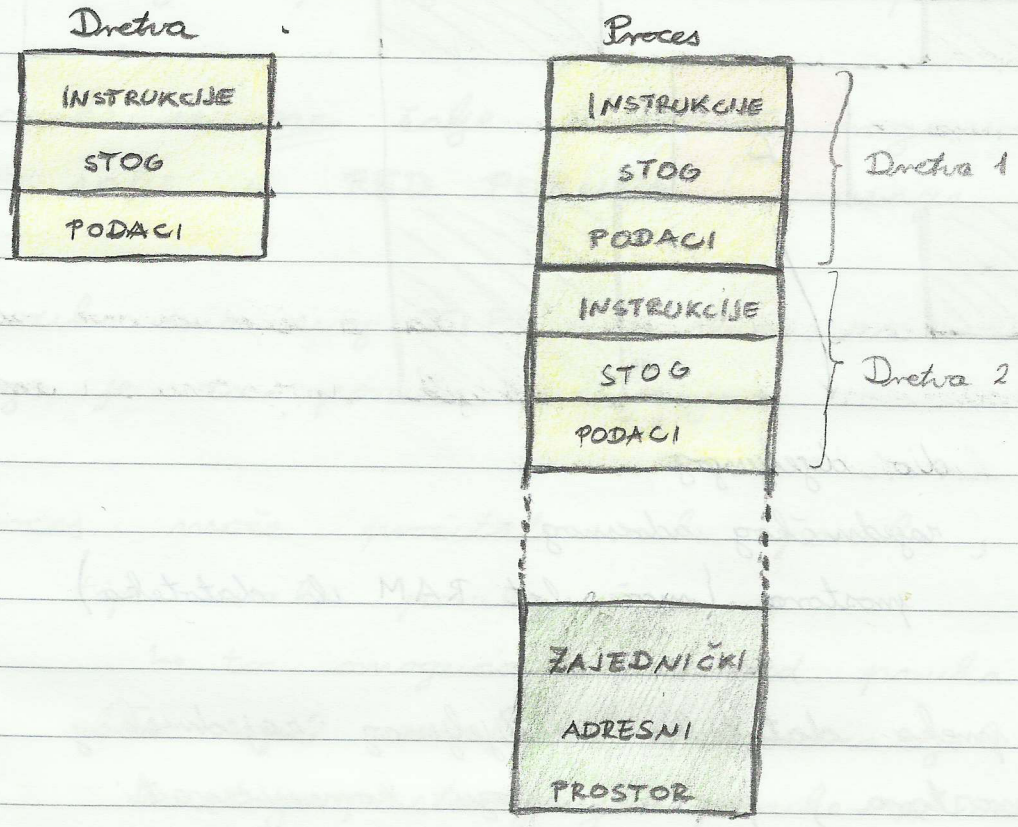
NAPREDNI OPERACIJSKI SUSTAVI

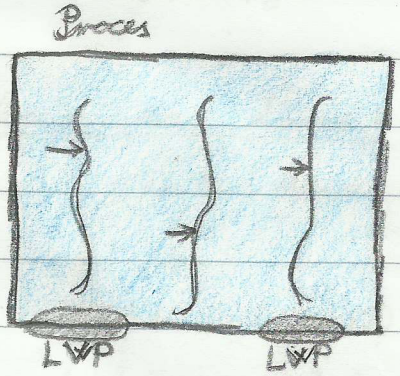
⇒ OPERACIJSKI SUSTAV ⇒ kompleks programa koj djeluju kao posrednici između sklopovlja i primjenskih programa te korisnika

⇒ sjeti se da je operacijski sustav tu da ti što manje smeta te da ti omogućava korištenje drugih programa (onih koje ti želiš koristit)

KOMUNIKACIJA IZMEĐU PROCESA UNUTAR ISTOG RAČUNALNOG SUSTAVA



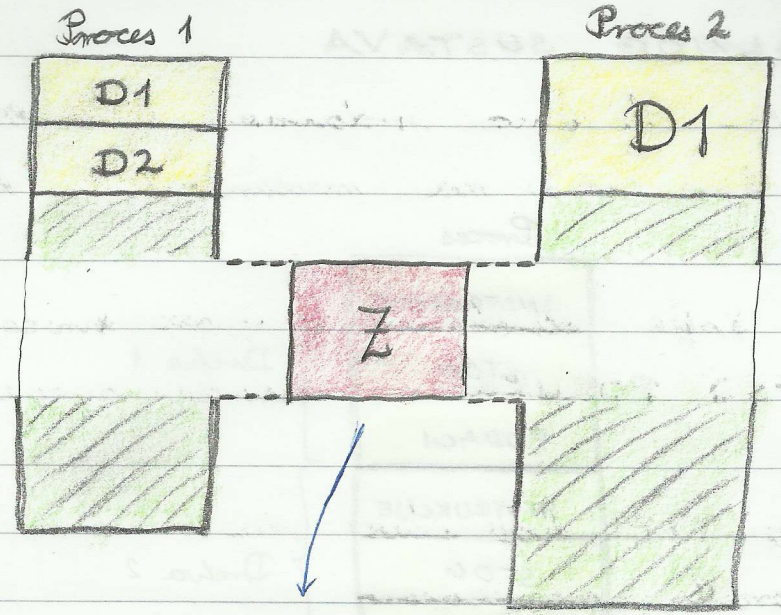
↳ ovo je prikaz memorije dretve i procesa



⇒ proces se može sastojat od više dretvi

→ „light-weight processor"
(virtualni procesor)

⇒ komunikacija između procesa (naredba SHMGET):



dio dijeljenog
razdvojenog adresnog
prostora (može biti RAM ili datoteka)

↳ osim preko datoteka i dijeljenog razdvojenog adresnog prostora, procesi mogu komunicirati pomoću varijabli okruženja

⇒ ovo što smo do sada rekli je gradivo Operacijskih sustava

KOMUNIKACIJA PORUKAMA

⇒ PORUKA ⇒ „mala" količina podataka (nekoliko stotina bajtova) koja služi za međuprocesnu komunikaciju

⇒ poruka se **ne** šalje direktno drugom procesu, već se šalje u **RED PORUKA** („message queue")

⇒ poruka ima svoj **TIP** koji služi za raspoznavanje za koga je ta poruka namijenjena

⇒ proces može pročitati svaku poruku, bez obzira na tip svake poruke

↳ to omogućava da red poruka koristimo kao opći **semafor** (sinkronizacijski mehanizam)

↳ stoga, stavljanje poruke u red je ekvivalentno naredbi POSTAVI_OSEM, a uzimanje poruke iz reda je ekvivalentno ČEKAJ_OSEM

↳ drugim riječima, ako je red poruka prazan, semafor je neprolazan

⇒ <u>funkcija za Linux:</u>

int **msgget** (key t_key, int flags)

↓

-1 ili ID reda poruka

(služi za dohvaćanje reda poruka)

↳ tipično korištenje funkcije msgget:

```
if ((msgqid = msgget(key, 0600)) == -1) {
    printf("Error!");
}
```

(u g o)

flags ⇒ prava pristupa (obično oktalno jer su tada znamenke „u g o")

↳ značenje tih slova:

u ⇒ „user" ; g ⇒ „group" ; o ⇒ „owner"

↳ 6 je 110 što je za **user**-a „rwx", pa smije čitat i pisat, al ne smije izvršavat („read", „write", „execute")

↳ zato se piše 0600

key ⇒ svaki red poruka treba imati svoj broj postavljen za „key"

↳ ako je više korisnika, a samo jedan proces, ovo je dobro:

```
key = getuid(); // il JMBAG
```

↳ no, kako postoji više korisnika i više procesa, preporuča se prenošenje ključa preko VARIJABLE OKOLINE (bit će objašnjeno što je to)

⇒ **dodatne funkcije za Linux:**

↳ slanje poruke: msgsnd

↳ primanje poruke: msgrcv

↳ kontroliranje / brisanje poruka: msgctl

↳ upoznaj se sa rastavicom IPC_NOWAIT
(trebala bi služiti za preskretanje poruka).

⇒ NAPOMENA: ako se ne izbriše red poruka funkcijom msgctl, unutar programa, taj red ostaje visjeti, pa da bi dobili popis redova poruka, u komandnu liniju upisujemo naredbu ipcs sa opcijom -q

↳ brisanje reda poruka:

ipcrm -q MSGQID

⇒ NAPOMENA: red poruka je neovisan o procesima koji ga koriste (isto kao i segment dijeljene memorije)

OKOLINA

⇒ OKOLINA („Environment"):

↳ to je niz znakova oblika "ime = vrijednost" koji se pridodjeljuju svakom programu prilikom pokretanja

⇒ VARIJABLA OKOLINE:

↳ varijabla koju procesi mogu međusobno dijeliti

↳ poznata varijabla okoline je PATH

↳ one poprimaju vrijednosti koje su niz znakova (treba ih parsirati ako želimo dobiti broj)

↳ postoje dva tipa:

- one koje se koriste samo u ljusci (njih ne dobiju programi prilikom pokretanja)
- one koje se predaju svim programima

⇒ KORISNIČKE LJUSKE (i neke naredbe):

	sh	csh
samo za ljusku	set ime=vrijednost	setenv ime vrijednost
popus (samo za ljusku)	set	set
one za procese	export	setenv

⇒ kako prenjet varijable okoline u program ?

↳ možemo koristit proširenu inačicu funkcije „main":

int main(int argc, char *argv[], char *envp[])

↳ no, kako ovo ne bi trebali stalno parsirat i slat svim funkcijama, „envp" se obično ne koristi direktno nego koristimo funkcije:

char *getenv(char *ime)

↳ dohvaća "ime = vrijednost"

int putenv(char *string)

↳ postavlja varijablu okoline

PROTOKOLNI SLOJ

⇒ ponovimo malo slojeve u mrežama kako bi dali uvod za komunikaciju u protokolnom sloju

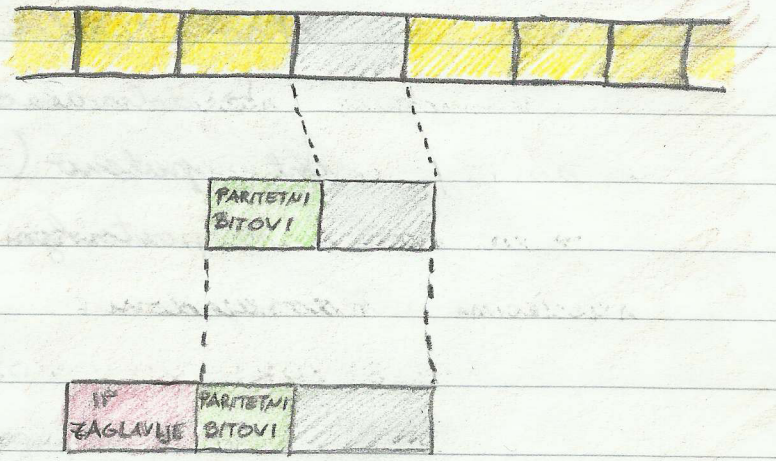
1. PRIMJENSKA RAZINA
("Application Layer")

2. PRIJENOSNI PROTOKOL
("Transmission-Control Protocol")

3. INTERNET PROTOKOL
("Internet Protocol")

4. FIZIČKI SLOJ
("Physical Layer")

Datoteka podijeljena na dijelove koju želimo poslat preko mreže





SLANJE IP PAKETA FIZIČKIM MEDIJEM

CJEVOVODI

⇒ sada još uzak razmatramo kako procesi mogu komunicirat u jednom računalnom sustavu

⇒ **kako povezati procese cjevovodom?**

↳ cjevovodi su ramišljeni za jednosmjernu komunikaciju, ali je moguće slat poruke u oba smjera uz oprezno i dobro - sinkronizirano komunikaciju

↳ za povezivanje, potrebno je izvodit operacije slijedećim redoslijedom:

1) STVORI CJEVOVOD



2) STVORI PROCES DIJETE



3) ZATVORI PISANJE ZA DIJETE



4) ZATVORI SLUŠANJE ZA RODITELJA



⇒ NAPOMENA: cjevovod nije nužno „1 na 1" (može više procesa čitat i pisat ir/i u cjevovod)

⇒ sličnost datoteka i cjevovoda je očita, ali ima razlika:

a) čitanje i pisanje kod cjevovoda ide po FIFO principu

b) cjevovod može biti pun i praran

c) čitanje kod cjevovoda je destruktivno

d) datoteka je predstavljena s jednim opisnikom, a cjevovod s dva opisnika

⇒ NEDOSTACI CJEVOVODA:

1) procesi moraju biti u nekom „srodstvu" da bi oni mogli komunicirati

2) već stvoreni procesi ne mogu komunicirati cjevovodom

3) cjevovodi koriste stranice za komunikaciju (engl. „page"), pa je ograničenje na veličinu u modernim računalima oko 4 kB

⇒ stvaranje cjevovoda:

↳ iz komandne linije: ls | wc ← ovo stvara cjevovod između ova dva programa

↳ iz programa:

int pipe (int fd[2])

0 ⇒ OK
1 ⇒ ERROR

fd[1] ⇒ WRITE
fd[0] ⇒ READ

} napravi #DEFINE za 1 i 0

⇒ raskidanje cjevovoda:

↳ iz programa:

int close (int fdn)

⇒ dupliciranje cjevovoda

↳ iz programa:

int dup (int fdn)

↓

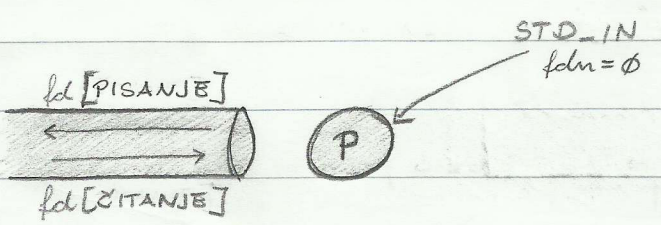
vraća novi broj opisnika datoteke nastao dupliciranjem danog opisnika

fdn	stream
0	STD_IN
1	STD_OUT
2	STD_ERR

} da ne pomiješaš ove nule i jedinice s onima u naredbi „pipe", na početku naprovi spomenute definicije:

```
#define PIPE_WRITE 1
#define PIPE_READ 0
```

