Jador x Lino Golden - Dau Moda | Official Video **dedicatie de la cami pt clej desi nu eu am pus melodia**

acu am vazut ca melodia e scoasa de ziua mea - gloria =))))))

CAND DATI CTRL F LA UN CUVANT DATI “CUVANT “

ADICA PUNETI SPACE CA SA NU FACA MATCH CU TOT DOCUMENTUL

eu am pus cu rosu întrebările/capitolele/raspunsurile de care zicea profu la consultatie

cu roz e ce nu știm

CUPRINS

- NAT
- IPV6 (motivatie, fragmentare)
- Dirijarea continuare
    - Vectorii distantelor
    - Problema numararii la infinit & solutii (split horizon + RIP)
    - Link state + Dijkstra + inundare
- Structura ierarhica a internetului AS \
    - Zone AS, mesaje OSPF
    - in AS: inundare, intre AS: BGP (tot cu Vectorul distantelor)
    - Dirijarea in retelele ad hoc
- Pachete Route Request si Reply

6 TRANSPORT

- functii
- TCP & UDP : utilizari, headere
- Socketi: conexiuni server-client, UDP & TCP
- TCP
    - stabilirea conexiunii
    - corectitudine si corectie, checksum!
    - controlul fluxului de date
    - probleme
    - controlul congestiei
    - controlul ceasurilor – RTT si RTO
    - The Real Time Transport Protocol
- UDP

7 DNS

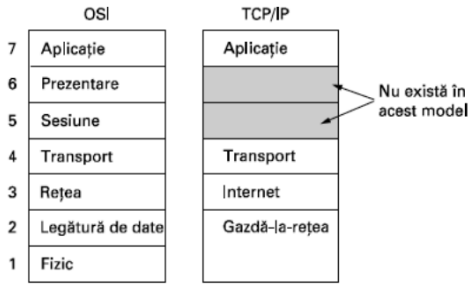
11 CRIPTOGRAFIE+ 12 SECURITATE

- principiile securitatii
- ipsec
- SA - ah + esp
- vpn
- sch de mesaje
- ssl
- cand se face verif certificatullui
- https
- semnatura digitala

2 MODELE DE REFERINTA

Grile

- Anumite protocoale în ce nivel sunt?
    - Ctrl+F după “Protocoale OSI” >- slide 28 în PDF
- Asocierea dintre stiva OSI si stiva TCP-IP



    - Nivelul inferior (fizic + lg de date) devine legatura gazda-retea (host - network)
        - mare varietate de protocoale
        - modelul nu impune reguli despre acest nivel
    - Nivelul retea devine nivelul internet
        - un singur protocol, IP
        - pentru retelele interconectate
    - Nivelul transport se pastreaza
        - TCP - canal sigur pentru siruri de octeti
        - UDP - canal nesigur pentru livrarea datagramelor
    - Nivelele prezentare si sesiune nu sunt mentionate in modelul TCP/IP
    - Nivelul aplicatie se pastreaza
        - varietate de protocoale pentru transferul fisierelor si al postei, login de la distanta, managementul retelei
- Nivele stiva OSI + fiecare nivel ce rol are
    - Nivelul fizic - transmitere a sirurilor de biti pe un canal de comunicatie
    - Nivelul legatura de date - realizează comunicarea sigură și eficientă a datelor între două noduri adiacente (conectate printr-un canal fizic de comunicatie)
    - Nivelul rețea - transmiterea pachetelor între oricare două noduri din rețea

    - Nivelul transport - asigura un transfer de date corect, eficient între procese din sistemul sursa și din sistemul destinatar
    - Nivelul sesiune - controlul dialogului între aplicații
      - sincronizarea transferurilor
      - stabilirea unor puncte de verificare și reluare a transferurilor
    - Nivelul prezentare - conversia formatului datelor intre sintaxa folosită de aplicații și sintaxa de transfer
    - Nivelul aplicație - servicii comune unor categorii de aplicații: schimb de mesaje, transfer de fișiere, terminal virtual, serviciu de directoare

3 LEGĂTURA DE DATE

- Cum pot sa-mi dau seama la nivelul legatura de date ca am ajuns la finalul unui cadru?
    - Folosind marcajele de inceput si final: SOH, STX, ETX. Caractere care marcheaza inceputul si finalul cadrului.
    - Pot sa am in header un camp de length (nu neaparat valabil doar pentru nivelul legatura de date) care sa-mi spuna cati bytes are cadrul => oricine primeste cadrul respectiv poate sa numere x bytes de la inceputul body-ului si cand a ajuns la x bytes sa considere ca s-a incheiat cadrul.
- Cum se face umplerea cu caracterxe? Putem primi un sir de biti ca un cadru si sa spunem daca considerand ca avem de ex caracterele speciale astea astea si astea care este cadrul rezultat dupa ce se face escaparea (umplerea cu caracterele respective)?
- Cum se face umplerea cu biti? Pentru un sir de biti care este cadrul rezultat dupa ce se face umplerea cu biti?
- Cum se face detectia si corectarea erorilor?
    - d(u,v) egal cu nr minim de modificari care trebuie facute pt a ajunge de la un cuvant la altul
    - Sn - multimea cuvintelor cu sens
    - Pentru a **detecta** a cel mult k erori se alege Sn astfel ca: d(u,v) >= k+1 pentru orice u,v din Sn
    - Pentru a **corecta** cel mult k erori se alege Sn astfel ca: d(u,v) >= 2*k+1 pentru orice u,v din Sn
    - S10 = {0000000000, 0000011111, 1111100000, 1111111111}
    - d(u,v) = 5 => putem **detecta** erori de maxim 4 biti & putem **corecta** erori de maxim 2 biti
- Diferenta intre detectia si corectarea erorilor?
    - Detectia erorilor se refera la detectia erorilor cauzate de anumiti factori in timpul transmisiei de la sender la receiver, in timp ce corectarea erorilor reprezinta detectia si reconstructia mesajului initial, inainte sa fie corupt.
- Ce este un bit de paritate?
    - Sender-ul are o secventa de biti pe care urmeaza sa o trimita. In prima faza, acesta numara cati biti de 1 se afla in secventa respectiva pentru a-i determina paritatea. In functie de paritatea obtinuta, el adauga un bit de

paritate la finalul secventei (0 - para, 1 - impara) si trimite mesajul. Receiver-ul primeste mesajul si calculeaza paritatea secventei fara ultimul bit, si apoi compara rezultatul cu valoarea ultimului bit. Metoda nesigura, pt ca nr par de erori => verificarea bitului de paritate nu le va detecta.

- Ce este, cum functioneaza si cum se calculeaza un checksum?
    - Un checksum este o secventa de numere si litere folosite pentru a verifica mesajul de erori. Daca stii checksum-ul mesajului original, poti folosi o functie de checksum pentru a confirma ca este egal cu cel al mesajului care a ajuns la tine.
    - Checksum oferă facilitatea de a verifica dacă informaţiile folosite la procesarea unei datagrame au fost transmise corect. Datele pot conţine erori. Dacă verificarea eşuează, datagrama este ignorată de entitatea care detectează eroarea.
    - Se calculează suma valorilor zecimale a fiecărui caracter din mesaj. Se împarte rezultatul la 255. Restul împărțirii se salvează și se adaugă la mesaj.
- Ce este metoda Hamming? (prea de detaliu => fara calcul de mana)
    - Hamming este o metoda folosita pentru corectarea erorilor de 1 bit.
- Coduri polinomiale + cum functioneaza CRC-ul?
    - Se da în cerința o secventa de biti și un generator. Luam secventa de biti și vedem cati biți de control trebuie sa ii adaugam, iar ca să faci asta tre sa se respecte **formula nr_initial_biti + nr_biti_control < 2^nr_biti_control**, adică în cazul asta 10 + 4 < 2^4 ceea ce se respecta. Dacă cumva nu se respecta ca sa zicem ca aveam o secventa de 13 biți 13 + 4 ar fi dat 17 care nu i mai mic decât 16 însemna că tre sa ii adaugi 5 biți de control ca pt 5 verifica. După doar concatenezi la secventa care ți se da zerouri în fct de câți biți de control ai si imparti ce obții la generator. Restul pe care îl obții îl concatenezi la secventa inițiala, acesta fiind cel care se verifica.
    - CRC se foloseste pentru detectia erorilor (receiver-ul repeta calculul de mai sus si compara restul obtinut cu restul primit de la sender; daca difera => eroare). CRC detecteaza 100% erori de 1 bit, 2 biti, un nr impar de biti, erori in rafala de lungimea codului CRC.
- Protocoale start-stop. Care este forma lor? Care sunt operatiile facute de sender si de receiver? Ce erori pot sa apara?
    - Forma -> desen in slide-uri (ctrl + F dupa “Protocol Stop and wait”)
    - Sender-ul trimite un singur cadru si asteapta primirea unei confirmari (ACK)
    - Receiver-ul trimite inapoi la sender confirmarea (ACK)
    - Pot aparea doua tipuri de erori: mesajul se poate pierde inainte sa ajunga din A in B, sau acesta poate ajunge, dar sa se piarda ACK-ul trimis inapoi de catre B in A.
- Se poate da un anumit flux de mesaje, in care s-a pierdut mesajul X. Daca am un protocol cu fereastra glisanta ce se intampla? Cum se corecteaza eroarea? Ce se intampla daca am un protocol start-stop?
    - Start-stop: pentru a se corecta eroarea se adauga un numar de secventa la time-out si se retransmite ultimul cadru, pe care B il accepta daca este corect (daca eroarea a aparut intre A si B), sau B ignora mesajul daca este dublura, dar totusi retrimite ACK (daca eroarea a aparut intre B si A).

    - Protocolul cu fereastra glisanta nu asteapta confirmarea cadrelor precedente pentru a transmite cadre noi
    - + go back n si selective repeat ce se intampla in ambele cazuri
- Exemplu de problema cu alte valori numerice
    - Ctrl + F dupa “Problema”
- De ce am flag-ul asta la inceput si la final? -> Ethernet
    - Marcheaza inceputul si finalul cadrelor.
- Sa explicam cum se face incapsularea pachetelor? + situatie -> Ethernet
    - Avem un pachet PPP care este intr-un pachet PPPoE, care este intr-un pachet Ethernet.

