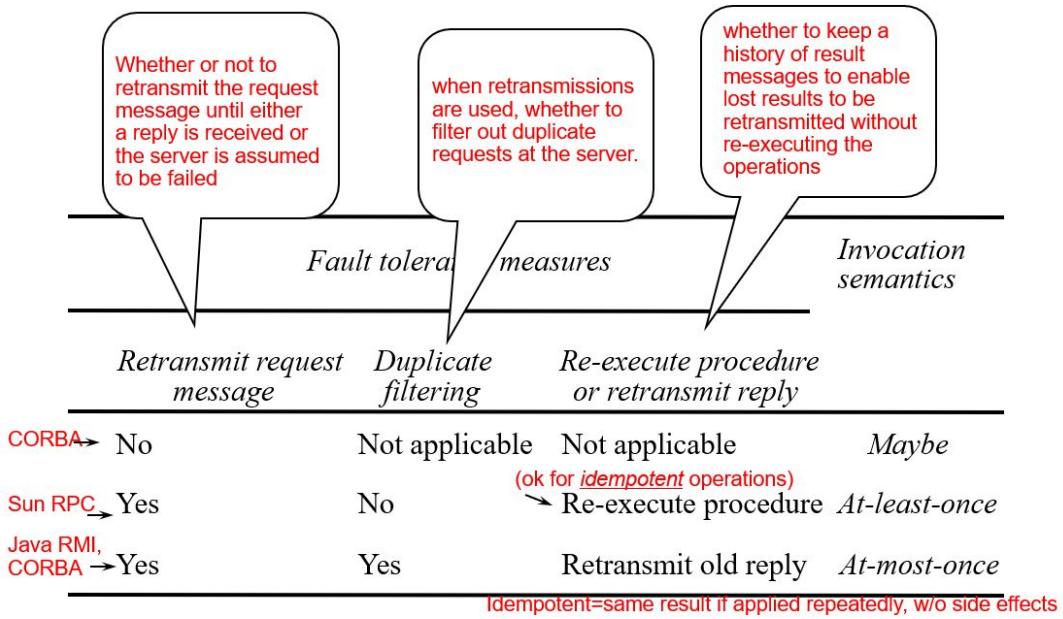
How are remote invocations different from local invocations?
For klienten er der ikke forskel på "remote" og lokale "invocations", men nogle ekstra elementer er nødvendige for at både klient og server kan se hvad den anden tilbyder.

Implementering af RMI
SLIDE 46: Implementing RMI: Forklar:
- Proxy objekter: Klasser der udgiver sig for at være lokale, men så invoker remote klasser når de bliver brugt. (Ideen fra RMI, gør remote lokalt)
- Remote reference module: ansvarlig for at oversætte til og fra lokale og "remote" objekt referencer, og for at lave referencer til "remote" objekter.
- Communication module: står for kommunikationen imellem klient og server.
- Skeleton & dispatcher: (Server side) hver process har én dispatcher, og en skeleton for hver lokal klasse.
    - Dispatcher: modtager all "request msgs" fra commiunication module, og videregiver den til det skeleton der matcher method id.
    - Skeleton: Un-marshals request og får argumenterne, invoker den tilsvarende metode, og marshals svaret og returnerer det til communication module.

Marshals: At lave en samling af "data items" (platform dependent) til en extern data representation (platform independent).

Un-marshals: Det modsatte.

Indempotent operation: Samme resultat hvis "applied" repeatedly, w/o side effects.



What happens step by step during a remote method invocation
SLIDE 44: Remote and local method invocations.

JAVA RMI: Remote interface, is just an interface.

What is the purpose of the registry
Serverne benytter dette registry til at angive deres tilgængelighed, og klienten benytter dette registry til at finde referencer til det "remote object" de vil invoke.

What is static and dynamic invocation
Static: pre-compiled skeleton and stubs. You ALREADY KNOW where and what you're calling.
Dynamic: makes calls to the registry to DISCOVER what services are available and how to call them.
In theory, this means that you don't need to know anything about the API. In practice...I don't think I've ever seen anybody do this in any realistic way.

Question 5: DSF (Distributed File System)

Introduction
DFS et system hvor lagring og tilgang af filer er baseret på en klient/server arkitektur. En eller flere servere lagre filer der kan blive tilgået, med de rigtige rettigheder, af klienter i netværket.
KRAV -> high transparency og performance.