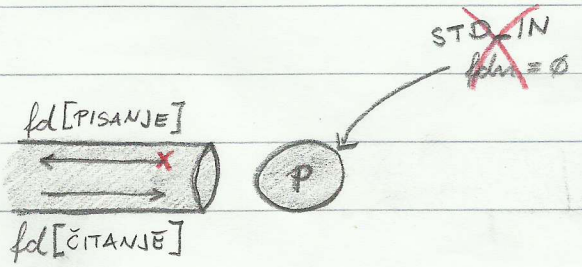
⇒ povezivanje cjevovoda na standardni ulaz:



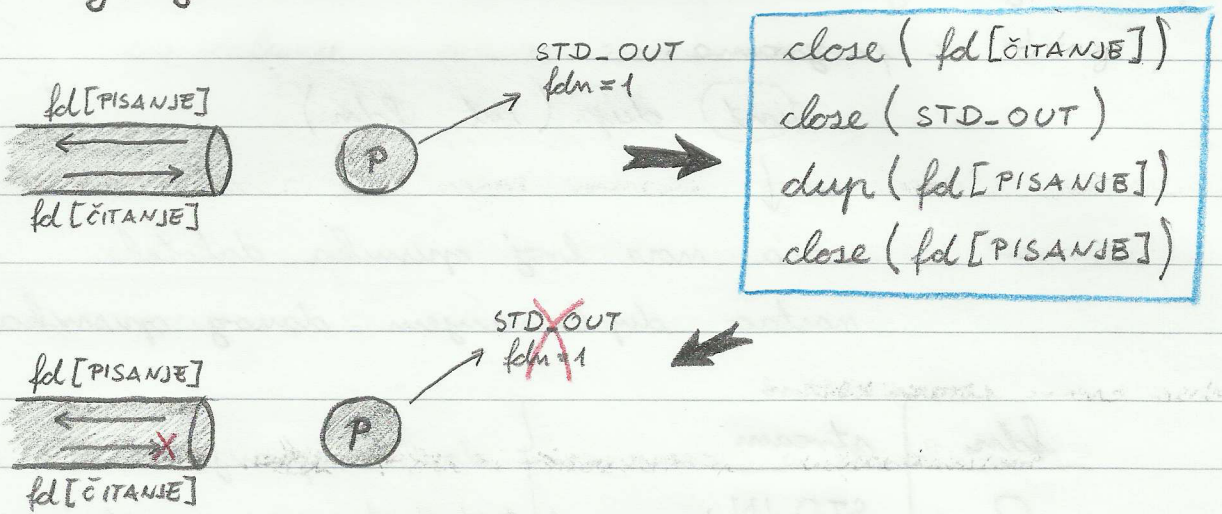

```
close(fd[PISANJE])
close(STD_IN)
dup(fd[ČITANJE])
close(fd[ČITANJE])
```

↓

zadnja naredba nije nužna jer samo zatvaramo standardni ulaz programa (ako želiš kasnije vratiti na staro, nemoj to zvati)



⇒ preusmjeravanje cjevovoda na standardni ulaz/izlaz:



⇒ sada smo uspjeli naučiti kako napraviti da procesi čitaju i pišu u cjevovode umjesto standardnog ulaza/izlaza

⇒ NAPOMENA: cjevovodi ne mogu konstiti za sinkronizaciju, jer čitač čeka ako je cjevovod praran, a pisač čeka ako je cjevovod pun

IMENOVANI CJEVOVODI

⇒ osmišljeni su zbog ograničenja i nedostataka običnih cjevovoda

⇒ imenovani cjevovod:

- kombinacija cjevovoda i datoteka
- vidimo ga kao datoteku (sa naredbom „ls")
  ↳ uvijek je veličine 0
- cjevovod <u>ne radi</u> dok se ne otvori i za pisanje i za čitanje (sluzi za sinkronizaciju)
  ↳ može se ignorirat sa zastavicom NO_DELAY
- podac se ne zadržavaju trajno u njemu

⇒ PREDNOSTI IMENOVANIH CJEVOVODA:

1) rad kao s običnom datotekom
2) procesi ne moraju biti u srodstvu
3) nema ograničenja na veličinu cjevovoda
4) može se naprovit in komandne linije, a ne nestaje završetkom procesa kao običan cjevovod
   ↳ funkcije: mknod

int mknod (char *path, int mode, int dev)

KOMUNIKACIJA IZMEĐU PROCESA U RASPODJELJENIM SUSTAVIMA

⇒ *načini komunikacijama:*

A) PORUKAMA

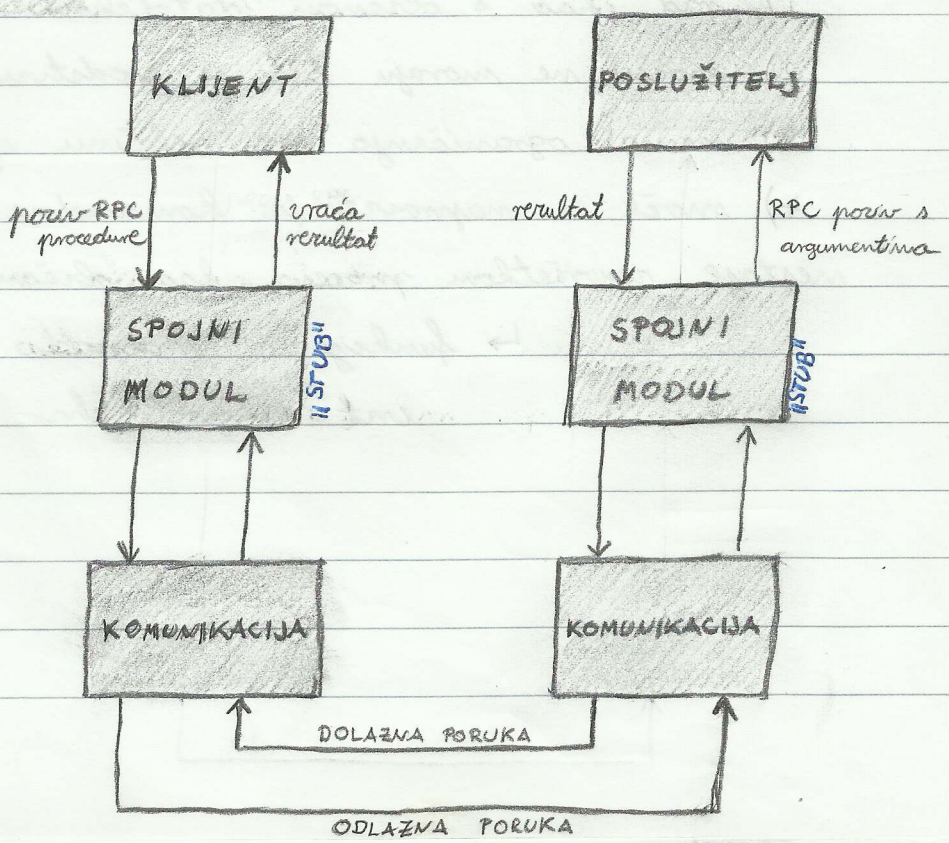
↳ nije red poruka jer je red poruka na jednom računalu (ovdje radimo raspodjeljene sustave)

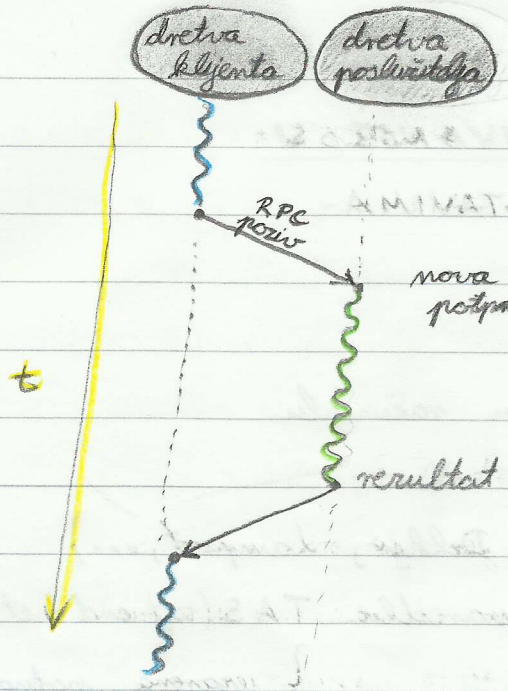
↳ poruke se šalju TCP/IP protokolima:

- trebamo priključnice (engl. SOCKET) i funkcije za rad s priključnicama

B) POZIV UDALJENIH PROCEDURA (RPC call)

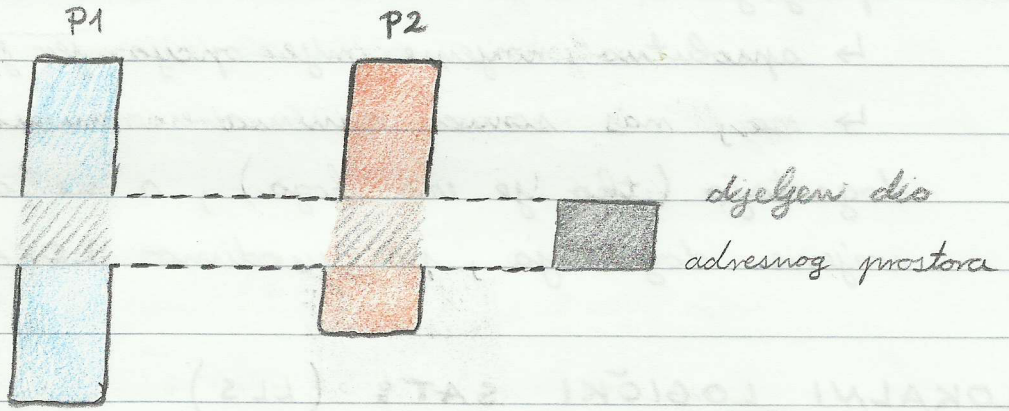
↳ funkcije se pozivaju na ovom računalu, već na udaljenom, pa se sigurno pozivaju metodom „call-by-value" (jer adresa na udaljenom računalu nije ista kao i na ovom)





klijentska dretva se
rablokira dok čeka
na odgovor (može
se sinkronizirati monitorima)

C) RASPODIJELJENI DIJELJENI SPREMNIČKI PROSTOR



↳ treba se čekat potvrda o uspješno pisanje
u dijeljeni dio adresnog prostora (opet rekakva
sinkronizacija monitorima)

MEĐUSOBNO ISKLJUČIVANJE U RASPODIJELJENIM SUSTAVIMA

⇒ **ponavljanje**: „mutex" na jednom računalu
↳ neki spomenuti načini:
① PROGRAMSKI (Dekker, Lamport, ...)
② SKLOPOVSKI (naredbe TAS (test-and-set), CAS, ...)
③ SEMAFORSKI
④ MONITORSKI (najbolje)
} sjegrena potpora

⇒ za raspodjeljene sustave trebamo VREMENSKO UREĐENJE:
↳ apsolutno vrijeme nije opcija jer je nepouzdano
↳ no, nas samo zanima vremensko uređenje događaja (tko je kad koga), a ne apsolutno vrijeme događaja, pa uvodimo LOKALNI LOGIČKI SAT

⇒ **LOKALNI LOGIČKI SAT**: (LLS)
↳ na svakom računalu
↳ radi kao brojilo (C_i ++)
↳ oznaka: C_i
↳ ako je $C_i(a) < C_i(b)$, onda je „a" prije „b"

⇒ **GLOBALNI LOGIČKI SAT: (GLS)**

↳ radi po pravilima:

1) P_i povećava svoj C_i nakon svakog događaja!

2) kada P_i šalje poruku „m", obavezno joj dodaje oznaku „T_m" svojeg LLS-a ($T_m = C_i$)

3) kada drugi proces P_j primi poruku „m", on mijenja vrijednost LLS-a C_j na:

$$C_j = \max \{ C_j, C_i \} + 1$$

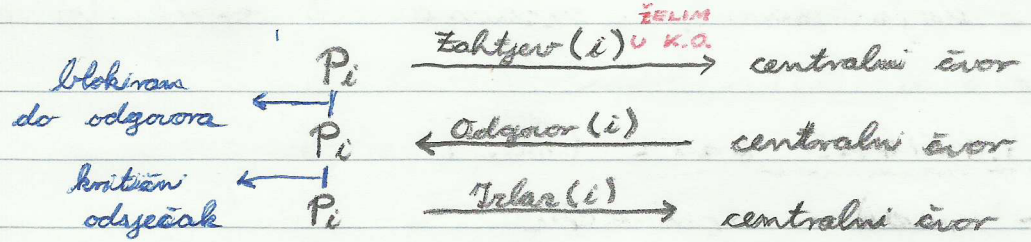
⇒ može se javit ist problem kao i kod Lamporta - dvije poruke imaju ist broj

↳ tada pusti onog s manjim „i"

⇒ **CENTRALIZIRANI PROTOKOL:**

↳ jedan čvor je glavni - radućen za „mutex"

↳ protokol ide ovako:



↳ problem: zagušenje centralnog čvora

⇒ **PROTOKOL S PUTUJUĆOM ZNAČKOM:**

↳ značka uređenim redoslijedom putuje kroz sve čvorove

↳ ako čvor ima značku, on može ući u kritičan odsječak

↳ ovo je analogno dodjeljivanju sabirnice u višeprocesorskom sustavu

↳ PROBLEM: ako proces ne želi ući u kritičan odsječak, on i dalje prima i šalje poruke jer mora poslati značku dalje

(dodatno: što ako neki čvor ispadne iz mreže?)