4-5 RETEA

- **Nivelul retea** - se ocupa cu dirijarea si forwardingul pachetelor
- Protocolul IP:
- *Mecanismul de fragmentare*
    - **Ce este si ce rol are MTU?**
        - MTU (Maximum Transmission Unit) este datagrama de dimensiune maxima ce poate fi transmisa prin urmatoarea retea. Prin compararea acestei valori cu lungimea totala a datagramei se stabileste daca este nevoie de fragmentarea acesteia sau se poate trece la urmatorul pas.
    - **Cum functioneaza flagurile MF si DF?**
        - DF - marcheaza un pachet cu flag Don’t Fragment
            - acest pachet nu poate fi fragmentat daca valoarea DF este setata la 1
        - MF - marcheaza un pachet cu flag More Fragments
            - trebuie sa aibe valoarea 1 pentru toate pachetele mai putin ultimul fragment -> asa se marcheaza faptul ca un anumit fragment este ultimul si se poate incheia partea de reasamblare a fragmentelor
    - **Ce este si cum funcționează offset-ul pentru fiecare fragment**?
        - Offset-ul este un camp ce indica pozitia fragmentului primit in datagrama originala si este masurat in unitati de cate 8 octeti.
        - Functionare (in cazul in care este necesara fragmentarea datagramei)
            - offset-ul primului segment fragmentat va fi mereu 0
            - offset-ul urmatoarelor fragmente va fi calculat ca fiind offset-ul fragmentului vechi la care se aduna numarul de blocuri de cate 8 octeti ai fragmentului curent
    - Se da un sir de retele cu MTU diferite - sa spunem pt un anumit pachet, de o anumita dimensiune, unde se face fragmentarea si unde se face reasamblarea (de ce?)
        - Fragmentarea
            - se face in momentul in care un pachet trece de la o retea cu MTU mai mare la o retea cu MTU mai mic decat dimensiunea pachetului respectiv
            - acest proces apare fie la sursa, fie pe parcursul traseului

        - Reasamblarea
            - se face intotdeauna la destinatia finala a pachetelor
            - motiv: optimizare astfel incat sa nu se faca fragmentari si reasamblari succesive ale aceluiasi pachet pe traseul de la sursa la destinatie
    - Dacă un pachet de 1800 de octeti trece printr-o rețea cu MTU=532 explicati câte fragmente se creeaza și care este dimensiunea fiecaruia.
- ***Adresa IP***
    - Fiecare gazdă şi ruter din Internet are o adresă IP, care codifică adresa sa de reţea şi de gazdă **aici @cami ->netid si hostid
    - Toate adresele IP sunt de 32 de biţi lungime şi sunt folosite în câmpurile Adresă sursă şi Adresă destinaţie ale pachetelor IP
    - “o adresă pentru fiecare rețea fizică separată” => folosire ineficienta a spatiului de adrese =>
        - subretele (o retea clasa B este impartita in mai multe subretele apropiate geografic)
        - adrese fara clase (= CIDR)
    - tabele de dirijare + algoritm de forwarding ip
    - clase de adrese:
        1. Clasa A
            - bitul cel mai semnificativ este 0
            - urmatorii 7 biti identifica reteaua
            - ultimii 24 de biti reprezinta adresa locala
        2. Clasa B
            - bitii cei mai semnificativi sunt 10
            - urmatorii 14 biti identifica reteaua
            - ultimii 16 de biti reprezinta adresa locala
        3. Clasa C
            - bitii cei mai semnificativi sunt 110
            - urmatorii 21 biti identifica reteaua
            - ultimii 8 de biti reprezinta adresa locala
        - **De ce au aparut clasele de adrese?**
            - pentru a prelungi durata de viata IPv4
        - **De ce nu se mai folosesc?**
            - risipeau prea multe adrese IP
        - adresele 127.ceva = adrese de loopback
- *ARP (Adress Resolution Protocol)*
        - face maparea intre adresa de protocol si adresa hardware
- *ICMP*
    - ICMP foloseşte IP ptr transmisie & IP foloseşte ICMP pentru raportare de erori
    - Mesajele ECHO REQUEST şi ECHO REPLY sunt folosite pentru a vedea dacă o anumită destinaţie este accesibilă şi activă.
    - ping trimite ICMP Echo şi aşteaptă un timp răspunsul (test accesibilitate)
    - traceroute trimite serie de datagrame cu valori TIME TO LIVE crescătoare şi primeşte mesaje ICMP Time exceeded din care extrage adresa ruterului

    - (Cum functioneaza traceroute?) - vezi poate e ceva pe net sau in lab ca am implementat o atunci
    - Path MTU = MTU minim pe o cale
        - **Cum se afla?** se trimit echo-uri de icmp din ce in ce mai scurte pana ce nu se mai primeste o eroare, moment in care s-a ajuns la path mtu
- Incapsulare ethernet -> ip -> icmp -
- Incapsulare ethernet -> ip -> tcp/udp
- *CIDR (Classless InterDomain Routing)*
    - alocă spaţiul de adrese IP în blocuri de lungimi diferite, fara a tine cont de clase
    - fiecare intrare din tabela de rutare este extinsa cu o masca de 32 de biti
    - rol masca: aflarea adresei de retea
    - cand soseşte un pachet IP, se extrage adresa IP a destinaţiei
    - forwarding: tabela de rutare este scanată intrare cu intrare, mascând adresa destinaţie şi comparând cu intrarea din tabelă, în căutarea unei potriviri (slide 35 curs)
    - Longest Prefix Match: Este posibil ca mai multe intrări (cu măşti de subreţea de lungimi diferite) să se potrivească, caz în care este folosită cea mai lungă mască.
- Clase vs CIDR (De ce este CIDR mai eficient?)
    - Rutarea este mai complicata la CIDR (ptc lipsesc clasele)
    - In cazul CIDR exista o singura tabela de rutare pentru toate retelele in care este precizata si masca de retea. Din acest motiv, este permisa unirea retelelor vecine, reducand dimensiunea tabelei de rutare => eficienta CIDR (*cred* - Wikipedia)
- **NAT - Translatarea adreselor de retea**
    - O adresă este asignata pentru mai multe calculatoare
    - Foloseşte adrese locale (private sau non-rutabile)
    - NAT translatează între adresa privată şi o adresă globală
    - **Transmisie** (pachetul iese din reteaua locala si trece prin boxul NAT)
        - înlocuieşte adresa IP locală cu o adresă IP globală
        - memorează (in tabela de translatare) corespondenţa şi număr port
        - inlocuieşte număr port cu index în tabela translatare
        - recompune sumele de control IP şi TCP
    - **Receptie** (pachetul vine din exterior si intra in reteaua locala)
        - obţine număr port din pachet (= index în tabela translatare)
        - extrage adresa IP locală şi număr port
        - inlocuieşte adresa IP şi număr port din pachet
        - recalculează sumele de control IP şi TCP
- IPv6 vs IPv4
    - https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_71/rzai2/rzai2compipv4ipv6.htm#rzai2compipv4ipv6__compfragments
    - IPv4 are ardrese de 32 de biti - Nu este suficient pentru ca toate dispozitivele conectate la internet sa aiba o adresa ip => descoperirea metodelor de patching (NAT, CIDR)
    - CIDR nu a fost suficient => solutia: adrese mai mari

    - IPv6 are adrese mai mari (128 biti = 32x4) = adresa sursa + adresa destinatie
    - header: IPv6 are antetul mai simplificat (7 campuri fata de 13 la IPv4)
    - Diferenţe circuit virtual:
        - 2 fluxuri cu aceeaşi etichetă se diferentiaza prin adr sursă + adr dest
        - aceeaşi pereche sursă+dest poate avea mai multe fluxuri
    - IPV4 nu tinea cont de serviciu, nu facea dif intre anumite situatii (streaming,securitate)
    - fragmentarea
        - la IPv4 fragmentarea se face pe parcurs
        - la IPv6 fragmentarea se face mereu la sursa
        - IPv6: fiecare fragment are Fragment Extension Header (are un un fragment part, apoi fiecare frag are fragmentation header)
        - IPv6: ruterele ignora datagramele mai lungi decat MTU => descoperă path MTU
        - reasamblare la destinatie pt ambele
    - Adrese IPv6 cu 96 zerouri prefix sunt interpretate ca adrese IPv4
- *Dirijarea*: distance vector si link state
- Distance vector
    - Fiecare nod trimite periodic vecinilor sai o lista cu distanțele de la el la celelalte noduri
    - situație care ajunge la problema numărării la infinit - să explicăm metodele de rezolvare a problemei (dc funcționează + cum) slide 61 în curs 5
        - split horizon
        - split horizon with poison reverse
    - de explicat o tabela de rutare
    - ce se intampla cu update-urile la dist vector
    - ce se int cand pica un nod => ce se intampla cu tabela de rutare
    - ce se int cu update-urile trimise între vecini la distance vector
- *Link state*
    - Presupune ca fiecare nod poate găsi legăturile cu vecinii și costul fiecărei legături
    - Exista un link state packet care este trimis prin flooding.
- Explicație cand este mai eficient un protocol de tip link state fata de un protocol distance vector
    - Acea inundare de la început este ineficienta si poate sa consume banda;
    - Se limitează într-un fel dimensiunea rețelei, ptc dacă este o rețea foarte mare atunci acea inundare va fi și mai costisitoare și va ocupa mai multe pachete;
- Explicați cum ați rezolva problema sa nu se propage prea mult un pachet, dacă de exemplu ajunge într-o bucla.
    - Avem un counter pentru fiecare pachet care se incrementeaza la fiecare retransmisie. În momentul în care ajunge la o anumită valoare maximă se da drop la pachet. Metoda prin care se limitează dimensiunea maximă a unei rețele.
- AS-uri (mai mult întrebări conceptuale)
    - Cum ați proiecta o anumită rețea?