Requirements to distributed file systems:
- Transparency: Access, location, mobility, performance, scaling.
- Concurrent file access
- Replication
- Heterogeneity of OS and hardware (understøttelse)
- Fault tolerance
- Maintaining consistency of data
- Security
- Efficiency

File Service Architecture
SLIDE 47: File Service Architecture: Giver adgang til filer ved at strukturere file service som 3 komponenter: (Ansvar)
- Flat file service:
    - Implementation af operationer på indhold af filer. (+ og - af: read, create, GetAttr)
    - Unique File Identifiers (UFIDs) er brugt til at refererer filer i alle "requests" for flat file service operations.
- Directory service:
    - Mapping imellem text file navne og deres UFIDs.
    - Supports tilføjelse af nye filer til mapper.
    - Operations: Lookup, AddName, UnName, GetNames.
- Client module:
- Tilbyder integreret service til klienten. Som i at på UNIX emulerer den alle unix file operations.
- Har også netværks lokation på serveren, og directory server processes.

Network File System (NFS)
Network file system tillader at man mounter remote file systemer systems (or a part of it). Rettigheder kan styre tilladelser: read-only, read-write. NFS bruger Remote Procedure Calls (RPCs) til at sende requests imellem klient og server. NFS version 2 og 3, tillader UDP over IP netværk og derved skaber "stateless" netværk forbindelser imellem klienter og server.

Stateful: Maintains a state of all open files.
Write through: klient svarer tilbage med det samme.
Idempotent operation: Samme resultat hvis "applied" repeatedly, w/o side effects.
stateless server: don't "store" data. Database og sådan ligger et andet sted.

SLIDE 49: Local and remote file systems accessible on an NFS client.

Question 6: P2P (Peer-to-Peer)

What is a p2p system: Et netværk hvor alle computere er "lige/peers", og lokaliseret i udkanten af netværket. Et logisk "overlay network", der ligger ovenpå et IP netværk.

Goals:
- Deling af data og resourcer på meget stor skala
    - Ingen central og separat servere -> self organised
    - Deler belastningen ved at bruge resourcer (mem. and CPU) fra "End-hosts" lokaliseret i kanten af netværket.
    - Dynamisk set af "peers"
- Privacy
- Anonymity

P2P Types:
- Unstructured: Links in overlay created arbitrarily
- Semi-structured: p2p systemes with super-peers
- Structured: logical topology on node and data ID's (most DHT's)

How are they different from centralized or client/server systems
- Alle "nodes" i netværket er lige og har samme ansvar. Ingen central server eller lignende
- Anonymitet og privacy

Common Issues
- Organize, maintain overlay network
    - Routing
    - Node arrivals
    - Node failures
- Resource allocation/load balancing
- Efficient placement & localization
- Locality (network proximity)

Idea: generic P2P middleware (aka "substrate"). Dette bliver håndteret af distributed hash table (DHT).

What is a distributed hash table (DHT)
DHT er det der gør at P2P netværk ikke skal bruge en server ved at uddele data over et antal nodes. DHT er et hash table der mapper en key til en value og er distribueret over et antal nodes i netværket. Hver node har en **GUID (Gobally unique ID)**
Ny "content (filer, etc..)" ankommer: Når noget nyt er tilføjet til netværket, bliver der genereret en hash key og en besked bliver sent til alle nodes der deltager i DHT. Når beskeden så når noden ansvarlig for denne key, bliver den gemt i denne node, sammen med dens value. En bruger kan så queryring hvilken som helst DHT node med en hash key genereret baseret på hvad de ønsker, og får dataen på samme måde ved at forspørgslen går fra node til node.

Hashing gør det hele tamperproof da man ikke kan ændre i en fil og stadig have samme hash.

Pastry (DHT implementation)
SLIDE 50: Pastry:
Hvis alle nodes kende alle nodes i netværket ville der altid kun være et hop -> slide 50. Men dette gør at alle nodes skal indeholde rigtig meget data = umuligt hvis rigtig stort. Derfor indeholder alle nodes bare et givent antal naboer, og hopsne vil derfor være -> slide 51. Forklar routing ud fra de to slides.

SLIDE 51: Pastry Routing:
Giver **ids** til **nodes** og bruger en **virtual ring**.
Leaf set: hver node **kender** dem/den der kommer før og efter. (**naboer**)
Routing er baseret på "**prefix matching**" og er derfor log(N).
Pastry forsøger også at tage **højde for** den **underliggende netværks topology**, ved at gøre **distancen** imellem dem så **små** som muligt.
Nabo peers er lagret på følgende måde: hvis jeg har id **011101**, og min nabo har **011000**, så vil jeg lagre det som **011***. Som i at de første 3 bits er det samme, og resten er beskrevet med *, da fjedre bit er forskellig. Når noden skal route til en peer, finder den den nabo med den "**largest matching prefix**". **Som i at den ser hvor mange bits, fra venstre, den kan få til at matche med en nabo.** Den var flest bits fra venstre matcher, router den videre til. Hvis noden har flere naboer med **samme** længde matching **prefix**, vælger den den med "**shortest round-trip-time**" - denne værdi kommer fra den **underliggende netværks topology** (noget lignende kortest ping time).