LAMPORTOV RASPODIJELJENI PROTOKOL

⇒ SASTOJI SE OD 5 PRAVILA:

1) kada proces P_i želi ući u kritični odsječak, on stvara poruku <u>rahtjev$(i, T(i))$</u>, pamti ju u svoj red poruka i šalje ju ostalim ($N-1$ procesa)

2) kada proces P_j primi poruku, on usklađuje svoj LLS u odnosu na GLS te još:
   - stavlja primljenu poruku u svoj red
   - šalje poruku <u>odgovor$(i, T(j))$</u>
     ↳ to je zapravo $T(j) = C_j$

3) proces P_i može ući u kritični odsječak kada je:
   a) njegov rahtjev prvi u njegovom redu
   b) primio odgovore svih ostalih procesa

4) proces P_i izlazi iz kritičnog odsječka tako da:
   a) briše <u>rahtjev$(i, T(i))$</u> iz svog reda
   b) šalje svima <u>izlazak$(i, T(i))$</u>

"svoj T(i)" je ona originalna vrijednost koja je bila kada se tražio ulazak

5) kada P_j primi <u>izlazak$(i, T(i))$</u>, on iz svojeg reda briše <u>rahtjev$(i, T(i))$</u>

⇒ loša strana: velik broj poruka ⇒ $N-1$ prilikom uloska
↳ ukupno $3 \cdot (N-1)$

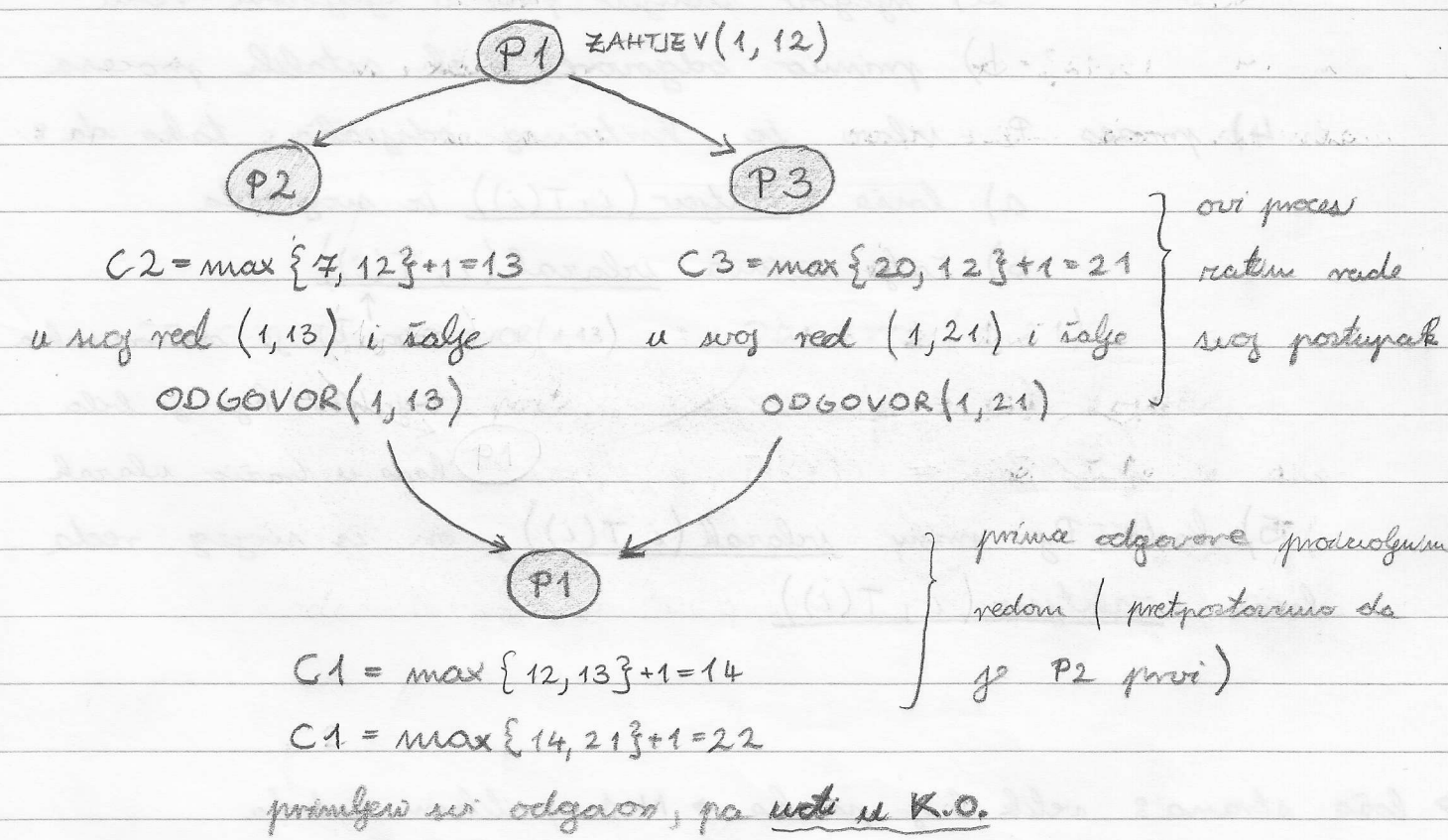
ZAD: U sustavu se nalaze 3 čvora i na svakom čvoru po jedan proces: P1, P2 i P3 koji u promatranom trenutku imaju vrijednosti LLS-ova 12, 7, 20. Ako u tom trenutku, P1 želi ući u K.O., opisati što će se sve dogoditi u sustavu.

Početno stanje	Red zahtjeva
P1: C1 = 12	—
P2: C2 = 7	—
P3: C3 = 20	—

P1 sada generira zahtjev

ZAHTJEV(1, 12)

i stavlja ga u svoj red pa šalje svim ostalim procesima



primivši svi odgovor, pa ući u K.O.

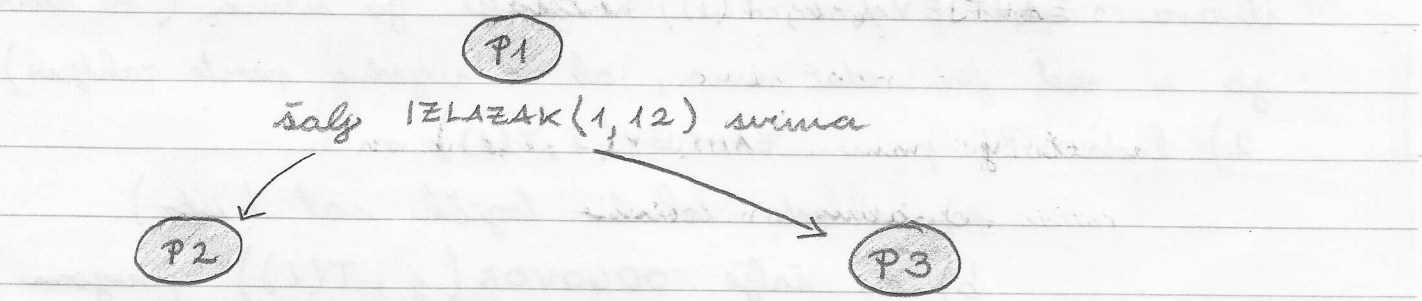
kada su primljeni svi odgovori, proces P1 ulazi u kritični odsječak

⇓

stanje sustava:

Stanje sustava	Red zahtjeva
P1: C1 = 12	ZAHTJEV(1, 12)
P2: C2 = 13	ZAHTJEV(1, 12)
P3: C3 = 21	ZAHTJEV(1, 12)

↳ sada, kada P1 izlazi iz K.O. događa se sljedeće:



C2 = max {13, 12} = 14

makni iz reda ZAHTJEV(1, 12)

C3 = max {21, 12} + 1 = 22

i makni iz reda ZAHTJEV(1,12)

⇓

stanje sustava:

Stanje sustava	Red zahtjeva
P1: C1 = 22	—
P2: C2 = 14	—
P3: C3 = 22	—

PROTOKOL RIKARDA I AGAWALLA

⇒ nastao u nad da se nađe protokol koj generira manj broj poruka od lamportovog protokola
↳ generira $2\cdot(N-1)$ poruka

⇒ SASTOJI SE OD 5 PRAVILA:

1) kada proces P_i želi ući u kritični odsječak, on stvara ZAHTJEV$(i, T(i))$ i šalje ga svima (ne stavlja ga u red jer reda nema, al se svejedno pamte zahtjevi)

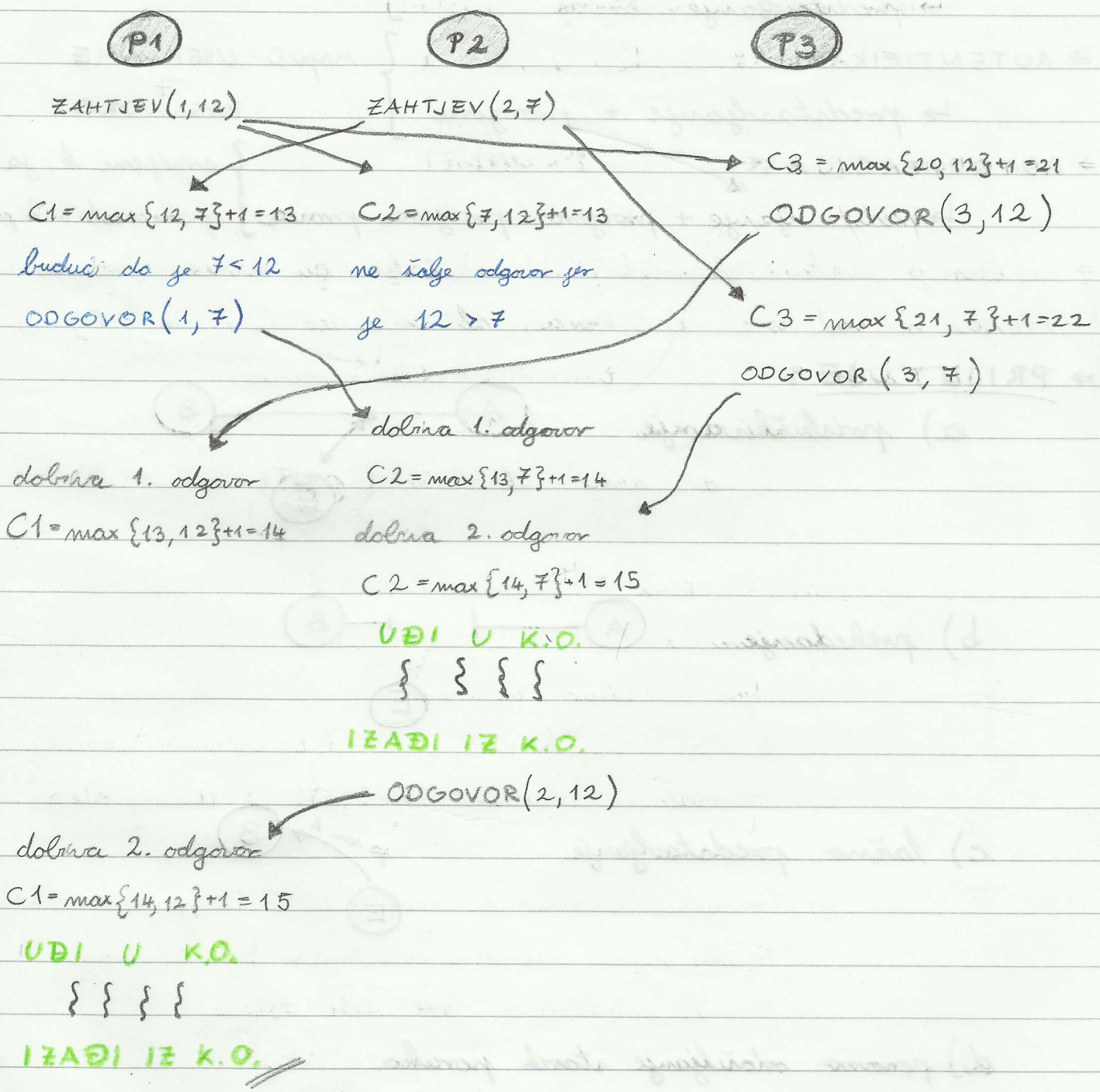
2) kada P_j primi ZAHTJEV$(i, T(i))$ on:
   a) uskladi lokalni logički sat (isto)
   b) on šalje ODGOVOR$(j, T(i))$ drugom procesu, al samo ako proces P_j ne želi ući u kritični odsječak il ako P_j želi u kritični odsječak i ako je njegov (od P_j-a) zahtjev došao kasnije
      ↳ dakle, P_j šalje odgovor ako postoj ZAHTJEV$(j, T(j))$ koji prethodi onaj $(i, T(i)) \Rightarrow$ ne šalje se ako je: $T(j) < T(i)$

3) kada P_i primi sve ODGOVORe u kojem je on, ulazi u K.O.

4) kada P_j izlazi iz K.O., on šalje ODGOVOR$(j, T(i))$ svim onim procesima čij zahtjev kod njega čekaju na odgovor

Zad: Isto zadatak kao prethodni, samo uz novi protokol i to tako da P1 i P2 istovremeno šalju svoj u kritični odsječak!

Početno stanje: P1: C1 = 12 P2: C2 = 7 P3: C3 = 20



⇒ ako su T(j) i T(i) jednaki, gleda se „j" i „i" i tko je od njih manji!