    - Dacă aveți o rețea a universității noastre cum ați proiecta un backbone, cum ați proiecta împărțirea pe facultăți și unde ați pune zonele, routerele etc?
- BGP
    - vectorii distanțelor conțin și rutele spre destinație nu numai distanțele

6 TRANSPORT

- **Funcție:** transfer de date de la aplicațiile unui host la alt host, comunicare între app, puncte de aces, se transfera siruri de bytes
- **Diferenta TCP și UDP.**
    - TCP
        - orientare pe conexiune
        - garantează o livrare sigură și în ordine a datelor
        - folosit de cele mai multe protocoale de transport
        - orientat pe flux de octeți (senderul trimite un șir de bytes -> buffer (segmentare) -> trimitere în rețea) => încapsulare pachete => flexibilitate (slide 10 curs 6)
        - controlul congestiei
        - full duplex (pot sa trimit date în același timp)
        - three-way handshake - stabilire sigură a conexiunii
        - eliberare lină a conexiunii – fără pierdere de date
    - UDP
        - fără conexiune (trimit date fără sa am o conexiune stabilită în prealabil -> trimit pachete independente)
        - livrează datagrame (user datagrams)
        - pachetele nu sunt verificate; nu se asigura dacă ajung la dest corect
        - Best effort – datagramele pot fi pierdute, primite in alta ordine
        - sume de control pt integritate
- Ce este un port? Ce range poate avea? Pot sa am mai multe aplicații care folosesc acelasi port în același timp?
    - Port = prin el se identifica aplicația de pe calculatorul respectiv cu care vreau sa comunic
    - Porturile sunt numere pe 16 biti => valori de la 0 - 65535; 0 - 1024 sunt porturi rezervate (ele sunt folosite de protocoale f bine cunoscute)
    - Nu pot sa am doua aplicații pe același calculator care sa folosească același port în același timp ptc aplicațiile trebuie sa fie identificate unic. Porturi duplicate => nesiguranța (nu știu exact către cine sa trimit datele)
- **Socket =** punctul în care procesul de aplicație se atașează la rețea, standard pt comunicare între procese; asemanator cu un fisier (descriptor)
- **Server -> UDP**
    - socket
        - creeaza socket și îi alocă resurse de sistem
    - bind

        - asociază un socket cu un port pe masina locală (<port, adresa_IP>); practic dau adresa locală în structura sockaddr, îi pun un port și apelez bind => se asociază socket-ul pe care l-am dat ca prim parametru cu <port, adresa_IP>
        - după ce fac bind, toate datele care ajung pe calculator, pe acel port, vor ajunge în socket-ul creat
        - dacă fac bind și deja exista o aplicație care folosește acel port pe masina locală, automat bind-ul va întoarce eroare (returnează valoare negativă)
    - recvfrom
        - funcția pentru primire de date (primește mesaje de la un socket aflat la distanta); asemănătoare cu funcția de citire din fișier, careia ii mai dau niște flag-uri, o structura de adresa și o lungime a adresei (parametri de ieșire, care îți vor spune după ce a venit un pachet de la cine am primit pachetul respectiv)
        - recvfrom blocată pana vine pachetul, primește payload
    - shutdown
        - oprește trimiterea sau/și recepția de date
    - close
        - închide socket - termina utilizarea socket-ului și eliberează resursele alocate
- Este vreo diferenta între a da shutdown (RDWR) sau direct close?
    - Dacă dam shutdown putem sa dam după aceea un alt bind. Dacă am dat close este invalidat complet acel file descriptor, daca doar am dat shutdown este oprit socketul dar resursele respective și file descriptor-ul încă exista, doar este inactiv.
- **Client -> UDP**
    - socket -> analog server
    - sendto
        - trimite mesaj la un socket aflat la distanta; fata de server structură de adresa și lungimea ei nu mai sunt parametri de ieșire, ci trebuie sa ii completez eu
    - shutdown -> analog server
    - close -> analog server
- **Server & Client ->TCP**
    - creare, send, recv – au ca param socketul ca sa știe pe ce canal se trimite, întorc nr de bytes trimiși/primiți
    - pe server: bind, **listen –** coada de așteptare, accept
    - client: connect
    - pe un socket se primesc date cu unul se face conexiunea
    - shutdown (parțial) și close (eliberează resursele din kernel)

- TCP

- **orientat pe conexiune**
- SSH
- livrare sigura și în ordine
- octeți în buffer + se trimit segmente
- interfata flux
- full duplex
- asem cu comunicatie telefonica
- protocol cu confirmare (asem cu fereastra glisanta)
- antet
    - mai complex, sequence, ack, flux de segmente trimise unul după altul
    - urgent pointer info extra, urgență
    - se poate limita banda de transmisie (kb/s)
    - flags, checksum ca la UDP
- Cum funcționează **three way handshake-ul**? De ce? Cum se face crearea conexiunii cu acele pachete de syn și cum se face eliberarea?
    - **caz normal** -> 3 pași
        - Gazda 1 alege un număr de secvenţă x şi trimite un CONNECTION REQUEST care conţine x gazdei 2. Gazda 2 răspunde cu CONNECTION ACK, confirmând x şi anunţând numărul său iniţial de secvenţă, y. În cele din urmă gazda 1 confirmă alegerea lui y gazdei 2 în primul mesaj de date pe care îl trimite.
        - client – server (H1 – H2 in lab)
        - H1 trimite cerere de conex către H2: pachet tcp cu SYN setat și SEQ x
        - H2 trimite ACK: pachet cu SYN setat și ACK = x + 1
        - H1 trimite SEQ x+1
    - **coliziune**
        - 2 hosturi vor sa înceapă o conexiune în același timp
    - **rejectarea conexiunii**
        - SYN întârziat, se folosește flagul de RST – erori și rejectie conex
        - dacă are ACK greșit → RST sau acceptata (f rar)
    - **deconectare / eliberarea conexiunii**
        - abruptă
            - pachet cu FIN setat, H2 da drop dacă mai primește ceva
            - poate genera pierderi de date
            - După stabilirea conexiunii, gazda 1 trimite un TPDU care ajunge corect la gazda 2. Gazda 1 mai trimite un TPDU dar, înainte ca acesta să ajungă la destinaţie, gazda 2 trimite DISCONNECT REQUEST . În acest caz, conexiunea va fi eliberată şi vor fi pierdute date.
        - normala
            - H1 trimite FIN, H2 trimite kind of ACK cu FIN, H1 trimite ACK (twh)
            - După stabilirea conexiunii, gazda 1 trimite un TPDU care ajunge corect la gazda 2. Gazda 1 mai trimite un TPDU dar,

înainte ca acesta să ajungă la destinaţie, gazda 2 trimite DISCONNECT REQUEST . În acest caz, conexiunea va fi eliberată şi vor fi pierdute date.
    - utilă atunci când fiecare proces are o cantitate fixă de date de trimis şi ştie bine când trebuie să transmită şi când a terminat
    - pot apărea probleme, toate țin cont de timeout
- **corectitudine & corectie**
    - checksum: transmisie ca UDP, recepție – dacă e 0 e ok (slide 34)
    - ACK number: corecția se face prin retransmisie
        - x + 1 = SEQ + 1 (ce trebuie primit), dacă trimit ACK x înseamnă ca tb sa retransmita pachetul x
    - pentru selective repeat (fereastra glisanta) e mai complicat: se trimite ACK + index fereastra
- Cum funcționează **controlul fluxului de date**? -> în funcție de buffer
    - dimensiuni fixe de buffere
    - transmițătorul nu trebuie sa furnizeze date mai rapid decat le poate accepta receptorul
    - WIN = nr pe 16 biti care spune senderului cat mai poate receiverul sa primeasca
    - ACK = poziția în setul de date, unde a rămas
    - dacă bufferul e plin, senderul mai poate trimite doar date urgente (ttl) sau 1 seg de 1 byte care verifica dacă mai are loc + dacă anuntul precedent al receptorului s-a pierdut
    - senderul va trimite cand se eliberează bufferul
- **probleme TCP**
    - durata depinde de viteza de transmisie: 2^32 bytes maxim se pot trimite
    - pot apărea segmente diferite cu același SEQ (de la viteza) **REZOLVARE** cu flaguri
      TCP Timestamps – asociază fiecărui seg un timestamp (dacă 2 pachete au același seq și același timestamp sunt duplicate)
    - WIN size de 16 biti = 64 KB → transferuri mici → nu se foloseste tot canalul și irosesc banda→ ar trebui mărit WIN **REZOLVARE** Window Scale (din header) : WIN va fi WIN*factor de scalare → voi avea ferestre mai mari
- **CONTROLUL CONGESTIEI**
    - în funcție de banda = păstrarea nr de pachete sub o anumită limită ai sa nu fie afectată performanța
    - capacitatea rețelei = fereastra de congestie
    - congestia apare cand numărul de pachete e aproape de capacitatea rețelei
    - **algoritm de stabilire fereastra de congestie**:
        - transmite un seg de dim max
        - dublează vol de date la fiecare transmisie (rafala)
        - la primul timeout se oprește procedeul ai fereastra va ramane la valoarea ultimei transmisii confirmate
    - **algoritm de control al congestiei:**
        - folosește un threshold
        - la un timeout pragul e setat la jumate din fereastra

        - crestere exponentiala pana la el, dupa liniara
        - **TCP Slow Start (estimare)**
            - se dorește creșterea rapidă a ratei de transmisie
            - fereastra de congestie crește exponențial pana la primul pachet pierdut
            - rata de transfer e foarte lenta la început dar crește exponențial la fiecare RTT
- **EVITAREA CONGESTIEI**
    - **crestere liniara:** cand dimensiunea ferestrei a ajuns la pragul pentru slow start, dim ei va crește liniar (incremental) pană se declanșează un timeout
    - **scadere multiplicativa:**
        - dacă a avut loc timeout se reia fereastra de la val 1 si se reporneste slow start
        - la primirea a 3 ACK duplicate, se reduce la jum fereastra si porneste cresterea liniara (e posibil sa nu fie atat de mare congestia pt ca se pierd pachete)
    - The TCP Sawtooth: grafic viteza de la crestere + evitare
    - rata de congestie nu e stabilă în timp, depinde de multe (interferențe, rețea)
- **gestiunea ceasurilor în TCP, unde pun timeout-ul?**
    - îl aleg în funcție de round trip time = trimis de la H1 la H2 + primit ACK, nu e stabil
    - grafice slide orientative pt o conexiune
    - distribuție de probabilitate, unde așez timeout-ul?
    - **METODE:**
        - la fiecare ACK actualizez RTT în funcție de vechiul RTT, aleg alfa si beta ai sa nu varieze RTT rapid
        - alegerea se va face după deviația standard(cat de departe e fata de medie) și vechiul RTT
- Concluzie: estimare timeout si congestie → metode empirice: mai masor, mai ajustez
- **The Real Time Transport Protocol:**
    - UDP cu avantaje de TCP dar și cu latenta mica și viteza mare, pentru fluxuri unde nu e nevoie de retransmisie dar e nevoie de ordine (nr secvența), tratarea pierderii unui pachet se face la aplicație (video)
    - peste UDP
    - **header:** timestamp, synchro - identifica sursa de date, padding, extension

- UDP

- **fără conexiune, trimit datagrame**
- best effort (încearcă, dar nu e garantat ca reuseste sa trimită datele la destinatie)
- sume de control pentru integritate
- se foloseste cand am app care au latenta mica si se vor rapide, nu sunt greoaie, nu conteaza corectitudinea: DNS, Voice over IP
- antet

    - **header** simplu, nu asigura control erori și control flux, checksum 16 biti
    - nu avem adrese IP pt ca cu asta se ocupă nivelul rețea
    - De ce nu apare adresa IP în antetul UDP?
        - Deoarece se ocupă nivelul rețea ca pachetul sa ajungă la adresa IP destinatie (avem incapsulat protocolul UDP în cel IP).