What information is contained by the routing table
SLIDE 52: Pastry Routing Table: **GUID** mapped til IP, på andre nodes end den selv. GRÅ = dens egen GUID. Så den behøver ikke at have alle fyldt ud, den vælger bare den med longest matching prefix.

How does Pastry function when nodes fail or appear/disappear dynamically
Forspørgsler er **bundet** på **hashing** og ud fra værdien vælges den node der "numerical" er **tættest på værdien**. Så hvis en node forsvinder eller ankommer, vil systemer **automatisk** sende forspørgsler til den på det tidspunkt mest "relevante".

What disadvantages do you see of the pastry approach
I can be very slow? = a lot of hops

Question 7: Physical Time

Introduction
Enheder der fungerer som et ur, der er brugt i computere.

How are clocks implemented in computers
Man kan købe **modtagere**, der kan modtage **radio bølger fra satallitter**, som kan **forbindes** til **computere**. Men det **normale** er at der i en computer sidder noget **HW** der tæller **oscilleringer** af en **quartz krystal**. Efter et **bestemt antal af svingninger** ligger uret én til et **register**.
Resolution are perioden imellem klokke opdateringer.
OS vedligeholder en software klokke der er baseret på registret.

What are the sources of inaccurate clocks

- Resolution: Hvis to events sker med kortere tidsinterval end Resolution, kan disse ikke adskilles i tid.
- Alle quartz uret "skew"s over tid.
    - Kommer an på temperatur.

Why can't computer clocks not be 100% accurately synchronized
Fordi urer "**skew**" forskellige og vil derfor **over tid** ikke være ens, og når urer **synkroniseres**, skal der kommunikeres og det kan **ikke** gøres "**instant**".

What is internal/external synchronization
External: urerne bliver sat efter en **ekstern** "authoritative" **source**.
Internal: urerne er synkroniseret **relativt til hinanden**.

How does Christians method and the Berkeley Algorithm synchronize clocks

Disse to algoritmer bliver brugt i asynkrone systemer.

Christians Algorithm
Client-server synkronisering.

- Klienten sender en **tidsreserverings forespørgsel** til **serveren**.
- **Serveren returnere så dens tid**.
- Klienten måler **hvor lang tid det tog**. Fra SEND til SVAR MODTAGET.
- Klienten **sætter så sit ur til**: serverens tid + (Fra SEND til SVAR MODTAGET) /2

Præcisheden er Fra SEND til SVAR MODTAGET /2 - minimal message delay.

- Det er **ikke** tilladt at justere uret **tilbage i tiden**.
- Og at **springe frem** kan også skabe problemer
- Så uret **justeres langsomt over tid**.

Berkeley Algorithm
Designed for intern synkronisering.

- Vælg en maskine om er mester (M)
- M **forespørger all andre maskiner deres** lokale tid.
- M udregner en **gennemsnit klokkeslæt** baseret på de modtagne tider.

- M siger så til alle hvad de skal justerer deres ur med for at være det samme som **gennemsnittet (+/-)**

What is the network time protocol
Skal vide hvad det er - behøver ikke at gå i dybten.

Synkronisering af klienter relativt til UTC, Universal Time Coordinated, på et **internet-wide scale**.
Pålideligt selv hvis store dele mister forbindelsen.
Tillader ofte synkronisering.

Question 8: Logical Time

Introduction

Logical time vs physical time

Physical time har man et ur, ofte med en quartz krystal. Hvis man så har flere computere kande synkroniserer deres ure indbyrdes eller stille dem efter en ekstern kilde. Det skaber en del problemer: skew, delay on synkronising.

Logical time er baseret på viden om at nogle events sker før andre, og på den måde kan man opstille dem i rækkefølge baseret på tid. Som i at vi ikke ved hvornår dette event præcist skete, men vi ved at den kom efter denne og før denne. Dette kaldes **Causal ordering**.

Happens-before relation

For eksempel at sende og modtage en besked. Så ved vi at beskeden skal sendes før den kan modtages. Så derfor er forholdet i tid altid at sende kommer før modtage.

SLIDE 53: Example - Event Relation: a->b->c->d->f, men vi ved ikke om e kommer før eller efter b!