SIGURNOST RAČUNALNIH SUSTAVA

⇒ IDENTIFIKACIJA:
↳ predstavljanje } samo napiši USERNAME

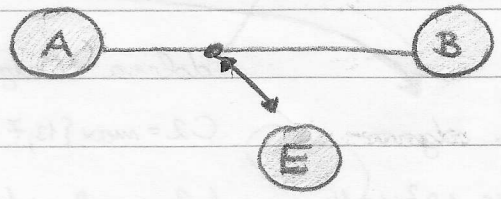
⇒ AUTENTIFIKACIJA:
↳ predstavljanje + provjera } napiši USERNAME + PASSWORD

⇒ AUTORIZACIJA:
↳ predstavljanje + provjera + provjera prava } smijem li ja pokrenuti neki program

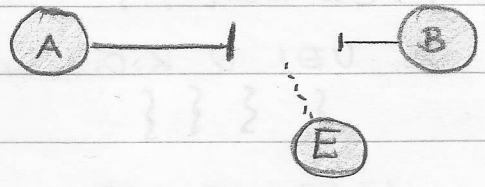
⇒

⇒ **PRIJETNJE:**

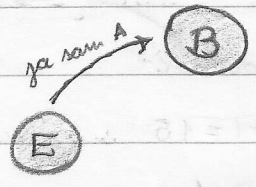
a) prisluškivanje



b) prekidanje

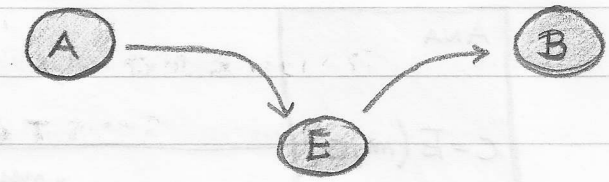


c) lažno predstavljanje



d) ponovno odašiljanje starih poruka

e) modifikacija paketa



f) poricanje

↳ „ja to nisam napravio"

(npr. nisam potpisao ugovor na Internetu)

⇒ SIGURNOSNI ZAHTJEVI: (OSNOVNI)

1) autentičnost

↳ sprječava prijetnje c) i d)

2) integritet

↳ sprječava prijetnju e)

3) tajnost

↳ sprječava prijetnju a)

4) neporecivost

↳ sprječava prijetnju f)

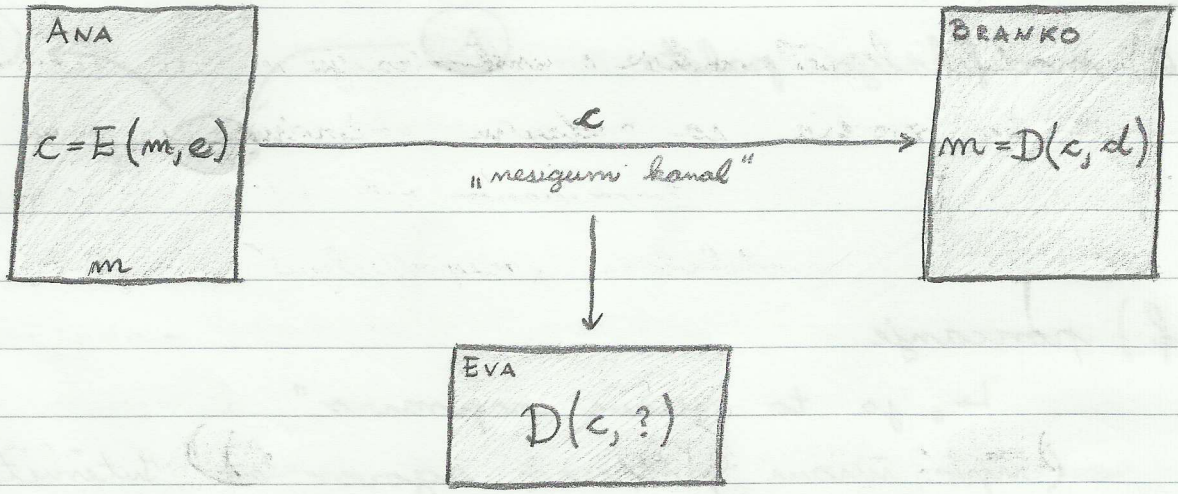
⇒ dodatni sigurnosni zahtjevi:

5) kontrola pristupa

↳ ispravno podešena, sprječava prijetnju a)

6) raspoloživost

↳ sprječava prijetnju b)



nema ključ
za dekripciju
↳ sigurnosni sustav mora biti
tako podešen da napadač u razumnom
vremenu ne može doći do ključa, a
time i do poruke

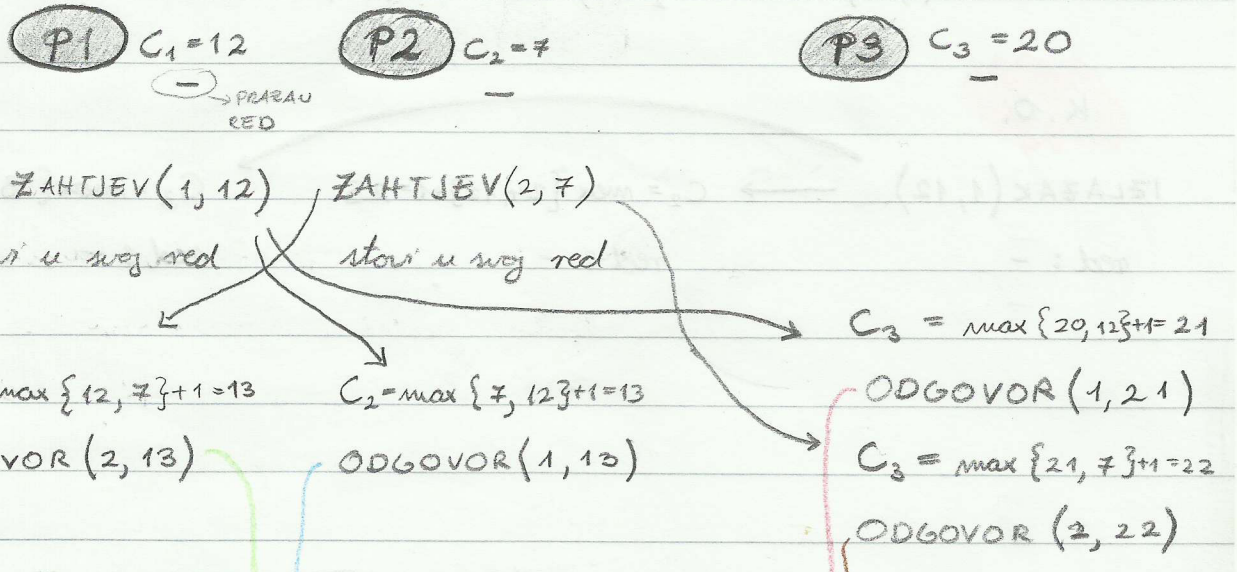
⇒ gornja slika služi samo da naučiš oznaku

⇒ TIPOVI KRIPTOGRAFSKIH ALGORITAMA:

a) simetrični
↳ isti ključ za enkripciju i dekripciju
$K = e = d$ (tajni, simetrični, sjedinički)
↳ neki algoritmi: DES, 3DES, IDEA, AES

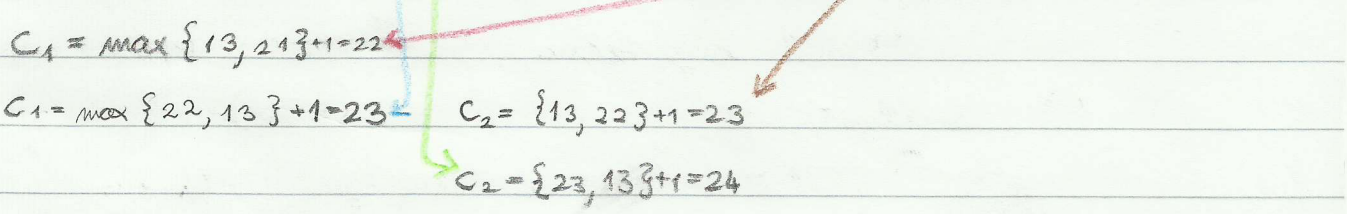
b) asimetrični
↳ različiti ključ za enkripciju i dekripciju
↳ imaju par ključeva: P (public), S (private)
↳ neki algoritmi: RSA, DH, ECES

ZAD: (Verano uz logičke satove i sinkronizaciju na više računala)
Imamo tri procesa sa radnim satovima i pravcem redosljeda. P_1 i P_2 istovremeno žele ući u K.O., a sinkronizira se protokolom „raspodjeljeni Lamport" 8



↳ sva tri procesa sada imaju isto stanje redova:
- sortirano po vremenskim oznakama:

ZAHTJEV(2,7) je prvi
ZAHTJEV(1,12) je drugi



↳ po primitku svih odgovora, stanje redova je:

ZAHTJEV(2,7)
ZAHTJEV(1,2)

↳ satovi: $C_1=23$ $C_2=24$ $C_3=22$

OKRENI

↳ prvi u redu je ZAHTJEV(2,7), pa P_2 ulazi u K.O.

↳ nakon izlaska, on šalje izlazak ostalima

K.O.

$C_1 = \max\{23,7\}+1 = 24$ ← IZLAZAK(2,7) → $C_3 = \max\{22,7\} = 23$

red: ZAHTJEV(1,12) | $C_2 = 24$, red: ZAHTJEV(1,12) | red: ZAHTJEV(1,12)

K.O.

IZLAZAK(1,12) → $C_2 = \max\{24,12\}+1 = 25$ → $C_3 = \max\{23,12\}+1 = 24$

red: - | red: - | red: -

SIMETRIČNI KRIPTOGRAFSKI ALGORITMI

⇒ **Kerkhoffov princip:**

↳ ideja kriptografije u prošlosti je bila da sam algoritam kriptiranja bude tajan (vezi se „Enigma machine"-a)

↳ danas to već nije tako

↳ ovaj princip:

- „kriptosustav mora biti siguran i onda kada je sve o kriptosustavu javno poznato osim javnog ključa"

⇒ matematički dokazan potpuno siguran kriptosustav:

$$C = M \oplus K$$

$$M = C \oplus K$$

↳ naziva se OTP („one-time-pad"):

- on je potpuno siguran ako je zadovoljeno:
  1. ključ je potpuno slučajan (ne pseudo-slučajan)
  2. ključ je jednake duljine kao i poruka te se mora sigurno prenijeti na drugu stranu

za svaku poruku!

⇓

ovo je glupo jer smo ondo i poruku te duljine mogli sigurno prenijeti (stoga, NIJE PROVEDIVO U PRAKSI!).

DES
(„Data Encription Standard")

⇒ nastao u IBM-u 1977. te se danas smatra nesigurnim

⇒ veličina ključa: 56 bita
(7 bajtova + 1 paritetni bajt) ⇒ ukupno 64 bita

⇒ ulaz: blok od 64 bita
↳ poruka se stoga mora podijelit na dijelove (blokove)
↳ stoga, takvi algoritmi se još nazivaju:
blok - simetrični algoritmi

⇒ ako radnji blok nije potpun, treba ga nadopunit
(engl. „padding")

⇒ BRUTE FORCE (gruba sila):
- isprobavanje svih mogućih ključeva

⇒ **DES je danas nesiguran zbog „male" veličine ključa**
↳ stoga, raspisan je natječaj za drugi standard i razvijen je Advanced Encription Standard (AES)
↳ u međuvremenu je dan međustandard (Tripple-DES) koji ima već ključ, ali je sporiji i nije prilagođen za današnja (64-bitna) računala //

⇒ USPJEŠAN NAPAD NA KRIPTOSUSTAV:

↳ svaki napad na kriptosustav koji ima složenost manju od grube sile

⇒ nezgodno svojstvo DES-a: (ne treba znati)

↳ ako kriptiramo simetričnu poruku sa simetričnim ključem, dobivamo simetričnu kriptiranu poruku, pa se prostor pretraživanja grubom silom efektivno smanjio na pola

$$C = DES(M, K)$$
$$C' = DES(M', K')$$

⇓

složenost se smanjila sa 2^{56} na 2^{55}, pa je ukupna složenost DES-a 2^{55}

no, ovo svojstvo olakšava dekripciju jer podključeve samo treba uzeti obratnim redoslijedom

⇒ izlaz: 64 bita

PR: Ako kriptiramo poruku od 100 bajtova DES-om, tada je izlazna poruka 104 bajtova jer mora biti višekratnik broj 8 (zbog „paddinga")

3DES

⇒ poruke kriptira sa tri ključa:

$$3DES(M, K1, K2, K3)$$

⇒ stoga, ponekad se koristi dva ključa, a ponekad tri:

↳ kada je dvostruka veličina ključa u odnosu na originalni DES, onda su K1 i K3 jednaki, pa je efektivna veličina ključa:

$$2 \cdot 56 = 112 \text{ bita}$$

↳ kada se koriste sva tri ključa, efektivna veličina ključa je:

$$3 \cdot 56 = 168 \text{ bita}$$

⇒ <u>ulaz</u>: blok od 64 bita

⇒ PROCES KRIPTIRANJA:

$$3DES(M, K1, K2, K3) = DES\left\{DES^{-1}\left[DES(M, K1), K2\right], K3\right\} = C$$

⇒ PROCES DEKRIPTIRANJA:

$$3DES^{-1}(C, K1, K2, K3) = DES^{-1}\left\{DES\left[DES^{-1}(C, K3), K2\right], K1\right\} = M$$

⇒ 3DES je 3 puta sporij od DES-a, ali se smatra sigurnim za današnja računala (bilo s dva ključa ili sa sva tri)

⇒ poboljšanje performansi:

↳ IZBIJELJENI DES (DESX):

$$C = DESX(M, K1, K2, K3) = DES(M \oplus K2, K1) \oplus K3$$

$$M = DESX^{-1}(C, K1, K2, K3) = DES^{-1}(C \oplus K3, K1) \oplus K2$$

IDEA

(„International Data Encription Algorithm")

⇒ veličina ključa: 128 bita

⇒ danas se smatra potpuno siguran

⇒ ulaz: blok od 64 bita

⇒ izlaz: blok od 64 bita

⇒ ima neke ranumljive koncepte koje su preuzeli novi algoritmi kriptiranja, pa čak i neke HASH funkcije

AES

(„Advanced Encryption Standard")

⇒ moguće veličine ključa:
128, 192 ili 256 bita
(sve varijante su sigurne)

⇒ ulaz: ne radi s bitovima, već s bajtovima
• 16 bajtova, ali u matrici 4×4, pa se kaže da je ulaz 2D

⇒ SCA („Side Channel Attacks"):
↳ napad koj uzimaju nuspojave kriptiranja kako bi doznali ključ
↳ npr. potrošnja struje, magnetsko polje, ...

⇒ ALGORITAM KRIPTIRANJA BLOKA: (općenito)
↳ algoritam prima tekst od „n"-bita te ga kriptira ključem od „k"-bita, izlaz je kriptirana poruka od „n"-bita
↳ siguran ako je napadaču „teško" na temelju kriptiranog teksta naći čisti tekst, čak uz puno parova (čisti, kriptirani) tekst

⇒ AES je otporan na grubu silu jer ima veliki ključ

⇒ KAKO RADI „AES"?