7 DNS

- Nivelu**l aplicație,** 1985 (înainte se folosea host.txt care nu era structurat, era greu de folosit, acum e folosit la nivel local acest host)
- DNS folosește în general protocolul **UDP** pe portul 53, dar, în cazul răspunsurilor de dimensiuni mai mari sau pentru operații ca transferul de zone, se utilizează și TCP
- **De ce e nevoie de DNS?** - ca sa nu retina adresa IP pentru fiecare domeniu,
- Numele sunt mai ușor de reținut decât adresele IP
- DNS: asociere între adresa domeniului și un nume simbolic (de tip n la n)
- URL: protocol// host port# path
- **Funcționare:** browserul trimite o cerere DNS către domeniul respectiv pentru a primi IP ul
- De ce **spațiul de nume are structura arborescentă**?
    - Conceptual, Internetul este divizat în peste 200 domenii de nivel superior, fiecare domeniu cuprinzând mai multe sisteme gazdă. Fiecare domeniu este partiţionat în subdomenii şi acestea sunt, la rândul lor, partiţionate ş.a.m.d. Toate aceste domenii pot fi reprezentate ca un arbore. Frunzele arborelui reprezintă domenii care nu au subdomenii (dar, bineînţeles, conţin sisteme). Un domeniu frunză poate conţine un singur sistem gazdă sau poate reprezenta o firmă, deci să conţină mii de sisteme gazdă.
    - rădăcina → nivel înalt (administrate de **ICANN, imediat după radacina, de ex in web.wap.com e .com)** → de nivel 2 … → frunzele (toate pot avea subdomenii)
- **DOMENIU** = orice drum din arbore slide 10 exemplu support.microsoft.com
- Daca avem un anumit domeniu cs.curs.pub.ro sa spunem care este top level domain, care e subdomeniul, subsubdomeniul etc
    - top level domain = ro
    - subdomeniu = pub.ro
    - subsubdomeniu = curs.pub.ro
    - subsubsubdomeniu = cs.curs.pub.ro
- **Zone DNS** = zone administrate de servere de nume distincte
- **Componente DNS**
    - **spațiu de nume** (namespace)
        - organizat ierarhic
        - fiecare nod are asociat un set de info în BD
    - **servere dns**
        - administrează zonele DNS (fiecare subarbore e administrat de un server DNS anume, spre **exemplu** .ro stie doar de pub.ro nu trebuie sa stie si de cs.pub.ro)
        - păstrează **BD** cu info necesare clienților

        - în înregistrări de resurse (resource records)
- **Înregistrări de resurse**
    - **BD** : conține înregistrări numite **Resource Record notat cu RR mai jos**
        - în format text, nume, ttl, clasa, tip, valoare
        - tipuri: A(32 biti), AAAA(128), MX, TXT
- **Serverul de nume** – pastreaza info pt domeniul pe care-l administrează și cunoaște adresele altor servere
    - datele din înregistrări RR sunt păstrate în Master files
    - primește cereri de la un Resolver și îi da răspuns
    - Resolverul are un cache pentru valorile IP **(AVANTAJ),** perioada de cache e data de un TTL din RR -- mai jos rubrică de probleme & avantaj
- **Pot exista mai multe adrese IP asociate unui domeniu sau aceeași adresa IP poate avea mai multe domenii**
- **Protocolul DNS**
    - La apelul unui client, Resolver trimite DNS request și primește de la server un DNS reply
    - software: host, dig, gethostbyname
- **Format mesaje:** header (dacă e ans sau qs), qs (intrebarea pt name server), ans (RR ul coresp intrebarii), authority, additional info
- **Cum funcționează DNS?**
    - DNS e server autoritate pentru numele gestionate (cum afla orice IP din internet)
    - dacă numele apelat e gestionat de el, răspunde direct, altfel forwardeaza cererea la alt nameserver care poate știe domeniul căutat (astfel incat sa ajungă la serverul autoritate pentru acel nume și răspunsul sa fie trimis pe calea inversa)
    - dacă e o cerere care a mai fost deja, se uita în cache
- **Probleme: cache-ul nu e updatat rapid ->** nu se notifică atunci cand se schimba vreo adresa IP, deci singura metoda este sa expire cache ul (aproximativ 1-2 zile) **SOLUȚIE** temporară sa se pastreze și adresa veche și cea nouă pentru a nu pierde vreun request
- **Avantaje**: se creeaza **cache** (nu se mai trimite pana la cel autoritate, se uita mai intai în cache sa vada daca are acea info acolo)
- **Caching-ul de la DNS** -> Pot sa am cache-ul de dns care pastreaza ultimele requesturi
    - dacă un req este deja în cache -> se rasp direct din cache
    - o intrare în cache are o durata de viata limitata (ttl)
- **Cum se trimit requesturile DNS?**
    - **forward -> rezolvare recursiva**
        - de la unul la altul pana la cel care are autoritate (ai mai sus toata explicatia)
        - fiecare server care nu are informaţia cerută o caută în altă parte şi raportează
        - răspunsul este trimis pe calea inversa
    - **rezolvare iterativa**

        - dacă serverul DNS nu poate rezolva întregul nume, el trimite clientului partea nerezolvată și adresa serverului DNS care o poate rezolva
        - nu se mai populeaza cache-ul
- **Cereri inverse REVERSE DNS**
    - funcționare inversa, am deja IP, vreau sa aflu numele asociat
    - pentru asta avem un domeniu special **in-addr.arpa**: cand Resolverul face o cerere de reverse DNS la o adresa IP o concateneaza la acest domeniu: ip.in-addr.arpa
    - serverul de nume cauta in RR sale acea adresa IP
    - se face în înregistrări de tip POINTERI
- **Replicarea serverelor DNS**
    - dacă avem domenii f aglomerate: avem mai multe servere care administrează același domeniu (era chiar un exercițiu în lab 9 la dig)
    - server primar: aici se fac toate modificările înregistrărilor, folosind Master files
    - server secundar (pot fi mai multe): primește info de la cel primar, aici se propaga modificările de la cel primar, se trimit requesturi între aceste servere pentru a păstra aceeași BD
    - **transfer toata zona:** cerere SOA de la secundar, în răspunsul SOA de la master server dacă serial number e mai mare decat cel local înseamnă ca exista modificări, trebuie actualizat BD ul, primește înapoi toata zona
    - **transfer incremental:** cerere SOA, răspuns SOA, master serverul trimite doar modificările corespunzătoare serverului secundar în BD
    - **notificari:** master serverul trimite notificări către toate serverele secundare, apoi server-ul secundar trimite cerere SOA și primește înapoi răspuns SOA

8 WEB

- **WWW**
    - Set de documente (pagini) cu legături între ele (hyperlinks)
        - Distribuite pe masini diferite
        - Include o pagina de referinţa
    - Funcționare (slide 5)
        - afișare pagini web (publicare și transmitere)
        - clientul realizează o conexiune TCP cu un server
        - după realizarea conexiunii TCP, clientul trimite către server un HTTP request
        - serverul poate răspunde cu pagini web
        - **sistem distribuit :**mai multe servere care se pot ocupa de mai multe domenii
- **trei elem de baza**
    - protocolul **HTTP (toate detaliile sunt mai jos)**
    - schema de adresare a documentelor: **URL**
        - protocol://host[*port]/path
        - parametrii structurați astfel nume=CAMI&parola=navem, cookie
        - anchor: marcaj de poziție într-o pagina

        - mailto, telnet, file
    - un limbaj de formatare a documentelor: **HTML**
        - marcaje specifice pentru text, imagini, formulare < >
        - formulare: nume=CAMI+MIRCIA&nota=10

PROTOCOLUL HTTP = protocol pentru transportul mesajelor specializate prin rețea

- port default 80
- HTTP este încapsulat în TCP
- **proces**:
    - browserul determina URL-ul apoi ip-ul prin DNS
    - deschide o conex TCP
    - trimite o comanda
    - conex TCP e închisă
    - conținutul este afișat (pentru fiecare element nou se mai creeaza o conexiune - de ex, imagini, animatii)
- **optimizare**: HTTP 1.1 o **singura** conexiune **persistentă** unde se trimit prin pipeline mai multe cereri HTTP, conexiunea se închide după ce se primește răspuns pentru cereri (răspunsurile pastreaza ordinea cererilor din coada)
- **protocol stateless** -> dacă se trimit mai multe cereri, acestea sunt independente între ele (nu trebuie sa primesc răspuns la prima ca sa o trimit pe a doua)
- folosește paradigma **request/response** în format text
- nu mai avem header strict, ne axam pe nivel aplicație
- **request contine**
    - metoda calea_catre_resursa_dorita /HTTP/nr_versiune
    - headere
    - body
    - metode GET/POST/REQUEST + host
- **response contine**
    - pe prima linie HTTP/versiune ca o confirmare ca se folooseste HTTP + status-code + mesaj
    - headere
    - body
    - răspunsul pentru o cerere de tip GET contine headerele Content-Type și Content-Length. Pentru un Content-Type de tip text/html, conținutul corpului răspunsului este de tip html.
- **metode HTTP**
    - **GET**
        - cerere de citire a unei pagini.un fișier de pe server, cea mai simpla
        - include și ce a completat utilizatorul (ca exemplul de la forms)
        - server-ul genereaza pagina și o trimite
        - request și response arată ca mai sus
    - **HEAD**
        - Cerere de citire a antetului unei pagini de Web
        - asemanator cu GET, răspunsul conține doar header-ul
        - request și response ca la POST
        - **ce e diferit:** răspunsul serverului conține doar headerul, fără conținutul paginii

        - de ce am nevoie? ca sa vad daca s-a modificat o pagina Last Modified
    - **POST**
        - adăuga ceva la o resursa specificata (de ex o pagina de Web)
        - request: cer doar resursa și toți parametrii sunt în body nu în URL, **avantaj ->** sa nu apara toți parametrii în clar in URL (password)
        - response: ca la GET
    - PUT - trimite o resursa pe server
    - DELETE - șterge ceva
    - TRACE - face serverul sa raspunda cu cererea care a sosit
    - OPTIONS - interogare opțiuni
    - CONNECT - conectare prin proxy
    - coduri de stare pe slide CTRL F “coduri de stare” , 4 și 5 sunt erori
- antete mesaje CTRL F “antete mesaje HTTP”
- **conținut answer:** mai multe tipuri - content-type, multipart **(optim)** cu delimitator - string random numit boundary
- **cache!!**
    - se păstrează conținutul paginii în cache, dacă știu ca pagina nu s-a schimbat
    - **CONSISTENTA CACHE-ULUI = CUM MĂ ASIGUR CA CE AM ÎN CACHE E OK (Asigura ca documentul din cache este același cu cel din server)**
        - **metoda 1:** se verifica head-ul cu comanda HEAD și se compara Last Modified cu data înregistrării în cache, transmite GET dacă doc din server e mai nou decat copia din cache
        - **metoda 2:** GET cu campul If-Modified-Since in antet - dacă nu e modificata înainte de data aia răspunde cu 304 Not Modified, altfel, dacă s-a modificat, răspunde normal la GET
        - **soluție pt performanța:**
            - răspunsul serverului sa conțină o data de expirare -> camp EXPIRES
            - Clientul verifica existenta paginii în cache
                - dacă nu exista pagina - cere resursa necondiționat
                - dacă e expirata - adaugă la cerere antet If-Modified-Since (– dacă server răspunde cu 304 Not Modified folosește intrarea din cache)
                - dacă nu e expirata - folosește intrarea din cache
    - 3 tipuri de caching:
        - la client – cache privat
        - la proxy, server – cache-uri partajate
    - controlat de câmpul **cache-control** trimis de server către client, spune unde poate fi păstrată pagina în cache
        - private: pagina poate fi păstrată de browser în cache-ul sau nu în shared caches
        - public: nicio restricție în caching
        - no-cache: nu poate fi păstrat în cache
    - Dacă ni se da cache control : private, public ce influență are, de ce am vrea la un moment dat sa fie private și nu public? Ce restrictii are un browser când primește o pagina care are cache controlul de private?