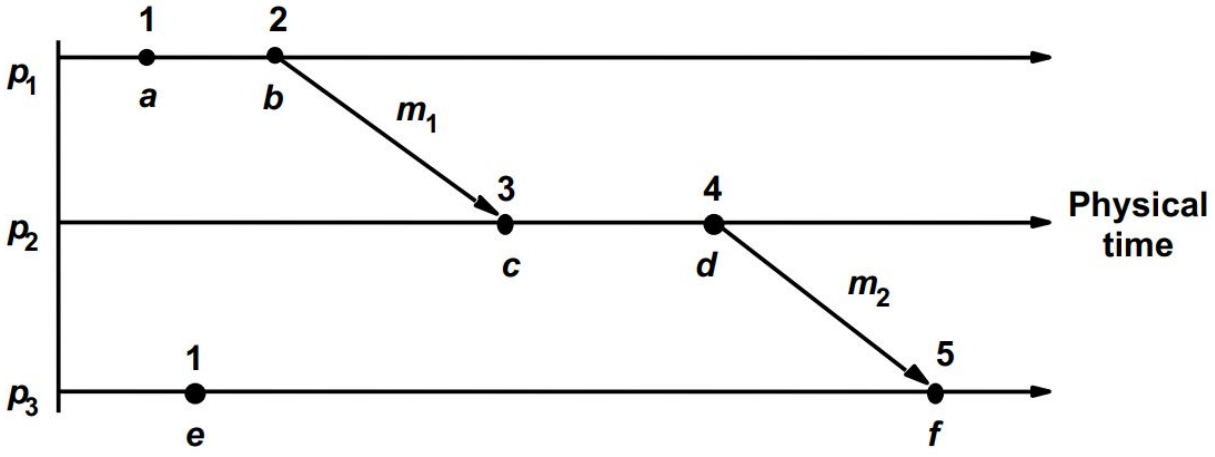
Lamport clocks

Lamportclocks er en måde at holdestyr på hvad der er vist på det ovenstående slide aka causal ordering.

Fremgangsmåde:

1. En proces ligger 1 til dens counter før hvert event.
2. Når en proces sender en besked, inkluderer den dens counter værdi.
3. Når en besked modtages, sammenlignes den modfølgende counter med processens egen counter. Og processens counter updateres baseret på hvilken værdi er højst. Så ligges der en til og dette er timestamp for received msg.

Forklar fremgangsmåden ud fra exam slide med disse værdier:

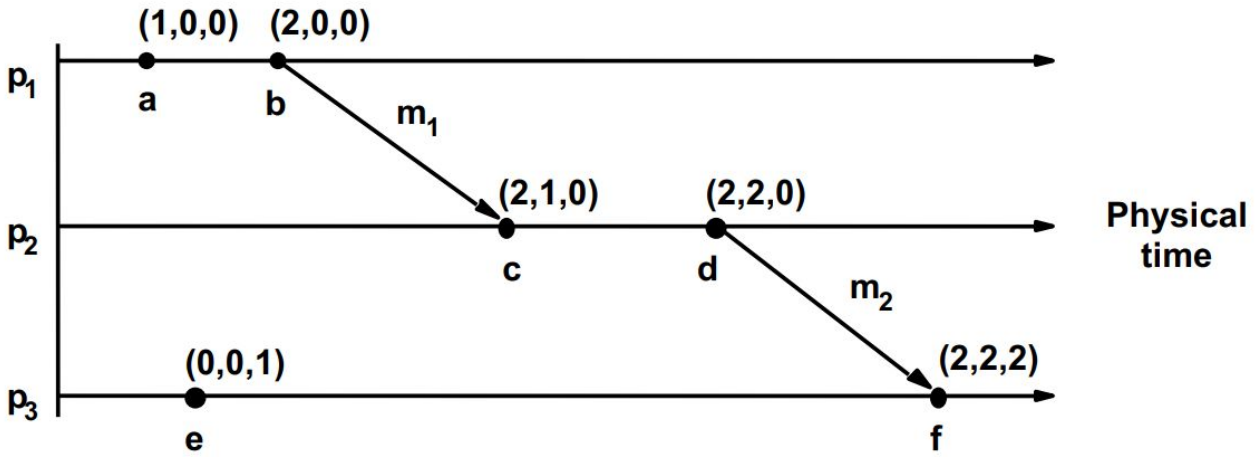


What is vector clocks

Ligesom med Lamportclocks bliver urerne sendt med beskeden imellem processerne. Alle processors ure er lagret i en vektor med størrelse lig med antal af processer. 5 processor = 5 tupel vektor.