↳ iteracijski algoritam ∇ ;

↳ **koraci jedne iteracije:**

1. proširivanje ključa
   - ↳ deterministički algoritam koj od jednog ključa daje (najčešće) 10 ključeva
     - ↳ ako je jasno poznati (deterministički) algoritam
2. runde
   - ↳ ima ih 10 (jer je 10 ključeva)
   - ↳ u svakoj : radi XOR sa ključem, radi supstituciju (pomoću tablice) i permutaciju (izmješaj rezultat supstitucije)

$$f(\dots f(f(m \oplus k_1) \oplus k_2) \dots) \oplus k_{10}$$

↓

cijela mudrost AES-a je u konstrukciji funkcije runde „f"
(mora biti invertibilna kako bi mogli iz kriptiranog teksta dobiti originalni tekst)

- funkcija runde se sastoji od 3 operacije:
  1. Byte Sub (supstitucija) — UVODI NELINEARNOST
  2. Shift Rows } permutacija — UVODI DISPERZIJU
  3. Mix Columns }

↓ posljednja matrična množenja koje osigurava invertibilnost

↓ mala promjena ulaza daje veliku promjenu izlaza

⇒ AES je implementiran u hardware-u ;

↳ Intel ima instrukcije koje rade AES runde (kako je ovo strojna instrukcija, ovo je jako efikasno)

↳ AMD ima svoje strojne instrukcije

↳ stoga, ako kriptiramo hard-disk, ne gubimo gotovo ništa osjetno jer je dohvaćanje s diska nekoliko redova veličine sporije od strojnih instrukcija

NAČINI KRIPTIRANJA

⇒ kako blokove kriptirati ?

↳ sa AES-om znamo da su blokovi sigurni, no ako se blokovi ponavljaju, ukupna kriptirana poruka može podsjećat na originalnu poruku (npr. dvije slike)

(VIDI SLAJDOVE)

⇒ stoga, radimo CIPHER BLOCK CHAINING (CBC):

↳ prethodni blok koristimo za enkripciju sljedećeg bloka (promjene se propagiraju)

$$c_0 = IV \qquad \text{(inicijalizacijski vektor)}$$

$$c_k = E(m_k \oplus c_{k-1}, K)$$

$$m_k = D(c_k, K) \oplus c_{k-1}$$

↳ problem je što je teško paralelizirat dekripciju

↳ sprječili smo pojavljivanje uzoraka u kriptiranoj poruci (to se javlja u ECB-u)

↳ primjer analogije s random generatorom

(VIDI VIZUALIZACIJSKU SHEMU NA SLAJDU)

⇒ na slajdovima se upoznaj s drugim načinima povezivanja blokova prilikom enkripcije

(BITNO! ⇒ dolaze na ispitima)

ZAD: Razmotrimo jednu rundu AES-a i pronađimo jedan način da odredimo ključ na efikasan način ▽

$$1AES(m, K) = M(S(B(m \oplus K)))$$

↳ znamo: m

$$c = M(S(B(m \oplus K)))$$

⇓ sve što trebamo za ključ je...

$$K = B^{-1}(S^{-1}(M^{-1}(c))) \oplus m$$

↳ POANTA: AES mora koristiti barem dvije runde ▽

ASIMETRIČNI KRIPTOSUSTAVI

⇒ problem simetričnih kriptosustava:

- kako razmijeniti ključ između pošiljatelja i primatelja na siguran način?

⇒ **zbog navedenog razloga, razvijaju se asimetrični sustavi**

- primatelj ima dva ključa:
  - privatni: Δk_G
  - javni: $p k_G$
- pošiljatelj enkriptira pomoću javnog ključa „$p k_G$", šalje kriptiranu poruku, a primatelj prima i dekriptira pomoću srodnog privatnog ključa „Δk_G"

⇒ **uvjet sigurnosti:**

- napadač ne može lagano na temelju javnog ključa i kriptirane poruke dobiti originalnu poruku

⇒ ovi sustavi su kompleksniji od simetričnih te su nekoliko redova veličine sporiji od simetričnih

↳ dodatno, za asimetrične kriptosustave potrebno je poznavati osnove teorije brojeva

TEORIJA BROJEVA

I NOTACIJA

$N \Rightarrow$ prirodni broj

$p, q \Rightarrow$ prosti brojevi

$\mathbb{Z}_N \Rightarrow$ ograničen skup brojeva unutar kojeg su sve operacije „modulo N"

$$\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$$

$$a = b \text{ u } \mathbb{Z}_N \Leftrightarrow a \equiv b \pmod N$$

PR: Pogledajmo neke operacije:

$9 + 8 = 5$ u $\mathbb{Z}_{12}$

$5 \cdot 7 = 11$ u $\mathbb{Z}_{12}$

$7 - 9 = 10$ u $\mathbb{Z}_{12}$

} primjeti da nema dijeljenja

⇒ PROPOZICIJA 1: za aritmetiku u prstenu $\mathbb{Z}_N$, vrijedi standardna asocijativnost, distributivnost i komutativnost

⇒ PROPOZICIJA 2: ako su $X, Y \in \mathbb{Z}$ i $k = nzd(X, Y)$ onda postoje „a", „b" $\in \mathbb{Z}$ takvi da vrijedi:

$$aX + bY = k$$

i „k", „a" i „b" se mogu efikasno odredit proširenim Euklidovim algoritmom

⇒ odredimo sada „dijeljenje" u $\mathbb{Z}_N$:

- INVERZ od $x \in \mathbb{Z}_N$ je $Y \in \mathbb{Z}_N$ takav da vrijedi:

$$X \cdot Y = 1 \quad u \ \mathbb{Z}_N$$

- ako postoji, označavamo ga sa X^{-1}

PR: Pogledajmo računanje inverza:

Inverz od 2 u $\mathbb{Z}_9$? 5

Inverz od 4 u $\mathbb{Z}_{10}$? Ne postoji!

⇒ PROPOZICIJA 3: broj $x \in \mathbb{Z}_N$ ima inverz ako i samo ako je $nzd(x, N) = 1$

↳ dokazimo smjer →:

- po propoziciji 2, ako $\exists a, b \ ax + bN = 1$, onda je $a = x^{-1}$

(BITNO: pomoću proširenog Euklidovog algoritma možemo doznati postoji li inverz nekog broja u $\mathbb{Z}_N$)

⇒ DEFINICIJA:

↳ $\mathbb{Z}_N^*$ je podskup koji se sastoji od elemenata skupa $\mathbb{Z}_N$ koji su relativno prosti u odnosu na „N":

$$\mathbb{Z}_N^* = \{ x \in \mathbb{Z}_N : nzd(x, N) = 1 \}$$

PR: Pogledajmo primjere određivanja $\mathbb{Z}_N^*$:

↳ p - prost broj: $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$

↳ za 15: $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$

⇒ DEFINICIJA:

↳ Eulerova phi funkcija:

$$\varphi(N) = |\mathbb{Z}_N^*|$$

PR: Primjer računanja Eulerove phi funkcije:

↳ za p - prost broj: $\varphi(p) = p-1$

↳ za 15: $\varphi(15) = 8$

↳ za p, q - dva prosta broja:

$$\varphi(p \cdot q) = (p-1)(q-1)$$
$$= pq + 1 - (q-1) - (p-1)$$

(probaj ovo dokazati!)

⇒ TEOREM: (Eulerov teorem) BITNO!

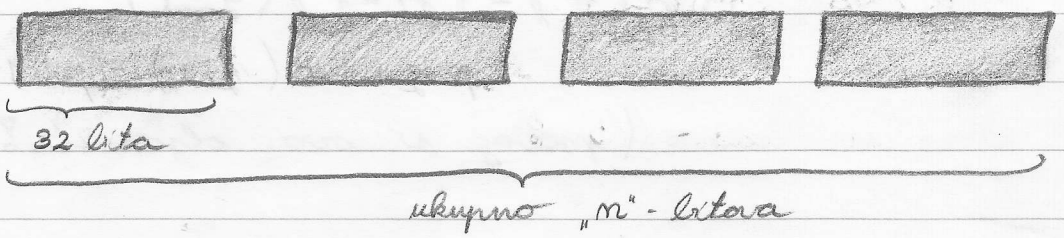
Za svaki N i svaki $a \in \mathbb{Z}_N^*$ vrijedi:

$$a^{\varphi(N)} = 1 \text{ u } \mathbb{Z}_N$$

RAČUNANJE S VELIKIM BROJEVIMA

⇒ kod asimetričnih algoritama, radimo s ekstremno velikim brojevima - brojevi će imati nekoliko tisuća bitova

⇒ velik broj raspodijeljeno u blokove veličine 32-bita:



↳ **zbrajanje**: isti „algoritam" kao u osnovnoj školi

$O(n)$

↳ **množenje**:

- tri opcije:
  - školski algoritam: $O(n^2)$
  - Karatsuba algoritam: $O(n^{1.6})$
  - algoritam temeljen na FFT: $O(n \log n \log\log n)$

↳ **inverz**: proširen Euklidov algoritam

$O(n^2)$

↳ **Mod Exp**: radimo ovu operaciju: $a^k \bmod N$

- možemo koristiti uzastopno kvadriranje:

$O(n^3)$

(uzastopno kvadriranje treba uvijek provesti za izlaz)

PR: Izračunaj 3^{53} u $\mathbb{Z}_{17}$ uzastopnim kvadriranjem!

$53 = 110101_2$ $\quad 53 = 32+16+4+1$

$3^{53} = 3^{32+16+4+1}$

$3^{53} = 3^{32} \cdot 3^{16} \cdot 3^{4} \cdot 3^{1}$

↳ radimo tablicu:

· u retku „k" želimo izračunat kolko je sada 3^{2^k}:

0	3^{2^0}	3^1	3	⇒ tog trebam	ovi navedeni se javljaju u rastavu
1	3^{2^1}	3^2	9		
2	3^{2^2}	3^4	13	⇒ tog trebam	
3	3^{2^3}	3^8	16		
4	3^{2^4}	3^{16}	1	⇒ tog trebam	
5	3^{2^5}	3^{32}	1	⇒ tog trebam	

↳ „k" ide dok u eksponenti ne dobijemo sve eksponente u rastavu našeg broja

$3^{53} = 5$ u $\mathbb{Z}_{17}$

KRIPTOSUSTAV RSA

⇒ **algoritam generiranja ključa G():**

↳ izabire dva slučajna velika prosta broja reda veličine 1024 bita:

• brojevi „p" i „q"

↳ pomnoži ih i izračuna vrijednost Eulerove funkcije:

$$N = p \cdot q, \quad \varphi(N) = (p-1)(q-1)$$

↳ odabere $e \in \mathbb{Z}_N$ takav da:

$$\text{nzd}(e, \varphi(N)) = 1$$

(u praksi: $e = 65537$)

↳ izračuna: $d = e^{-1}$ u $\mathbb{Z}_{\varphi(N)}$

↳ sada imamo:

JAVNI KLJUČ: $(e, N) = p_k$

PRIVATNI KLJUČ: $(d, N) = s_k$

⇒ **algoritam enkripcije $E(m, p_k)$:**

$$E(m, p_k) = m^e \text{ u } \mathbb{Z}_N$$

⇒ **algoritam dekripcije $D(c, s_k)$:**

$$D(c, s_k) = c^d \text{ u } \mathbb{Z}_N$$

⇒ primjeti da su enkripcija i dekripcija iste operacije (Mod Exp) samo sa drugim argumentima što olakšava implementaciju

↳ inače, enkripcija je brza jer za „e" obično uzimamo $65537 = 2^{16} - 1$ što olakšava operaciju Mod Exp

⇒ je li kriptosustav korektan?

↳ Jest! Pokažimo to:

$$D(E(m,(e,N)),(d,N)) = D(m^e,(d,N)) =$$

$$= (m^e)^d = m^{ed} = m^{1+k\cdot\varphi(N)} =$$

$$= m \cdot m^{k\cdot\varphi(N)} = m\cdot\left(m^{\varphi(N)}\right)^k = m \quad (u\ \mathbb{Z}_N)$$

↑ jednako 1 zbog Eulerovog teorema

↳ no, pomoću RSA možemo kriptirati brojeve koji su u $\mathbb{Z}_N^*$ → ako „m" nije u $\mathbb{Z}_N^*$, $nzd(m,N)$ je „p" ili „q" pa smo faktorizirali N što smo u startu htjeli da ne kriptiramo (stoga, ovo nije problem)

⇒ ako postoji efikasan algoritam za rastav na proste faktore, RSA nije siguran (jer se može izračunati N iz $\varphi(N)$, a time i privatni ključ iz javnog)

↳ zato „p" i „q" trebaju biti veliki!

⇒ je li RSA siguran kriptosustav?

↳ Nije! Običan RSA nije siguran!

↳ RSA je otporan na „brute-force" rastav na proste faktore, ali nije otporan na druge (pametnije) napade

• vidi probleme na slajdovima:

PR1: ne kriptiraj jedinicu jer je

$1^e = 1$
$2^e =$ nešto

} ako vidiš 1, glas je za 1, inače 2!

PR2: bajt ima 256 kombinacija, pa možemo sve moguće isprobati i dobiti enkriptirane poruke (sve moguće)

↳ nemoj slati nešto s malo entropije (informacije)

PR3: ako se ne desi modulo, samo uvadimo treći korijen

$c = K^3 \Rightarrow K = \sqrt[3]{c}$

↳ jer je K 128 bitni, najveći broj je 300 i nešto bitni, a N je 1024 bita, pa se nije desio modulo N

↳ pouka je da treba konstati veliki „e" 32 bita

PR 4: nešto složenij napad koji uspjeva u 20% slučajeva jer se desi da se „c" može ovako zapisati:

$k = k_1 k_2$ „k_1", „k_2" ~32 bita

$c = k_1^e k_2^e$

$c(k_1^e)^{-1} = k_2^e$

↳ „meet-in-the-middle" napad:

· radim tablicu 2^{32} broja:

k_2	k_2^e mod N
0	0
1	1
⋮	⋮
2^{32}	472...146

↳ sada u „for" petlji obilazim od 1 do 2^{32} da nađem vrijednosti koje odgovaraju jednadžbi:

$c(k_1^e)^{-1} = k_2^e$

⇒ kako bi izbjegli ovakve i druge „pemetne" napade, naprovljene su smjernice kako treba koristiti kriptosustav RSA

↳ zato jer „sirovi" RSA ima ovakve rupe!