pai e fixx definitia lor nu? eleno?
pai si dc vrem private si nu public?
pentru ca ala shared e folosit de mai multi clienti, cand alegi private ii zic ca vrei cache ul tau propriu
https://my.kualo.com/knowledgebase/109_litespeed-cache/1357_public-cache-vs.-private-cache.html

- **autentificare și autorizare**
  - protejez anumite pagini prin user si password
  - **acțiuni**
    - clientul cere resursa restrictionata
    - serverul răspunde cu 401 pt ca trebuie sa se autentifice
    - apare campul WWW-Authenticate
      - Basic = doar user & password
      - realm = mediul în care am acces ca sa nu pun mereu user & password (domeniul protejat)
    - serverul verifica credențialele de autorizare și satisface cererea (sau refuză cu 403)
  - user și parola sunt trimise **în clar** -> am nevoie de HTTPS
  - **suport de sesiune: COOKIE = mecanism de transmitere de informații de stare prin HTTP, se asigura o sesiune = următoarele cereri conțin un id de sesiune ca sa nu se mai autentifice la fiecare cerere**
    - inițiată de server prin Set-Cookie
    - acceptata de client prin antetul Cookie
    - dacă au flag de **secure** trebuie trimise prin HTTPS
- schema browser - generare cereri CTRL F ca e un desen cu pașii
- **generator cereri** verifica dacă e URL absolut (cale completa, incepe cu /) sau relativ (porneste de la folder curent)
- diverse exemple de cereri (slides)
- tot parcursul cache-cookie-sesiune (sumar, cum se leaga)
- diverse exemple cu coduri de răspuns de la server

9 EMAIL

- **Arhitectura sistemului de e-mail**
  - agent utilizator: interfața care permite citire și scriere de mesaje
  - agent de transfer
    - transmite email-uri de la sursa la destinatie
    - agentul client preia un mesaj, stabilește conexiune cu serverul și îi transmite mesajul
    - agentul server primește mesajul și îl plasează în cutia poștală
  - agenți = demoni de sistem care rulează pe fundal, lol ok
- Structura mesaj -> CTRL+F “envelope”
  - envelope = informatia necesara pt transportul mesajului, folosita in protocolul SMTP
  - header = informația de control pentru agentul utilizator

        - perechi nume-valoare referitoare la utilizatori și la conținutul mesajului
    - body = informația destinata utilizatorului
        - text sau multimedia
- MIME - encodare pentru atasamente
    - toate mailurile trebuie sa fie format ASCII (ca să poată fi afișat în terminal), encodare base64 (pentru SMTP normal)
    - boundary, formatari ca la HTTP (vezi mai sus la web)

- **protocolul SMTP**
    - protocolul standard de aplicație prin care se livrează mailul de la sursa la destinatie
    - folosește **TCP** și un schimb de mesaje text între client și server
        - comenzi (4 caractere = comanda + parametri) și răspunsuri (nr din 3 cifre urmat de text)
    - oferă o livrare sigură a mesajelor (conexiune TCP)
    - verifica numele unui utilizator
    - **cum funcționează o sesiune** (slide 14)
        - se stabilește o conexiune TCP intre client și server
        - clientul trimite un HELO
        - serverul trimite un cod
        - clientul trebuie sa dea MAIL FROM (de la cine) și RCPT TO (pentru cine)
        - clientul spune DATA, așteaptă 354 de la server și apoi scrie tot corpul mailului (header, body, conținut)
        - la final pune un punct singur pe o linie ca sa anunte ca a terminat
        - server da 250 și impacheteaza
        - se închide conexiunea
    - **problema:** nu avem securitate, nu se cere autentificare, se trimit în clar, nu se verifica daca adresa de la MAIL FROM este a clientului, deci putem trimite mailuri către oricine care sa para de la oricine :)))) xD
    - **rezolvare:** se folosește în spatele unor module de securitate
    - **email gateways:** server dedicat, asigura o adresa unică pentru toate mesajele dintr-un grup, se folosește un exploder pentru a trimite o copie a mesajului pentru fiecare adresa destinatie din lista
    - **livrare finală:** SMTP duce doar pana in server mesajul, se foloseste o conexiune temporara cu protocolae POP SI IMAP pentru citire
    - **POP3:** comenzi puțin diferite pentru autorizare la inbox CTRL F “POP3”, RETR citeste mailul, DELE marcheaza mailul pt stergere, se sterg la QUIT, nu pastreaza mailuri (trebuie descarcate), nu are foldere
    - **IMAP:** pastreaza mailuri, are folderele, pot fi accesate mailurile de oriunde
    - **dezavantaj:** POP3 și IMAP trimit in clar

10 FTP

- protocol vechi (făcut înainte de TCP) pentru transfer de fișiere între client-server sau 2 servere
- protocol independent de SO și de hardware
- permite listare directoarelor sau modificarea lor, se pot descarca
- drepturi pentru utilizatori
- oferă o linie de comanda cu utilizatorul
- în prezent se folosește cu interfața grafica
- **doua conexiuni TCP -> FTP are 2 porturi rezervate, separa transmisia de comenzi de transmisia de date**
    - **de ce avem doua conexiuni?**
        - implementare mai simpla pentru ca face deja separarea intre cele 2 metode de comunicatie
        - controlul congestiei și al fluxului pentru date fără sa afectez comenzile
        - nu mă obliga sa am conexiunea de date cu același server (permite existența unei conexiuni de comenzi client-server prin care se comanda un transfer de fișiere între 2 servere)
    - **de control**
        - transmite comenzi și răspunsuri FTP
        - USER PI- inițiază conexiunea și SERVER PI - generează comenzi (PI = Protocol Interpreter)
    - **pentru date (este detaliată mai jos)**
        - transmisie/recepție simultană
        - User DTP și Server DTP (Data Transfer Protocol)
- comenzi de 3-4 caract + parametri CTRL F “cateva comenzi FTP”, seamănă cu cele din Linux
- nu avem **securitate** nici aici, sunt datele trimise în clar (mai jos sunt metode de securitate)
- oferă autentificare, pentru securitate vin alte protocoale/alte versiuni care fac asta
- **răspunsuri FTP**
    - 200 cod OK
    - format: cod numeri & blanc & text descriptiv
- **cum arată o sesiune:** CTRL+F “acme mail” sau “O sesiune FTP”
    - se rulează ftp
    - se deschide conexiunea, port 21 default
    - serverul trimite un HELLO
    - clientul se logheaza, dacă e ok poate trimite comenzi asemănătoare cu cele din linux de control de fișiere (după logare, utilizatorul schimba directorul distant și listeaza conținutul)
    - la quit se închide
- **conexiunea de date (cum se face?)**
    - **metoda activa**
        - **serverul inițiază conexiunea de date**
        - **serverul activ se conecteaza la client**
        - clientul specifica pe conexiunea de control o adresa IP și un port
        - serverul inițiază conexiunea de date la **portul specificat de client** (clientul deschide socket și așteaptă cereri de conexiune)

    - **metoda pasiva**
        - **clientul inițiază conexiunea de date cu serverul (pasiv)**
        - clientul cere serverului sa asculte la o adresa si un port (nu cel standard, pe conexiunea de control)
        - **serverul alege și comunica** adresa și portul
        - clientul inițiază conexiunea de date la portul specificat de server
    - conexiunea de comanda e facuta de client in ambele cazuri
    - bucăți numerice de valori (nu trimite fix numărul portului, se trimit bucăți de 8 biti)
- **FTP PRIN NAT (folosirea modului pasiv)**
    - **în metoda activa: nu** e posibil pentru ca serverul FTP nu poate ajunge la rețeaua locală din spatele NAT-ului
    - **in metoada pasiva:** e posibil pentru ca serverul deschide socket și asculta, dar clientul inițiază cererea (acesta are o adresa IP locală)
- **transfer FTP intre 2 servere (folosirea modului pasiv)**
    - e posibil deoarece clientul nu trebuie sa dea propriul ip/port
    - clientul face 2 conexiuni de control cu 2 servere: A și B
    - B e în mod activ îi dau portul și adresa data de A cand initiez conexiunea
    - trec conexiunea cu serverul A în modul pasiv deci acesta îi va da un port și o adresa unde se asteapta conexiunea
    - concluzie: transmit unui server datele celuilalt server
- **securitate FTP**
    - FTP transmite în clar user & password deci are nevoie de metode de securitate
    - SSH - transmite password criptat
    - SFTP - securitate pentru date