- Alle ure starter på 0
- Ved internt event, "increments" den proces' ur i vektoren.
- Besked sendes: opdateres dens ur, altså kun en gang, og så sendes der en kopi af vektoren med beskeden.
- Besked modtages: increment own clock value, og opdater alle værdier i egen vektor med den højeste værdi fra enten den modtagne vektor eller egen.

Forklar fremgangsmåden ud fra exam slide med disse værdier:



What can be done with vector clocks that cannot be achieved with Lamportclocks
Vector clocks kan se forskel på om to operationer er concurrent eller causally dependent. Holder mere information.

LamportClocks kan ikke konkludere at der er "causal happend-before relation". e(1) og c(3), 1 er mindre end 3, men det betyder ikke at "e" kom før "c".

Vector clocks kan: To determine if two events are concurrent, do an element-by-element comparison of the corresponding timestamps:

- Hvis alle sæt af værdier i en vektor er størrer end værdierne i den anden, så kommer den med højeste værdier efter den med mindste værdier.
- Hvis nogle værdier er større og andre er mindre, så er de "concurrent".

e og c er "concurrent".

Question 9: Security

What are the main security goals

- **Confidentiality**: sikkere sig at den **rette modtager**, og kun denne, modtager beskeden/data'en. Aka, hvis en **ikke godkendt person** får fat i dataen kan han **ikke bruge** den til noget.
- **Integrity**: at sørge for at **data'en** der krypteres, **sendes og modtages, er komplet**. Eks. en person modtager data, og **data'en kun kan dekrypteres hvis den er komplet**.
- **Availability**: sørger for at data'en er **tilgængelig** til de **autoriserede parter**.

What kinds of threats are a distributed system subject to

Threats:

- Eavesdropping: "Lytte" til data for at finde f.eks. en nøgle.
- Masquerading: At udgive sig for at være en anden for at få adgang.
- Message tampering: Ændre i beskeder.
- Replaying: WiFi hack, optag authentication, og gensend.
- Denial of service: Dette angreb kan bliver brugt til at lægge dele af et system ned.

Describe the difference between shared secret and public-private key cryptography

Shared-secret-key crypto:

(Symmetric cryptography)

Shared secret, er et stykke data, der kun er kendt af de involverede parter. Det kan være et kodeord, et tal eller noget lignende. Det er brugt i kommunikation og enten aftalt på forhånd eller ved starten af en komminikation.

Public-private-key crypto:

(asymmetric cryptography)

Public keys:

- only be used to encrypt msgs

Private keys:

- used to decrypt msgs with a matching public key
- is kept secret
-

Public key kan blive delt ud til alle, og private key bliver holdt hemmelig. Brugere kan så kryptere data med public key og verificere digitale signaturer. Digitale signaturer kan blive lavet med private key.

Digital signatures - content is digitally signed with an individual's private key and is verified by the individual's public key

Encryption - content is encrypted using an individual's public key and can only be decrypted with the individual's private key

Foreskelle

Shared-secret bruger en nøgle til at kryptere og dekryptere. Public-private-key benytter en til hver.

How can public-private key cryptography be used to ensure the integrity and authenticity of a message

Ved at sørge for at data'en der krypteres, sendes og modtages, er komplet. Eks. en person modtager data, og data'en kun kan dekrypteres hvis den er komplet.

Digitale signaturer.

What is a HMAC
Hash-based message authentication code.
Hvor der bruges en hash function og en hemmelig nøgle til at verificere både data integrity og beskedens autentitet.
HMAC uses two passes of hash computation. The secret key is first used to derive two keys – inner and outer. The first pass of the algorithm produces an internal hash derived from the message and the inner key. The second pass produces the final HMAC code derived from the inner hash result and the outer key.

Explain the Needham Schroeder protocol or Kerberos
Chosen: Kerberos: SLIDE 58: System Architecture of Kerboros:

Authentication protocol for client/server applications. Purpose: security and authentication.
Formål: undgå at kodeord bliver aflyret (eavesdropping), og samtidig give authentication til brugere.
1:

- Request er enkrypteret med brugers shared-secret-key.
- KDC, har også denne key, og verificere brugeren.

2: svaret brugeren får tilbage er krypteret med en anden key, som kun KDC kender.
3: Dekrypterer så beskeden med KDC key.
4: Server ticket er enkrypteret med en 3. key, som kun KDC og Server S, kender.
5: Server S, verificerer så brugerens token med den key der er kendt af S og KDC.
6: Hvis godkendt, får brugeren en session key, der giver brugeren adgang i et stykke tid.