⇒ NAPOMENA: ovakve primjere napada možeš očekivati na ispitima

SIGURAN SUSTAV KRIPTIRANJA JAVNIM KLJUČEM

⇒ **trebamo:**

a) RSA

b) neki simetrični kriptosustav (npr. AES u CBC modu)

c) neku kriptografsku „hash" funkciju

POŠILJATELJ		PRIMAOC
$x \leftarrow Z_N$		$x = RSA^{-1}(y, s_k)$
$k \leftarrow H(x)$	$\xrightarrow{(y,\, c)}$	$k = H(x)$
$y \leftarrow RSA(x, p_k)$		$m = D_s(c, k)$
$c \leftarrow E_s(m, k)$		

⇒ ovo je nastalo tako jer je iz onih napada zaključeno da je RSA siguran ako njime kriptiramo slučajan broj (nešto s puno entropije)

⇒ ako napadač dobije recimo pola bitova od „x", to mu ne pomaže jer „x" ide u hash funkciju da dobijemo sam ključ enkripcije, pa mu onih pola bitova ne pomaže jer hash to sve „poremeti" na gotovo slučajan način

SIMETRIČNA VS. ASIMETRIČNA KRIPTOGRAFIJA

⇒ pretpostavimo mail veličine 1kB:

↳ na procesoru i7-2600, treba nam:

- za AES: 850 ciklusa
- za RSA: 4 000 000 ciklusa

⇓

razlika je 3 reda veličine!

↳ to je zbog operacije ModExp koja uzima puno ciklusa

⇒ stoga, za vrijeme jedne enkripcije RSA kriptosustavom, mogu enkriptirat 5MiB podataka AES-om u CBC modu

⇒ POUKA: simetričan ključ izmjeni RSA kriptosustavom, a same podatke kriptiraj AES-om u CBC modu!

GENERIRANJE VELIKIH SLUČAJNIH PROSTIH BROJEVA

⇒ za algoritam RSA trebam odmah u prvom koraku dva velika slučajna prosta broja, pa pogledajmo kako to napravit

⇒ NAIVAN ALGORITAM:

```
def gen_prost(n):
    while True:
        p = n-bitni broj
        if prost(p):
            return p
```

⇒ funkcija gustoće prostih brojeva $\pi(n)$:

↳ veli koliko su gust prosti brojevi od „n" bita

↳ ispada da za 1024 bita, otprilike je svaki 700-ti broj prost, pa naivan algoritam ima smisla (no još trebamo efikasnu provjeru je li broj prost)

⇒ **kako provjeriti je li broj prost na efikasan način?**

↳ pa, ne znamo efikasan način, ali možemo napisati funkciju **vjerojatno_prost** koja je efikasna

↓

- ako ona vrati False, broj je sigurno složen (sigurno nije prost)
- ako ona vrati True, broj je vrlo vrlo vjerojatno prost (vjerojatnost pogreške je astronomski mala $\sim \frac{1}{2^{100}}$)

```
def vjerojatno_prost(p):
    ponavljaj „t" puta:
        a = slučajan broj manji od p
        if svjedok_složenosti(a, p):
            return False
    return True
```

↳ „t" ćemo tako naštelati da vjerojatnost pogreške bude astronomski mala, a funkcija još uvijek bude efikasna!

⇒ pogledajmo funkciju svjedok_složenosti(a, p):

↳ funkcija je ispravan svjedok ako ona vraća True samo kada je „p" složen

↳ primjer ispravnog svjedoka:

no, ovaj svjedok nije efikasan

```
def s1(a, p):
    return nzd(a, p) != 1
```

↳ efikasan svjedok: funkcija je efikasan svjedok ako za složeni „p" i za slučajni „a" (< „p") funkcija vraća True s neznatnom vjerojatnošću

↓

laički rečeno, postoji puno „a"-eva koji bi mogli posvjedočiti složenost

⇒ FERMATOV TEOREM:

↳ „p" je prost ako je za „a" (< „p"):

$$a^{p-1} = 1 \quad u \quad \mathbb{Z}_p$$

```
def svjedok_fermat(a, p):
    if a^(p-1) mod p != 1:
        return True
    return False
```

ispravan i efikasan za skoro sve brojeve, osim za rijetke Carmichaelove brojeve (za njih ne postoje Fermatovi svjedoci)

⇒ MILLER - RABINOV SVJEDOK:

↳ pogledajmo ideju:

$$X^2 = 1 \quad u \quad \mathbb{Z}_p$$

↓

ako je „p" prost, onda jednadžba ima samo trivijalna rješenja jer „p" dijeli $(X-1)\cdot(X+1)$

• matematički rapisano:

$$p \mid (X-1)(X+1)$$

↳ to je zato jer ako „p" dijeli umnožak, onda dijeli jednog ili drugog (ili oba)

↳ ako je „p" prost, onda su rješenja: $X = \pm 1$ u $\mathbb{Z}_p$

• matematički rapisano:

„p" prost ⇒ $X = \pm 1$ u $\mathbb{Z}_p$

↳ sada općenito:

$$a^{p-1} = 1$$

p ⇒ prost, (p-1) ⇒ nije prost (paran)

$$\left(a^{\frac{p-1}{2}}\right)^2 = 1$$

⇓

ovo rješavamo i imamo 3 mogućnosti:

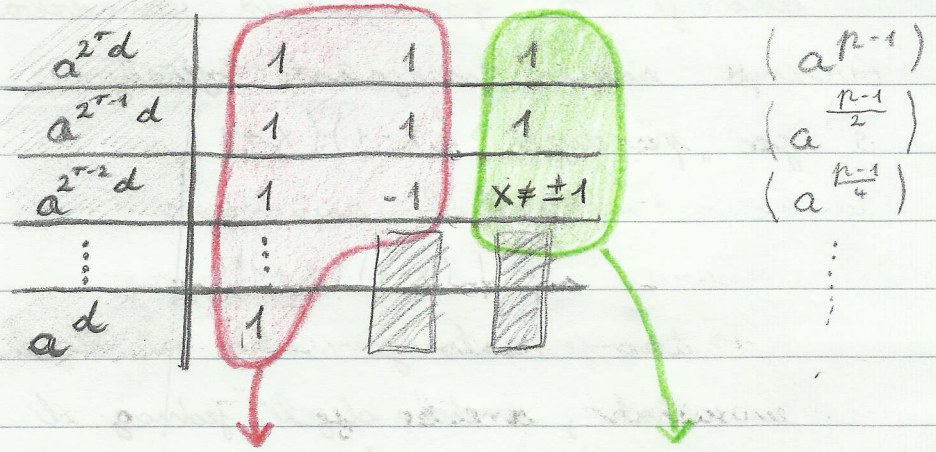
a) $X \neq \pm 1$ ⇒ broj je složen (korijen veći od 1)

b) $X = -1$ ⇒ odustajemo

c) $X = 1$ ⇒ razmatramo $a^{\frac{p-1}{4}}$ istim postupkom

↳ da bi dobili tijek algoritma, vidimo koliko puta možemo podijelit $(p-1)$ sa dva:

$$(p-1) = 2^r d \qquad d \text{ je neparan}$$



u ovim slučajevima, vraćamo False (nismo našli svjedoka)

u ovom slučaju, vraćamo True jer je broj netrivijalni korijen od „p"

```
def svjedok_miller_rabin(a, p):
    p-1 = 2^r d
    x = a^d mod p
    if x ∈ {-1, 1}:
        return False
    ponavljaj r puta:
        x = x·x mod p
        if x == 1:
            return True
        if x == -1:
            return False
    return False
```

⇒ TEOREM 3 (nije neophodno znati)

↳ ako je „p" složen, onda će za barem 50% brojeva „a" ($< „p"$) funkcija svjedok-miller-rabin dati True

⇓

ovo je dokazano matematički, a za nas to znači da je funkcija svjedok-miller-rabin u knjizi u pola slučajeva.

↳ ako radimo 100 iteracija, vjerojatnost greške je $\frac{1}{2^{100}}$ što je za nas OK jer je veća vjerojatnost da napadač baš pogodi naš ključ

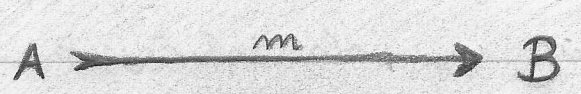
↳ ovaj teorem govori i o efikasnosti našeg svjedoka po Milleru i Rabinu !?

DIGITALNI POTPIS

⇒ po defaultu ne osigurava tajnost

⇒ **DIGITALNI POTPIS:**

↳ ideja s dva ključa kod svake strane



m = " Pošalji 1000 kn s računa X na račun Y "

↳ želimo sljedeća svojstva:

a) AUTENTIČNOST - poruku je stvarno napisao A

b) INTEGRITET - poruka nije izmijenjena

c) NEPORECIVOST - A ne može poreći da je poslao „m"

↳ prije slanja, potpis računamo na temelju poruke i svog privatnog ključa

↳ šaljemo poruku i potpis (nema tajnosti !!)

↳ primatelj provjerava potpis pomoću javnog ključa pošiljatelja i utvrđuje je li potpis u redu

↳ matematički, digitalni potpis je trojka efikasnih algoritama:

(G, S, V)

- G: generiranje ključeva
- S: algoritam potpisivanja
- V: algoritam verifikacije potpisa

⇒ sustav digitalnog potpisa je siguran ako je teško krivotvoriti potpis, čak i ako je dostupno puno drugih potpisa

⇒ super je da RSA možemo koristit kao digitalni potpis na sljedeći način (nesiguran potpis):

- generiranje ključeva $G()$ je isto kao u običnom RSA i rezultat je: (e, N), (d, N)
- **potpis:**

$$S(m, (d, N)) = m^d \mod N$$

- **verifikacija:**

$$V(m, s, (e, N)) = s^e == m \mod N$$

↳ sigurnost:

↳ ja provjeravam je li $s^e = m$ u $\mathbb{Z}_N$

↳ kako je $s = m^d$ u $\mathbb{Z}_N$, m^d je mogao stvarati samo onaj tko zna „d"

BITNO!
(ispit)

PR: Pogledajmo mogući napad na digitalni potpis!

↳ dvije poruke:

m_1 , $s_1 = m_1^d$

m_2 , $s_2 = m^d$

$m_1 m_2$, $s_1 s_2 = m_1^d m_2^d = (m_1 m_2)^d$

⇓

ovako možemo lagano poslati nekoliko dvije poruke koje je netko drugi potpisao

⇒ ISPRAVLJANJE NEDOSTATKA:

- trebamo dvije dodatne funkcije:

  H – kriptografska hash funkcija

  Pad – dodavanje "paddinga" kako bi dobili jako "veliki" broj

- **potpis**:

  $$s = RSA^{-1}(Pad(H(m)), s_k)$$

- **verifikacija**:

  $$Pad(H(m)) == RSA(s, p_k).$$

siguran digitalni potpis //

⇒ DIGITALNI PEČAT:

↳ digitalna omotnica + digitalni potpis

↳ osigurava:

- tajnost
- integritet
- autentičnost
- neporecivost

HASH FUNKCIJE

⇒ željena svojstva:

a) efikasna
b) deterministička
c) javna
d) izgleda slučajno

$$\{0,1\}^* \longrightarrow \{0,1\}^{256}$$

⇒ hash-funkcija nije dobra ako je za zadan output lagano (ili nije teško) naći input

↳ primjer: ako dvije poruke imaju ist hash, onda one imaju i ist digitalni potpis (stoga, da hash-funkcija nije dobra, digitalni potpis nebi bio siguran)

⇒ ovo se svojstvo naziva i OTPORNOST NA KOLIZIJE:

"nemoguće je pronaći bilo koje dvije poruke M_1 i M_2 za koje se dobiva ist sažetak"

↳ NAPOMENA: kolizije postoje, al su one rijetke

⇒ NnapAD: hash-funkcija je napadnuta ako smo našli kolizije

⇒ vidi paradoks rođendana u sklopu hash funkcija ∇.

MD5

⇒ „Message Digest" (version 5)

⇒ 128 bita (sažetak na izlazu)

↳ zbog „rođendanskog napada", prostor pretraživanja se smanjuje na korijen:

$2^{128} \xrightarrow{\sqrt{}} 2^{64}$

↳ stoga, na današnjim računalima ovo je premalo (isti problem kao kod DES-a)

⇒ drugi problem: što ako je ovo što hashiramo „malo"?

↳ je li moguće naći poruku koju smo hashirali?

(npr. ako hashiramo 128 bitni AES ključ)

⇓

moderne hash funkcije nemaju taj problem! } za MD5 ovo nije veliki problem

⇒ treći problem: kolizije - moguće je naći dvije poruke koje daju isti sažetak (u realnom vremenu)!

⇓

na MD5 je ovo danas moguće čak i na „smartphone"-u!

⇒ koristi se samo za „legacy" razloga

⇒ KAKO RADI MD5?

- ima 64 koraka (4 kruga po 16 koraka)
- ulazni blok: 512 bita

↓

ako radnji blok nije 512 bita, nakon radnjeg bita 1 slijede nule do 64 bita prije kraja u koj se upisuje duljina poruke u bitovima (ako nema mjesta na 64 bita u tom bloku, dodaje se novi sa samo nulama i u radnjih 64 bita piše duljina poruke)

PADDING (ovo treba znati?)