11 CRIPTOGRAFIE

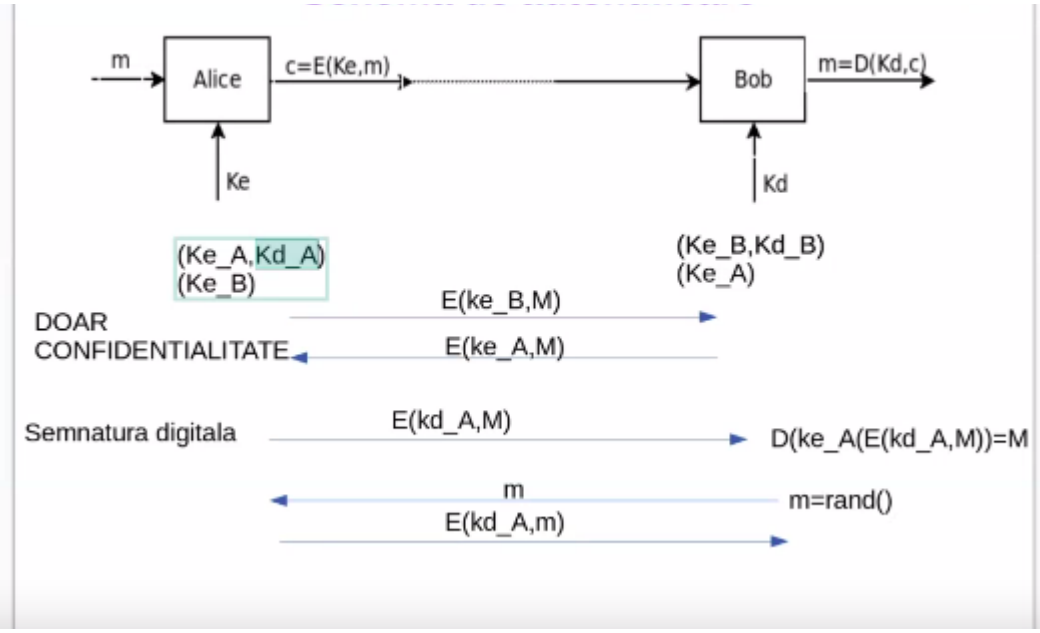
- https://ocw.cs.pub.ro/courses/pc/laboratoare/12
- **Scopul securității:**
    - confidentialitatea datelor = un atacator nu poate decripta mesajele pe care le interceptează, informația e disponibila doar utilizatorilor autorizati
    - integritatea datelor = informația poate fi modificata doar de utilizatorii autorizați sau în modalitate autorizata, dacă atacatorul introduce date false în textul criptat => oricine primește datele ar trebui să își poată da seama că au fost modificate
    - disponibilitatea = accesul la informație al utilizatorilor autorizați nu este îngrădit (opusul e denial of service)
- **Probleme derivate**
    - autentificarea = determinarea identității persoanei cu care schimbi mesaje înainte de a dezvălui informații importante
    - autorizarea (control accesului) = protectia împotriva accesului neautorizat
    - **non-repudierea =** transmițătorul nu poate nega transmiterea unui mesaj pe care un receptor l-a primit
- **Modelul de bază al criptării**

    - text clar M - la Alice -> criptare cu cheie de criptare K -> text criptat C -> decriptare cu cheie de decriptare K' -> text clar M - la Bob
    - notații
        - transformare : {M} X {K}-> {C} mesaje, chei, criptograme
        - operația de criptare C = E(k,M)
        - operația de decriptare M = D(k',C)
        - D(k',(E(k,M)) = M înseamnă ca criptarea e corecta
    - confidentialitatea = intrusul să nu poată reconstitui M din C (să nu poată descoperi cheia de descifrare K')
    - integritatea = intrusul să nu poată introduce un text cifrat C', fără ca acest lucru să fie detectat (sa nu poată descoperi cheia de cifrare K).
- **Scop Trudy** = găsirea cheii de criptare/decriptare (atacatorul)
- **Criptologie** = Criptografie (mod prin care se creeaza cifrari)+ Criptanaliza (spargere, găsire cheie pentru un text cifrat, mai multe metode in curs CTRL F "criptanaliza")
- **Ce presupune un algoritm eficient/bun?**
    - Criptare + decriptare rapide. Deși mereu va exista o scadere de performanta
    - Algoritmii sunt considerați publici și stiluri de toate lumea => criptarea trebuie sa depinde de chei, nu de algoritmi
    - Sa fie foarte greu sa obtii cheia
    - cerințe pentru confidentialitate/integritate curs CTRL F "cerințe criptosisteme"
- **Algoritmi simetrici/Criptare simetrică**
    - Se folosește **aceeași cheie K** și pentru criptare și pentru decriptare. (K - cheie privată) => Scopul atacatorului: găsirea cheii.
    - **Vulnerabilitate**: Alice și Bob trebuie sa cunoască cheia, dar nu si-o pot transmite pe o cale pe internet deoarece nu este considerată sigură.
    - istorici: Cifrul lui Cezar, Cifruri de substituție, Substituție polialfabetică, ROT13 (se sparg ușor, în curs au toti regulile)
    - reali/eficienti: Data Encryption Standard(DES), 3DES, Advanced Encryption Standard(AES)
    - 3DES -> soluție de urgență pentru DES
- **Algoritmi asimetrici**
    - Ke - criptare => cheie publica, oricine are acces la ea. Kd - decriptare, cheie privată
    - funcționare: Bob generează Ke, Kd, Alice folosește Ke și îi trimite lui Bob, doar Bob poate decripta cu Kd
    - rezolva vulnerabilitatea algoritmilor simetrici
    - asigura confidentialitatea datelor
    - **Probleme:**
        - conexiunea nu mai este full duplex: Alice -> Bob, Bob -> Alice. Avem nevoie de mai multe chei de criptare (avem nevoie de 2 perechi de chei ca sa facem bidirecțional)
        - Absolut oricine poate cripta date cu cheia publica. Bob nu are garanția ca datele vin de la Alice sau sunt criptate de un atacator => Se pierde partea de "autentificare" (integritate)
    - **Scop:** rezolva vulnerabilitatile algoritmilor simetrici, cheia de la algoritmii simetrici este trimisă folosind algoritmi asimetrici, astfel se ajunge la criptare

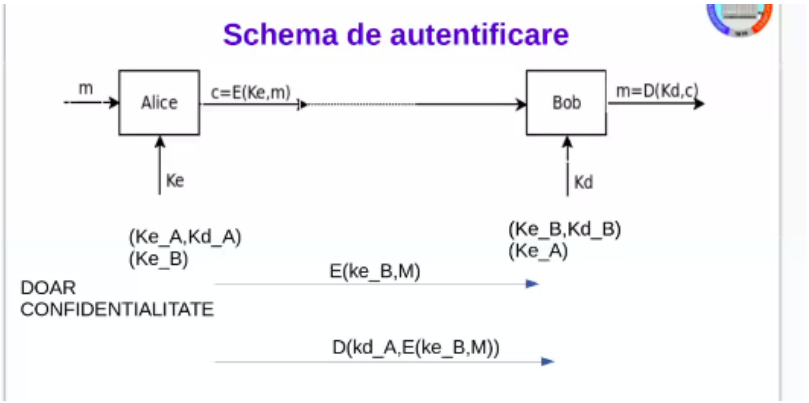
simetrică. **Flow:** Bob generează (Ke, Kd). Bob ---- trimite Ke----> Alice. Alice poate acum sa cripteze mesajul folosind Ke si cripteaza K cu Ke. Îl trimite K lui Bob criptată => Bob decripteaza => Acum putem folosi algoritmul de criptare simetrică, cheia se genereaza periodic

- acest model asigura doar confidentialitatea: doar B, care are cheia privată Db poate înțelege mesajul M, pentru integritate avem semnatura digitala

- **Semnatura digitala**(schema de integritate)
  - rezolva problema legată de **integritatea** mesajelor pentru criptarea asimetrica.
  - asigura și **autentificarea** (pot confirma 100% ca mesajul vine de la Alice)
  - Ke și Kd pot fi folosite interschimbabil. CTRL + F după “Schema de integritate (semnatura digitala)” pentru schema din curs
  - Alice generează Ke și Kd. Mesajele sunt criptate cu Kd (cheia privată) si Bob le poate decripta folosind Ke (cheia publica). Oricine poate folosi cheia publica Ke pentru a decripta mesajul. Nimeni nu poate modifica mesajul M deoarece nu cunoaște cheia privată kd
  - Alice criptează cu Kd_A și Bob decripteaza cu Ke_A și astfel demonstrează că a primit de la Alice
  - Bob poate trimite un challenge lui Alice într-un mesaj, dacă răspunde corect înseamnă că e ea (trimite același mesaj criptat cu Kd_A, dacă e semnat digital corect e ok)



- **Schema de autentificare**
  - CTRL F “ schema de autentificare”
  - se folosește cheia privată a lui Alice că sa se semneze digital ce s a criptat cu cheia publica a lui Bob și se trimite către Bob



- one time pad = la cifrare prin substitutie - polialfabetica tre sa se gaseasca o cheie de dimensiunea mesajului
- cifruri prin transpozitie, cifruri produs (exemple în curs)
    - https://www.dcode.fr/transposition-cipher
- **DES - simetric**
    - 16 iterații, cheie de 56 de biți, a fost înlocuit de AES
    - se împarte inputul în Left și Right
    - Right = XOR cu Left și o funcție între cheie și Right
- **AES**
    - **simetric**
    - matrice
    - mai multe chei de runda
    - nr de runde depind de lungimea cheii
- **RSA**
    - **asimetric**
    - functie exponentiala
    - CTRL F “Metoda RSA”
- **Diffie-Hellman key exchange:**
    - **asimetric**
    - Alice și Bob vor sa stabilească o cheie de criptare comuna fără sa își transfere acea valoare în clar prin canalul de comunicatie. => Transfer protejat de chei
    - Se bazează pe faptul ca unele **funcții sunt greu inversabile**. x => f(x). f(x) => este f greu sa aflăm f^(-1) (x) = x
    - Alice alege X, Bob alege Y
    - Alice trimite un mesaj care contine n și g; g - generator
    - Fie n și g numere mari. n prim, (n - 1) / 2 prim.
    - Alice -> Bob: g^x mod n = functia greu inversabila, chiar daca n si g sunt trimise în clar prin canalul de comunicatie; Bob -> Alice: g^y mod n
    - Acum atat Bob, cât și Alice pot calcula (g^x mod n)^y mod n ⇔ g^(xy) mod n. Analog y x inversate.
    - g^(xy) mod n = cheia de criptare ⇔ f greu de obtinut x
    - asimetrie: n g publice, x e privată pt A, y pentru B
- **Rezumatele mesajelor:**
    - **de ce?** e mai ușor sa semnezi digital rezumatul decât întreg mesajul

    - Rezumatul (**hash**) unui mesaj este un șir de biți de lungime fixă, generat cu ajutorul unei funcții de dispersie neinversabile aplicată mesajului
    - **Algoritm hash**: ușor de calculat MD(P) = hash code-ul = semnătura mesajului. (MD = message digest)
    - MD(P) - dimensiune fixa, P - absolut orice dimensiune
    - Metoda de a verifica integritatea mesajelor
    - MD sunt calculate pentru mesaje f mari. (Fisiere de GB si MD de 128 biti).
    - Rezistenta la **coliziuni**: o funcție H este rezistenta la coliziuni dacă este foarte greu sa se gaseasca a și b, a!=b astfel ca H(a) = H(b).
    - funcții hash
        - MD5: nu este rezistent la coliziuni => se folosesc SHA1 și SHA2 acum.
        - SHA 1 (considerat secure), SHA 2
    - criptarea nu e același lucru cu rezumarea!
- **Certificate digitale (la asta se folosesc semnăturile digitale):**
    - conținut: fișier public cu detalii despre proprietar
    - garantat de Certificate Authority
    - structura care asociaza identitatea cu cheia publica (contine cheia publica a serverului)
    - **Rol**: leaga cheia publica de un proprietar (principal) sau de un atribut. Un certificat permite autentificarea unei entități în cazul în care nu a mai fost vreun contact anterior.
    - **Cum se face crearea lui?**
        - Un certificat nu este secret (tot certificatul este în clar). Pentru ca acesta sa fie valid el trebuie sa aiba hash-ul creat și hash-ul respectiv sa fie semnat de o autoritate de certificare - CA (Certificate Authority). CA criptează cu cheia sa privată rezumatul certificatului
    - **Pași pentru verificarea certificatului** de către Alice:
        - A aplica cheia publica a CA asupra semnaturii
        - A calculează rezumatul SHA-1 al certificatului (fără semnătura)
        - A compara cele doua rezultate
    - **Revocarea certificatelor** -> Un certificat trebuie revocat când:
        - cheia primara este compromisa
        - cheia primara este pierduta
    - Cand trebuie revocat?
        - cand cheia primara e compromisa/pierduta
        - o persoana pleaca din companie
    - **Revocarea trebuie anunțată tuturor utilizatorilor** => dificil! Se folosește CRL (Certificate Revocation List). Metoda:
        - se verifica listele de revocare înainte de utilizarea certificatelor
        - CRL sunt publicate de CA care a emis certificatele
        - Listele pot fi consultate sau duplicate (cache)
        - difuzarea listelor de revocare prin HTTP, LDAP, etc.
    - Pentru desen CTRL + F “Verificarea revocarii Certificatelor”
- **Dacă am un mesaj pe care îl trimit de la un sender la un receiver, urmat de hash-ul mesajului semnat cu cheia privată a senderului - ce rezolv cu asta?**