⇒ primjeti da je blok 512 bita jer je to 16 x 32 (MD5 je prilagođen 32-bitnim računalima)

⇒ MD5 nije standardiziran

- razvijan je natječoj i dan je SHA-0
- ubrzo nakon, NSA daje SHA-1 koj se razlikuje od SHA-0 u samo jednom bitu
- tri godine nakon, NSA predlaže SHA-2 koj je samo „upgrade" algoritma SHA-1 (vrlo slično)
- SHA-3 je proračen 2012. godine te je drastično različit od prijašnjih algoritama (SHA-0, SHA-1 i SHA-2 su jako inspirirani algoritmom MD5)

SHA - 1

(Secure Hash Algorithm)

⇒ sažetak na izlazu:
160 bita

⇒ korjen je 80 bita što je danas još sigurno za napad grubom silom (ali je problem s kolizijama)

⇒ računanje riječi W; samo ima dodan „rotate" (rotaciju) za jedan bit te je to jedina razlika u odnosu na SHA-0

⇒ blok na ulazu je isto 512 bita

DODATNA VAŽNA SVOJSTVA HASH FUNKCIJA

COLLISION RESISTANCE

↳ nije moguće naći dvije iste poruke koje daju ist sažetak

SECOND PREIMAGE RESISTANCE

↳ za poznati M i H=h(M) je nemoguće pronaći M' koj daje ist H

(puno teže je provaći ovo jer je M unaprijed zadan)

⇒ DIFUZIJA: mala promjena ulaza daje veliku promjenu na izlazu

⇒ još treba znat „first-preimage-resistance" (vidi knjigu)

SHA-2

⇒ **postoje 4 algoritma:**

a) SHA-224
b) SHA-256
} za 32-bitne arhitekture

c) SHA-384
d) SHA-512
} za 64-bitne arhitekture

⇒ napomena: kako se ne bi zbunio između SHA-2 i SHA-3, možda je bolje pisat SHA-2-<broj_bitova>

⇒ novost u odnosu na SHA-1 je da se uvodi operacija pomaka („shift"), pa se neki bitovi gube
↳ ima i drugih manjih promjena (npr. više konstanti)
↳ ta sličnost nije dobra jer to znači da imamo vrlo slične slabosti kao i SHA-1 i MD5

SHA - 3

⇒ danas je to važeći standard

⇒ ista 4 algoritma kao kod SHA-2:

a) SHA-224

b) SHA-256

c) SHA-384

d) SHA-512

⇒ padding: M10*1

↳ kod ovih prije je bilo M10*(64-bita na duljinu)

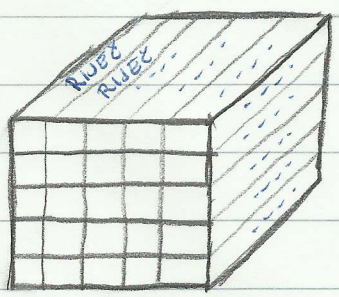
⇒ SPUŽVASTA KONSTRUKCIJA:

↳ dvije faze:

a) upijanje

b) cijeđenje

⇒ imamo 3-D matricu:



} svaka riječ je promjenjive duljine (omogućuje proširivost i rad na malim („embeded") sustavima)

⇒ ukupan broj koraka ovisi o veličini riječi:

- nije potrebno znati kako on ovisi, al samo znaj za tu ovisnost

⇒ u svakom koraku se na našu 3D matricu latova primjenjuje 5 funkcija

(samo znaj za „5 funkcija u svakom koraku")

SIGURNOSNI PROTOKOLI

⇒ sigurnosni protokol su sastavni dio svakog modernog operacijskog sustava

↳ za njih, potrebna nam je kriptografija (zato smo učili sve ove kriptografske algoritme)

DIFFIE - HELLMANOV PROTOKOL

⇒ prinjeti da je ovo protokol, a ne kriptografski algoritam poput RSA

⇒ služi za razmjenu tajnog ključa

⇒ Ana i Branko žele razmjenit tajni ključ:

1) dogovore se za dva velika broja „p" i „g" za koje vrijedi:

$$nzd(g, p) = 1$$

(naglašeno je da je „p" prost broj)

2) Ana generira nasumičan broj „X", i iz njega računa „X" i šalje ga Branku:

$$X = g^x \bmod (p)$$

3) Branko radi isto samo s „Y" i šalje ga Ani:

$$Y = g^y \bmod (p)$$

4) Ana računa: $K_1 = Y^x \bmod (p)$

5) Branko računa: $K_2 = X^y \bmod (p)$

↳ prinjeti da su K_1 i K_2 isti jer je:

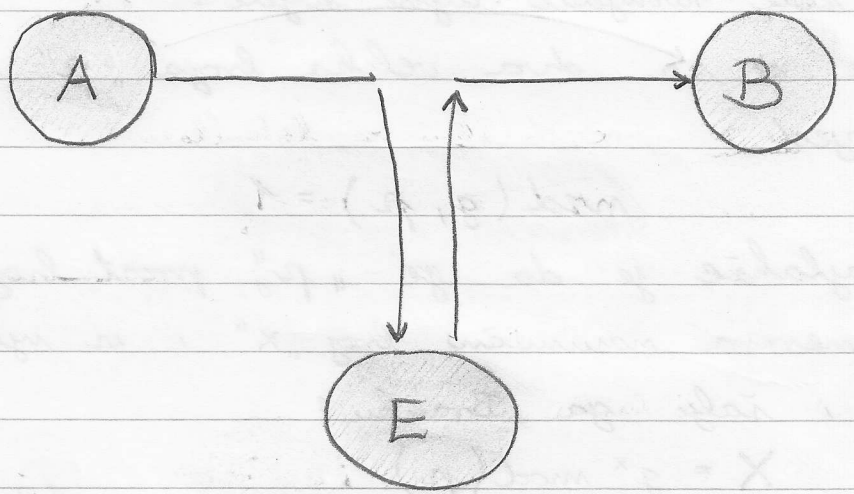
$$Y^x \bmod (p) = g^{xy} \bmod (p) = X^y \bmod (p)$$

↳ stoga, ključ je zapravo:

$$K \equiv K_1 \equiv K_2$$

⇒ NAPAD „ČOVJEK U SREDINI"

- Eva presreće i dogovori se za jedan ključ s Anom, a drugi s Brankom
- tada može čitati sve poruke od jednog, odmotati ih, opet zamotati (potencijalno izmjenite) i poslat ih drugome bez da taj drugi išta sumnja



⇒ RJEŠENJE:

↳ rješenje napada „čovjek-u-sredini" je da se Ana i Branko autentificiraju (najčešće certifikatima)

↳ vid video od Computerphile-a na temu:

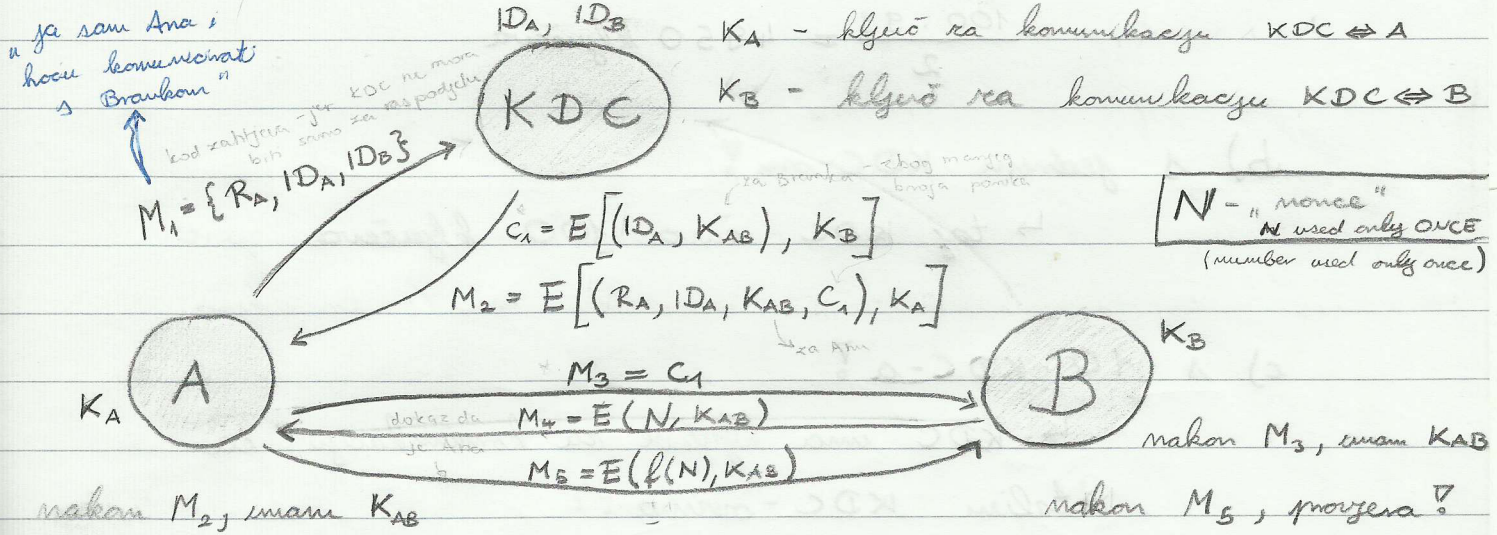
Man-in-the-Middle napada

(priča Tom Scott)

RASPODJELA KLJUČEVA U ZATVORENOM SIMETRIČNOM KRIPTOSUSTAVU

⇒ kako napraviti raspodjelu ključeva s puno korisnika na efikasan način (da nema ključ nakon na nakon na stalno)?

↳ uvodi se CENTAR ZA RASPODJELU KLJUČEVA (engl. „Key Distribution Center" - KDC)



⇒ raspodjela prema Needhamu i Schroederu:

- slika gore prikazuje zahtjev da Ana komunicira s Brankom
- ključ K_{AB} se generira nasumično za svaku komunikaciju i nakon toga se briše

RASPODIJELJENA RASPODJELA KLJUČEVA

⇒ ne možemo imati jedan centar jer ako imamo jako veliki broj korisnika, dolazi do raguženja

PR: Ako imamo 100 korisnika u ratvorenom kriptosustavu, odredi broj ključeva?

a) bez KDC-a?

$$\frac{100 \cdot 99}{2} = 4950 \text{ ključeva}$$

b) s jednim KDC-om?

↳ taj KDC čuva 100 ključeva

c) s 10 KDC-a?

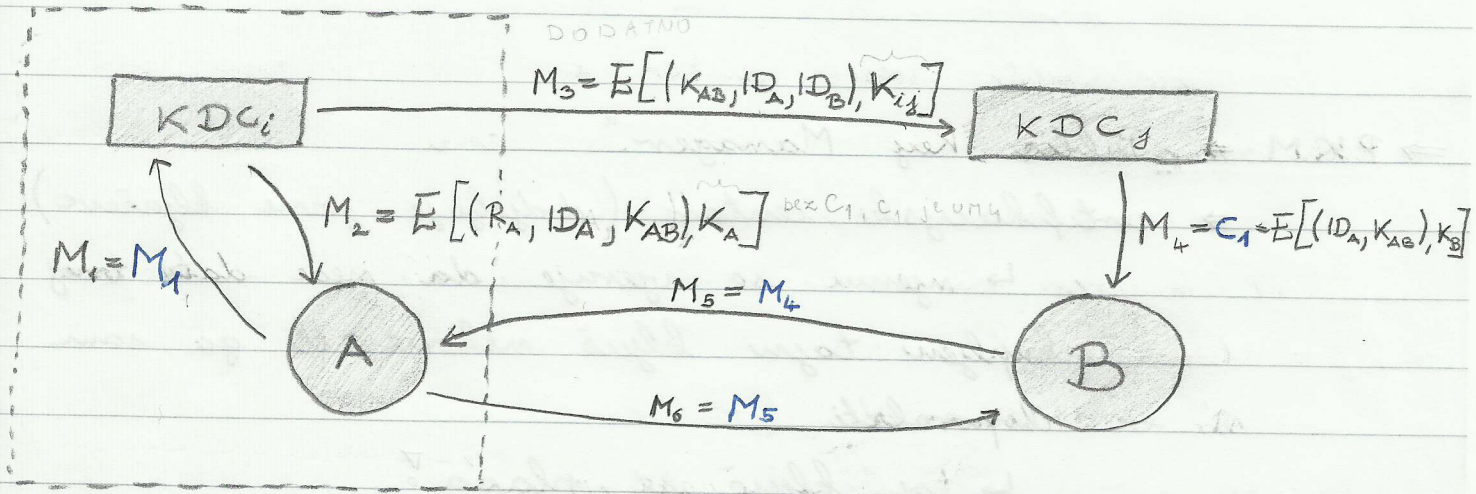
↳ KDC ima ključeve za komunikaciju sa ostalim KDC-ovima:

$$\frac{10 \cdot 9}{2} = 45$$

↳ pošto imamo 100 korisnika, trebamo još 100 ključeva, pa je ukupno u sustavu 145 ključeva, ali je bolje jer je svaki centar puno manje (≈10 puta) opterećen nego centar pod b)

⇒ skicirajmo komunikaciju između Ane i Branka koj se ne nalaze pod istim KDC-ovima:

(ANA ŠALJE BRANKU!?)