    - confidentialitatea - **NU**
    - integritatea mesajelor - **DA** (ptc am pus hash mapul acolo)
    - autentificarea - **DA** (ptc hash-ul e semnat cu cheia privată a sursei, deci mesajul vine neapărat de la sursa)
- **PGP**
    - securitate pentu emailuri
    - oferă confidentialitate, autentificare, semnatura digitala, compresie
    - minutul 1h:29 curs 11.05.2020

12 SECURITATE

- https://ocw.cs.pub.ro/courses/pc/laboratoare/12
- intrebari
    - ipsec - imp
    - sa database - ce este?
    - sa: esp - transport & tunel, avem ath header &esp (criptare + autentificare)
    - hash map
    - isakmp -> diffie hellman
    - ike
    - firwall - ce este?
    - ni se da un schimb de mesaje si sa il explicam (ath cu cheie secreta partajata/ atac prin refl)
    - ce este atacul prin reflexie - explicatie schema eventual
    - ce se intmpla daca trudy vrea sa faca un atac prin refl + restul atacurilor
    - ssl - cand se face verif certificatului; daca se face verif certificatului clientului sau a serverului + certficate forta
    - https - http bagat peste ssl
    - secure naming - inregistrari in plus cu cheia si semnatura (secure dns)
        - dc nu pot sa fac un server fake de dns in care sa falsific acea semnatura? - securitatea unei semnaturi digitale !
    - ce se intampla daca nu stiu cine isi pierde credibilitatea? (a fost compromisa semnatura unui ca = a fost compromisa semnatura lui)
      in mom in care un ca isi pierde credibilitatea (a fost compromisa cheia lui)
        - atacatorul poate sa genereze certificate false ptc care nu e nicioo metoda cu care sa ne dam seama daca sunt reale sau nu
        - niciun certificat care a fost semnat de catre ca nu mai poate fi considerat sigur => toate trebuie invalidate; regenererare cheie; recrte certificatele
- **IPsec:**
    - Implementat la nivel IP
    - secure network protocol **suite**
    - peste IPV4, IPV6 iau pauza 10 min ca lesin mor cad okk
    - connection oriented, deși e la nivel IP. Asemanator lui TCP cu modul în care își stabilesc conexiunea
    - **security association** = legatura securizata **unidirectionala** intre sender -> receiver

        - se face dif intre sender si receiver
        - 2*SA pentru a securiza orice canal bidirecțional (și receiver -> sender)
    - asigura autentificarea mesajelor sau autentificare + criptare
    - SA nu este legată de un singur algoritm de criptare sau de o singura cheie.
    - Se pot specifica următorii **parametrii de securitate:**
        - algoritmul și modul de criptare (ex. DES în mod block-chaining)
        - cheia de criptare
        - parametrii de criptare (ex. Initialization Vector)
        - protocolul de autentificare și cheia
        - durata de viata a unei asociații (permite sesiuni lungi cu schimbarea cheii dacă este necesar)
        - adresa capatului opus al asociației
        - nivelul de senzitivitate al datelor protejate
    - nu este dependent de un algoritm de criptare. Dacă găsim unul mai bun, îl putem înlocui => se pastreaza un parametru în care reținem algoritmul de criptare
    - durata de viata (timp limitat) depinde de:
        - cheia de sesiune
        - cheia de criptare care se regenereaza periodic
    - dacă folosești o cheie de criptare de mai multe ori
        - îi oferi unui atacator mai multe date de analizat
        - o cheie se sparge în x ore => trebuie sa te asiguri ca spargerea >= generarea unei noi chei => cheia veche nu mai este validă și degeaba o sparge Trudy
    - Permite alegerea granularitatii
        - conexiune TCP
        - toate legăturile între două calculatoare (tunel)
        - toate legăturile între două rutere
- **SA Database**
    - pentru orice sistem care permite IPsec
    - un sistem păstrează o baza de date cu asociațiile de securitate. Pentru fiecare SA pastreaza parametrii de securitate (vezi mai sus) și
        - contor numere de secvența: pentru antete de securitate
        - Indicator overflow pentru contor numere de secventa: ce-i de făcut la depasire limita contor
        - fereastra anti-replay: determina dacă un pachet este o copie
        - Path MTU: path Maximum Transmission Unit (pentru evitare fragmentare)
    - overflow: se trimit foarte multe pachete. Util în cazul în care un pachet ajunge prea târziu și s-a dat reset la contor.
    - se pastreaza integral ultimele x pachete.
        - un atacator încearca sa afle info retrimitand ultimele pachete
        - se verifica byte cu byte
        - criptare => absolut imp sa trimitem 2 pachete identice byte cu byte
        - ne putem da seama f ușor dacă Alice a trimis mesajul sau Trudy

- path mtu: pentru a evita fragmentarea. Deoarece headerele sunt criptate => nu le vrem fragmentate
- Un SA este identificat în mod unic prin:
  - Security Parameters Index (SPI): identificare SA la receptor
  - IP Destination Address
  - Security Protocol Identifier
- **două protocoale de securitate**
  - AH (Authentication Header) - inserat în datagrama IP
  - ESP (Encapsulating Security Payload)

- doua moduri de lcuru
  - transport
  - tunel

- **AH**
  - layout: ip header + ah header + tcp header
  - **nu asigura confidentialitatea** pt ca datele nu sunt criptate, asigura numai autentificarea
  - Functionalitati:
    - verifica originea datelor
    - un hash al datelor => integritatea datelor
  - sequence number (camp în header) => evitarea atacului prin replica
  - **hmac** = **hash** pentru întregul pachet
    - conține o cheie simetrică
    - măsură suplimentară pentru integritate
    - calculeaza rezumat peste întreaga datagrama (campurile variabile neincluse)
    - masura pentru autentificare: cheie simetrică => trebuie sa fie stiuta doar de cele 2 capete ale comunicatiei (ale SA-ului) => cheia este folosită în hash
- **ESP**
  - protocol combinat criptare/autentificare **(asigura si confidentialitatea**)
  - criptarea protejează incarcatura
  - autentificarea protejează antet ESP + criptograma
  - CTRL + F “ESP în modurile transport și tunel” în curs pentru a vedea desenele
- **ESP - mod transport**
  - layout: ip + esp + tcp + payload + hmac
  - antentul ESP e plasat intren antentele IP si TCP
  - campul “protocol” din antentul IP e modificat si arata ca urmeaza un antet IPsec
  - cheia pt HMAC este folosită (cu un algoritm simetric) și criptează TCP + payload
  - hash-ul se calculează pentru ESP + TCP + payload
  - IP rămâne intact pentru a putea realiza forwarding-ul

    - SA-ul se face end-to-end
- **ESP - mod tunel**
    - new IP + esp + old ip + tcp + payload + hmac
    - la pachetul IP se adauga antetul IPsec și un nou antet IP
    - tunelul se poate termina înainte de destinație (de ex. la un firewall)
    - new IP: poate avea un SA între routere
        - rețea locală A
        - rețea locală B
        - facem tunel între router-ul prin care ieșim din A și cel prin care intram in B
        - adresa IP este cea a router-ului unde se termina tunelul (B)
    - IPsec + ESP tunel => comunicatia prin internet este securizata. În interiorul rețelei nu avem nevoie de securizare => O folosim doar pentru comunicarea între rețele
- **HMAC**
    - CTRL + F în curs (sunt 5 bulinute)
- **ISAKMP**
    - genereaza o cheie distinctă pt fiecare asociație
    - implementat cu IKE
    - folosește Diffie-Hellman
- **IKE**
    - CTRL “IKEv1” pentru desene
    - Phase 1:
        - Clientul trimite o cerere de tipul IKE Security Association (SA)
        - Se trimit identitățile (ID-uri, certificate, etc.)
        - Include schimbul de chei Diffie-Hellman
    - Phase 2:
        - SA este stabilit și securizat
        - Parametrii SA sunt renegociati după un interval fix de timp
- **IKEv2**
    - Pasul de autentificare este criptat
    - Poate include protocoale de autentificare diferite
    - Autentificarea este un pas separat (1.5)
- **Firewall**
    - are posibilitatea de a bloca sau de a modifica traficul la nivelul rețea (filtrare de pachete => poate da drop unora)
    - nu sunt protocoale de comunicatie
    - sunt așezate la intrarea în rețea
    - mod de funcționare:
        - open pentru Internet
        - cu firewall: default closed. Toate pachetele sunt filtrare. Orice nu se potrivește este blocat
    - Criterii de filtrare:
        - după adresa IP sursa + dest
        - în funcție de protocol: tcp/udp
        - în funcție de port => ne dam seama de protocolul de nivel aplicatie

    - nu face analiza detaliată a pachetelor
    - sunt **stateless**:
        - pachetele sunt tratate individual
        - filtrare doar în baza parametrilor din pachetul curent
        - nu putem filtra pachetele care vin ca răspuns la un request făcut de user
- **VPN:**
    - metode de protectie peste rețele publice
    - ascundem adresele IP sursa + dest
    - este transparent pentru aplicații (la nivel rețea)
- **Protocoale de autentificare:**
    - Determina dacă o entitate (utilizator, proces) este cu adevărat cine / ce pretinde ca este
        - diferită de autorizare, se bazează pe un schimb de mesaje prin Internet (prezentate, de regula, ca schimb între Alice și Bob).
        - mesajele pot fi interceptate și folosite de alte entități (de regula Trudy)
    - Protocolul generează și o cheie de sesiune
    - Folosesc criptografia cu
        - Chei secrete partajate
        - Chei publice
- **Autentificare cu cheie secreta partajata:**
    - protocol de tip challenge - response. Bob e singurul care poate răspunde la mesajul lui Alice => își demonstreaza identitatea
    - CTRL + F pentru schema:
        - A - id-ul lui Alice
        - Rb - challenge-ul lui Bob
        - Ra - challenge-ul lui Alice
        - Kab(Ra) si Kab(Rb) - challenge-ul codificat cu cheia comuna (secretă)
    - Se realizeaza astfel autentificare mutuala
    - RA, RB - numere aleatoare foarte mari, folosite contra atac prin replica
- **Atacul prin reflexie**
    - CTRL + F pentru desen
    - Reduce vulnerabilitatile: nu folosește Rb
    - Trudy inițiază mai multe sesiuni (Bob nu-si da seama de diferenta dintre sesiuni):
        - Trudy trimite A și Rt (un challenge capcana).
        - Bob răspunde cu Rb si cu Kab(Rt) => Trudy obtine Rb
        - Trudy trimite A, Rb
        - Bob raspunde cu Rb’ si Kab(Rb) => Trudy obtine mesajul Rb codificat cu cheia partajata
        - Trudy poate trimite acum Kab(Rb) => se va identifica ca Alice
    - Analog: Trudy poate stabili 2 sesiuni autentificate cu Alice (CTRL + F “Atacul prin reflexie pe protocolul initial”)
- **Man in the middle**:
    - Se bazeaza pe Diffie Hellman. Vulnerabilitate pt Diffie Hellman

    - Trudy este fix in mijlocul canalului de comunicatie. Intercepteaza toate datele Alice <-> Bob
    - Alice are impresia ca vorbeste cu Bob, dar schimbul de mesaje se face doar cu Trudy. Analog Bob.
    - Alice porneste partea de key exchange (crezand ca vorbeste cu Trudy). CTRL + F “Atacul man-in-the-middle” pentru schema
    - Trudy realizeaza 2 Diffie Hellman: unul cu Alice, unul cu Bob => cheie secreta stabilita cu Alice g^(xz) mod n + cheie secreta stabilita cu Bob g^(yz) mod n
    - Rezolvare: semnaturi digitale (vezi mai sus)
- **Key distribution center**:
    - Autentificare fara sa folosesc chei private. Am la mijlocul comunicatiei un KDC (o entitate sigura) si care poate intermedia canalul de comunicatie
    - Alice si Bob vor sa stabileasca un Ks - cheie de sesiune.CTRL + F “Autentificarea folosind Key Distribution Center” pentru desen
    - KDC detine toate cheile de criptare. Pasi:
        - decripteaza Ka(B, Ks) -> obtine B si Ks.
        - inlocuieste B cu A. Cripteaza din nou totul cu Kb
        - Trimite Kb(A, Ks).
    - Atat Bob cat si Alice afla asa care e Ks.
    - Probleme:
        - KDC trebuie sa aiba toate cheile. Cum distribui cheile?
        - Trebuie ca toata lumea sa aiba incredere in el => absolut orice atacator poate compromite orice din tot sistemul
        - Mesajele nu sunt semnate
        - Mesaje vulnerabile la replay attack. Putem retrimite acelasi mesaj si totul este ok
- **Needham-Schroeder:**
    - protocol de autentificare care se bazeaza pe chei simetrice. Incearca sa verifice identitatea mai multor entitati printr-un KDC. (autentificarea lui Alice si a lui Bob)
    - CTRL + F “Autentificarea cu protocolul Needham-Schroeder” pentru desen
    - Alice genereaza Ra1 (valoare random folosita o singura data).
    - KDC cunoaste toate cheile si are chei separate, share-uite cu Alice si Bob. Ka,kdc cunoscuta doar de catre Alice si KDC. Kb,kdc cunoscuta doar de Bob si KDC
    - Kab - cheie de sesiune cunosta de Alice si Bob.
    - Mesajul 2 din desen: Ra1 este introdus din nou pentru a te proteja against replay attacks. Alice nu poate decripta Kb,kdc(A, Kab) si nici nu are nevoie.
    - Mesajul 3: Bob poate decripta cu Kb,kdc => afla A (autentificare pentru Alice) si Kab => poate decripta si Kab(R2) => obtine R2. (Este un fel de challenge => Trebuie sa raspunda la challenge).
    - Mesajele 4 + 5: Bob + Alice raspund la challenge => se face autentificare => acum pot folosi cheia partajata Kab.
    - Vulnerabilitate: un fel de replay attack. CTRL + F “Slabiciune Needham-Schroeder” pentru desen
        - Chuck afla cheia Kab

        - Trudy (Chuck) poate intercepta mesajul 3 si i-l trimite lui Bob
        - Bob crede ca primeste mesajul 3 de la Alice (identic cu mesajul pe care l-ar trimite Alice)
- **Autentificarea folosind Protocolul Otway-Rees:**
    - CTRL + F pentru desen
    - Bob cand primeste mesajul 1 nu poate decripta a 2a parte din mesaj (nu detine Ka).
    - RA, RB – numere aleatoare folosite de KDC in mesajele 3 si 4 pentru a face legatura cu mesajele 1 si 2. KDC poate decripta mesajul 2. Trimite mesajele 3 si 4
    - Problema: nu se garanteaza ca mesajele 3 si 4 ajung in acelasi timp la Alice si Bob. Alice poate primi mai intai mesajul 4 => ii trimite mesaje criptate cu Ks => Bob nu le intelege pt ca inca nu a ajuns mesajul 3.
- **SSL/TLS:**
    - SSL = Secure sockets layer
    - TLS = Transport layer security
    - Este facut ca un protocol care sa ofere end-to-end security. Vine peste o conexiune tcp normala si o securizeaza. Putem face ori este nevoie pt autentificare, schimb de chei, etc.
    - Functionalitate:
        - Confidentialitate: toate datele intre client ⇔ server sunt criptate
        - Integritatea datelor: prin hash-uri
        - Autentificare
    - Folosit pentru verificarea de certificate peste HTTP
    - Elemente SSL: (CTRL + F din curs “Componente SSL”)
        - SSL Record Protocol: confidentialitate cu hash
        - SSL Handshake Protocol
        - SSL Change Cipher Spec Protocol
        - SSL Alert Protocol
    - layout in sativa TCP/IP: ip + tcp + ssl + http
    - Este nevoie ca aplicatiile TLS sa fie aware de asta. Sa stie ca folosesc conexiune criptata => nu trebuie sa aplice efectiv operatii de criptate/decriptare => Modificari la cod pentru a imbina cu functii specifice SSL.
    - De obicei se autentifica serverul, dar clientul nu intotdeauna. Doar f rar se autentifica clientul. (Ca parte din SSL Handskahe)
        - Site-ul trimite un certificat valid
        - Clientul trimite inapoi un certificat (extrem de rar. Cand trimit un request HTTP, nu trebuie sa imi dovedesc identitate trimitand un certificat valid)
    - Autentificarea clientului se face cu username si parola (pentru a dem identitatea), insa asta se face deja peste HTTPS => Conexiune sigura
    - Sesiune SSL:
        - se face handshake si se stabilesc parametrii => se considera ca s a inceput o sesiune intre client si server

        - by default, tcp nu permite conceptul de sesiune. Sesiunea e adaugata de SSL
        - optimizare: se face conexiunea, se stabilesc parametri, schimb de chei, etc. Dupa: comunicatie pe tcp. Prin conceptul de sesiune pot sa creez mai multe conexiuni tcp in cadrul aceleiasi sesiuni ssl si comunic direct. Pastrez parametrii de la primul handshake
        - Exemplu: 10 conexiuni intre acelasi client si acelasi server, nu e nevoie de 10 handshake-uri. Algoritmii se pastreaza, etc.
        - Pot fi mai multe sesiuni in paralel intre acelasi client si server (**DAR** NU E RECOMANDAT IN PRACTICA)
    - Etape pachet SSL/TLS:
        - Fragmentarea pachetului => SSL plain text
        - Compresie (zip pe segmentul de date) => SSL Compressed
        - Pachet final: se adauga un MAC (+/- padding) (partea de autentificare) => SSL Ciphertext. (Padding-ul e folosit pt ca unii algoritmi lucreaza doar cu blocuri de dimensiune standard). Ultimul byte din padding reprezinta lungimea padding-ului
    - Cele 3 campuri din header-ul de SSL sunt trimise in clar.
    - SSL Alert protocol:
        - specificatii pentru alerte: warning sau fatal
        - warning: afisate mesaj pentru utilizator. (Ex: website cu certificat expirat)
        - fatal error: se invalideaza id-ul de sesiune => nu se poate folosi conexiunea actuala => nu mai pot sa creez noi conexiuni pe baza aceleiasi conexiuni => handshake de la 0 + negociere parametrii de securitate
    - **SSL Handshake:**
        - clientul trimite un hello message (initiaza conexiunea): trimite cipher suite si compression methods
        - server-ul trimite si el un hello (contine certificatul). Il cere si clientului certificatul (daca este cazul).
        - clientul verifica certificatul serverului (autentifica serverul). (Optionala. Daca fail-uieste, da warning). Creeaza cheia de criptare = master secret. Cripteaza mesajul cu cheia publica a serverului (luata din certificat).
        - Doar serverul va putea sa decripteze si doar serverul va stii cheia de criptare. Creeaza un master secret key pentru sesiune si ii spune clientului.
        - Clientul decodifica, afla master key-ul
        - Handshake over
    - **Vulnerabilitati SSL Handshake:**
        - Security downgrading: clientul trimite o lista de algoritmi de criptare. Prin hello-ul serverului, este trimis inapoi algoritmul ales de criptare. Daca un client trimite numai algoritmi de criptare slabi => criptare slaba a mesajelor

        - Rezolvare: serverele HTTPS contin numai algoritmi buni si sunt refuzati toti algoritmii slabi
    - **TLS vs SSL:**
        - In loc de algoritm de MAC, TLS se foloseste HMAC. Acopera si campul de versiune din header, pentru verificarea integritatii
        - Introduse coduri de alerta
        - A fost scos un algoritm din lista de key exchange.
    - Probleme TLS:
        - nu ai garantia ca datele sunt pastrate in siguranta (in baze de date sigure)
- **HTTPS:**
    - HTTP + SSL => ne trebuie certificate
    - Se foloseste ssl + handshake si dupa folosim http prin acel canal securizat
    - Singura diferenta e ca portul se schimba in 443.
- **Secure naming:**
    - securizare baze de date DNS
    - in loc de o singura inregistrare, se adauga 3:
        - DNS normal
        - cheia publica a serverului DNS
        - semnatura folosind cheia privata
    - Pentru un server dns fake, nu poti sa falsifici semnatura serverului dns real.