↳ ovo je jako slično kao s jednim KDC-om

(jedina nova poruka je M_3 ⇒ komunikacija KCD-ova)

↳ pisano penkalom je isto kao prethodni protokol

⇒ NAPOMENA: svaki centar ima uvid u listu ID-a korisnika svih drugih centara

RASPODJELA KLJUČEVA U ZATVORENOM ASIMETRIČNOM KRIPTOSUSTAVU

⇒ PKM ⇒ „Public Key Manager"

⇒ certifikacijski centar (dodjeljuje javne ključeve)

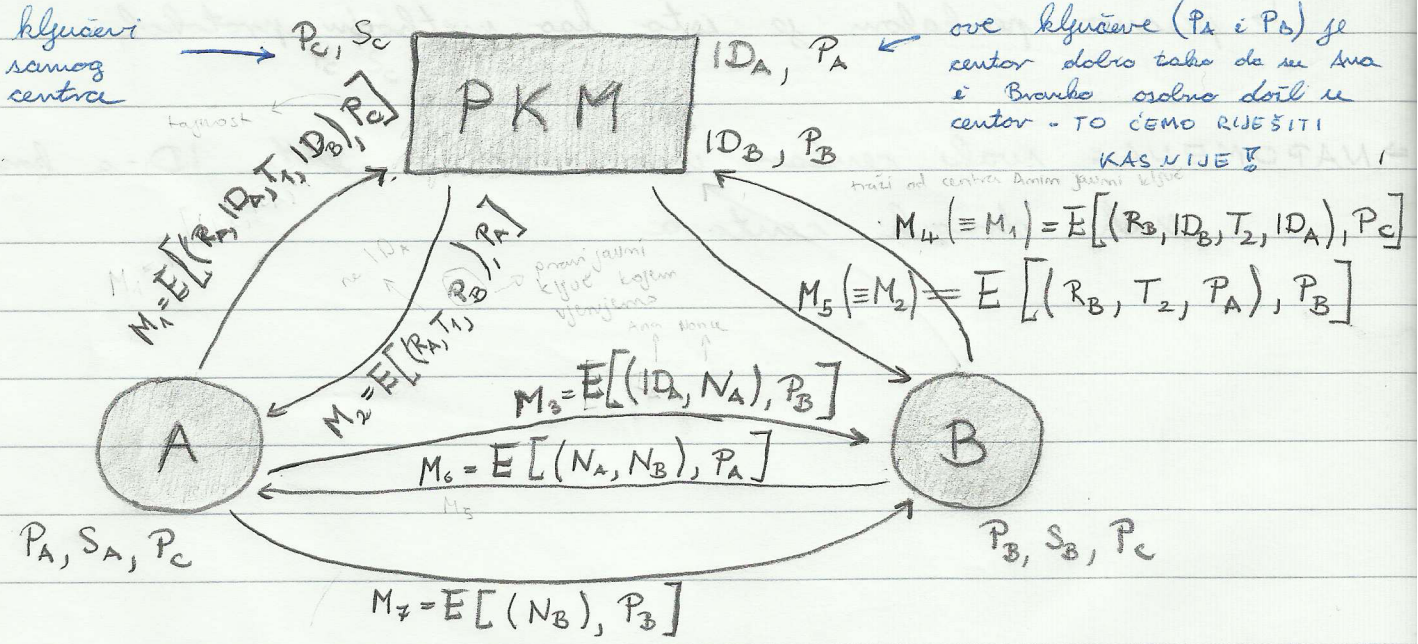
↳ njemu se vjeruje da neće dati tvoj kupljen tajni ključ nekome il' ga sam zloporabiti

↳ taj ključ se plaća ▽

⇒ ponovno, pogledaj spomenuti video od Computerphile-a ▽

(ANA ŠALJE BRANKU)

⇒ skicirajmo komunikaciju između Ane i Branka:



P ⇒ javni ključ S ⇒ privatni ključ

T_i ⇒ vremenska oznaka

⇒ vremenska oznaka T_i ⇒ služi da se spriječi dvostruko ponavno odašiljanje poruke

⇒ za asimetrične kriptosustave, potrebno je da je poruka koja se kriptira manja od dužine ključa

⇒ NAPOMENA: pretpostavimo da za M_2, PKM nije provjerio je li vlasnik M_1 stvarno od Ane, ali nema veze jer napadač nema privatni ključ koji je par od P_A, pa neće moći dekriptirati poruku

⇒ NAPOMENA: sa porukom M_7, Branko se potvrđuje da Ana sa njim komunicira

(M_6 i M_7 služe istu svrhu kao i M_5 i M_6 u prošlom ciklusu)

AUTENTIFIKACIJA

⇒ može biti jednostrana i obostrana

⇒ <u>jednostrana</u>:

- onaj koji inicira komunikaciju treba biti provjeren

⇒ R_i ⇒ kôd rahtjeva

⇒ definira kakav je kriptosustav (simetričan, asimetričan) i ostale stvari vezane uz rahtjev

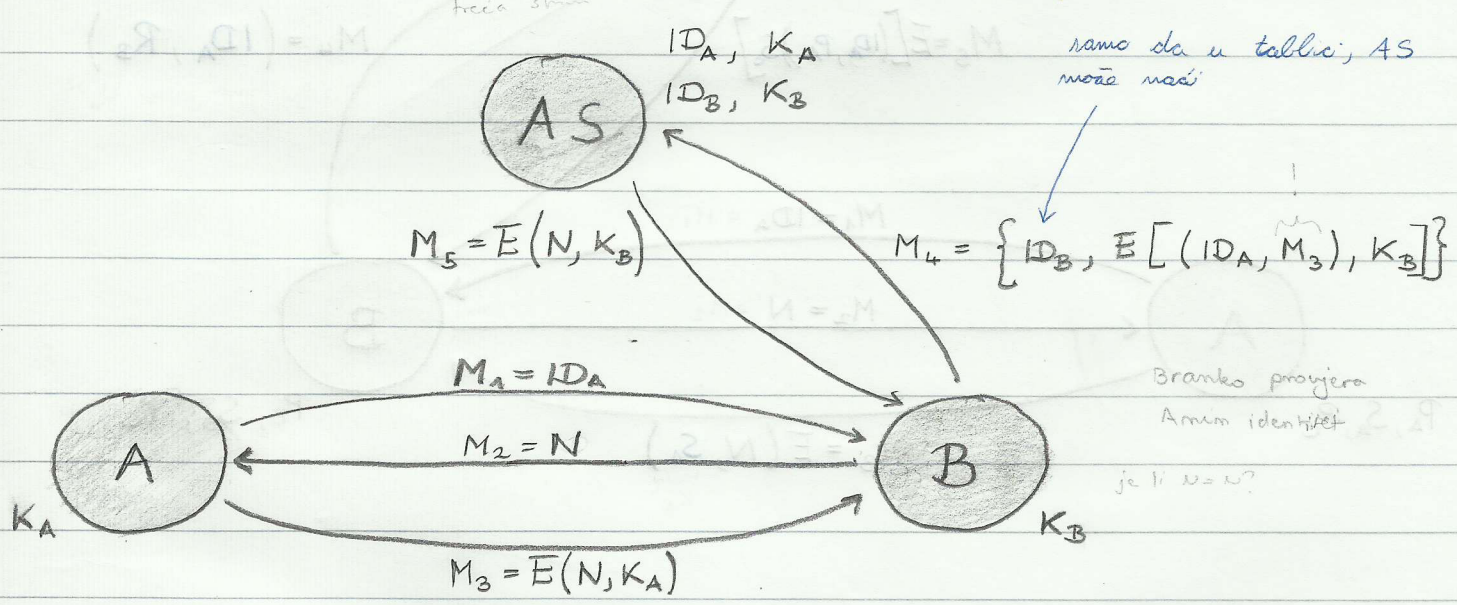
JEDNOSTRANA AUTENTIFIKACIJA U ZATVORENOM SIMETRIČNOM KRIPTOSUSTAVU

⇒ sada nemamo KDC, nego AS

⇒ AS ⇒ Autentifikacijski centar („Authentification Server")

⇒ skicirajmo autentifikaciju između Ane i Branka:

↳ Ana inicira komunikaciju, pa Branko želi autentificirat Anu (želi se uvjerit da stvarno komunicira s Anom)



⇒ ako je Branko dobio N koj je poslao, tada je siguran da razgovara s Anom jer je jedino ona mogla s K_A enkriptirat N

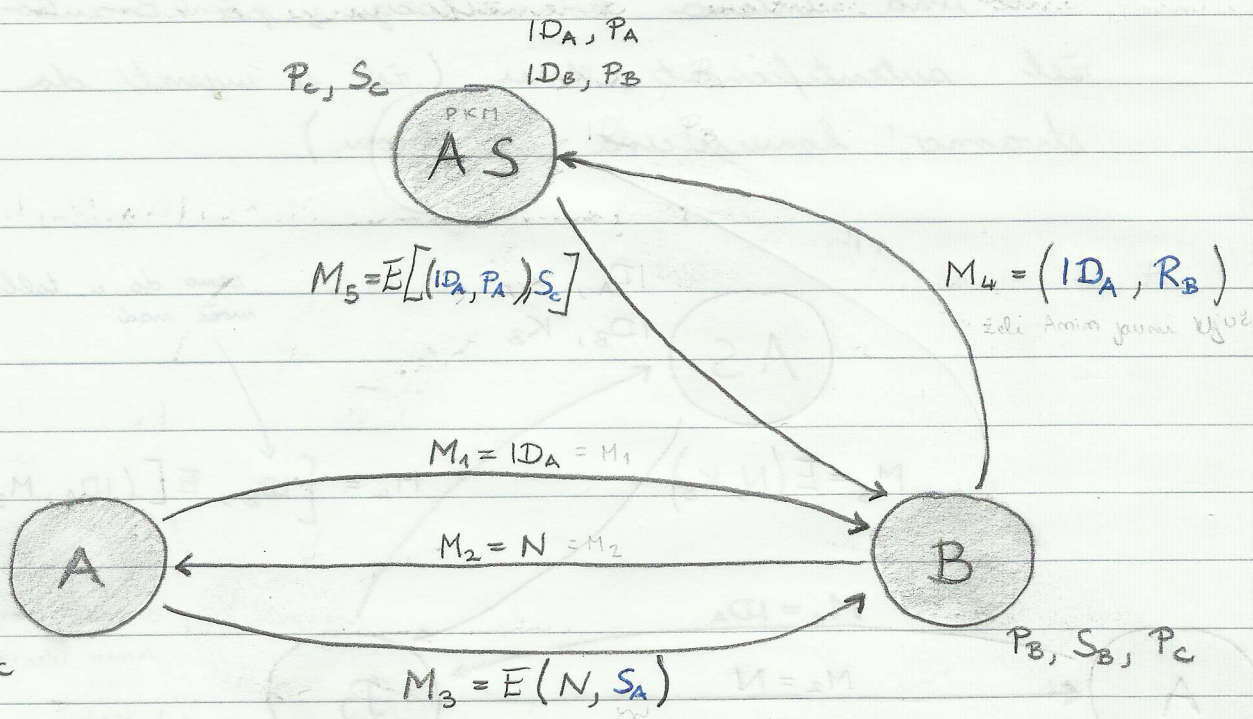
JEDNOSTRANA AUTENTIFIKACIJA U ZATVORENOM ASIMETRIČNOM KRIPTOSUSTAVU

⇒ *isti je cilj kao u simetričnom sustavu*

↳ *vidjet ćemo da i rješenje nije jako razlikuje ta*

(ono što je razlikuje je u boji ?!)

⇒ *ponovno, Branko želi autentificirat Anu:*



⇒ *Branko nakon što primi* M_5 *može otključat tu poruku (signuo od centra jer je potpisana) i uzet iz nje* P_A

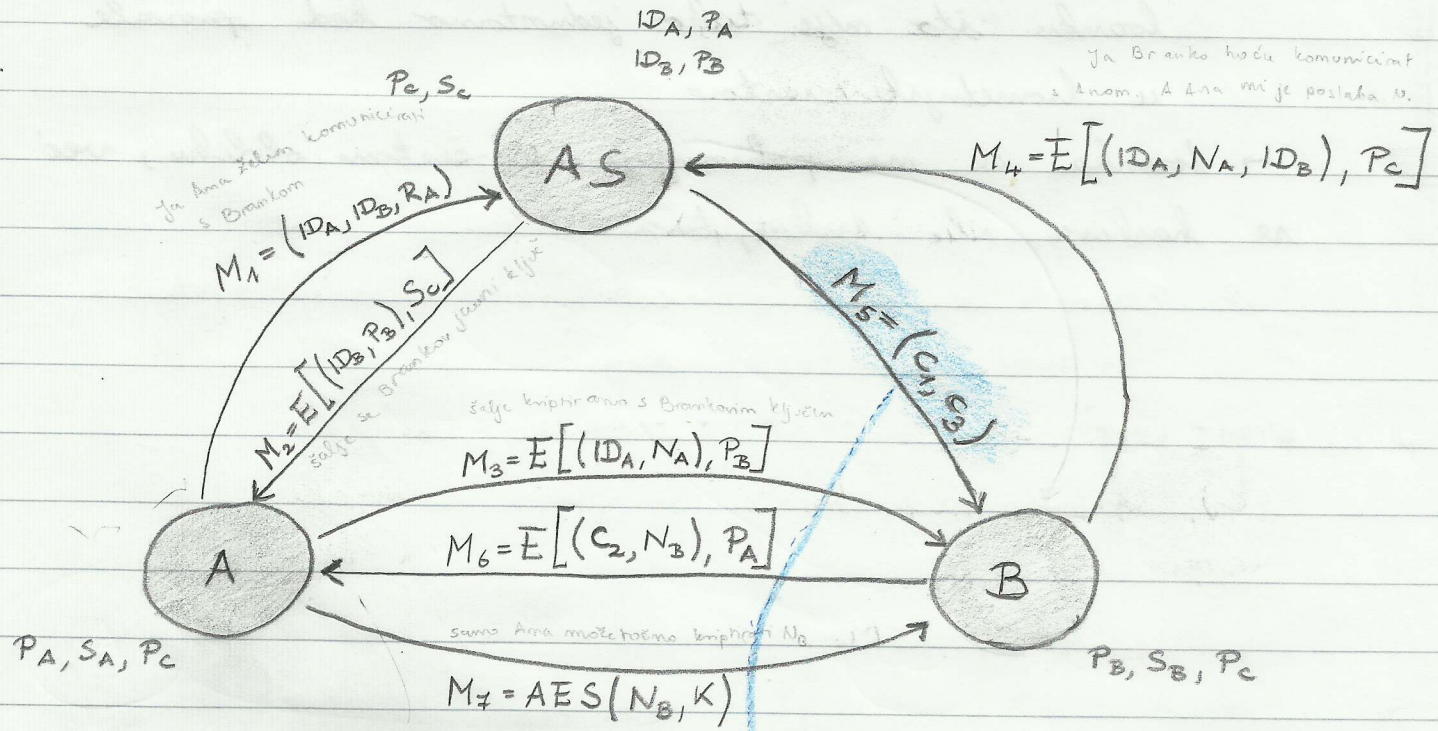
↳ *sada može otključati* M_3 *i usporediti poslani i dobiveni* N *kako bi autentificirao Anu*

OBOSTRANA AUTENTIFIKACIJA U ZATVORENOM ASIMETRIČNOM KRIPTOSUSTAVU

⇒ možemo naprovit dvije jednostrane, no to je razbacivanje porukama

↳ umjesto 10 poruka (2 x 5) možemo naprovit to u manje poruka!

⇒ skicirajmo obostranu autentifikaciju Ane i Branka:



⇒ nakon M_6, Ana autentificira Branka

⇒ nakon M_7, Branko autentificira Ane

$C_1 = E[(ID_A, P_A), S_C] \equiv M_2$ (Anin javni ključ)

$C_2 = E[(ID_B, K, N_A), S_C]$ tajni k

$C_3 = E(C_2, P_B)$ → samo Branko može pročitat zbog tajnog ključa K

K ⇒ simetrični ključ za komunikaciju Ane i Branka nakon autentifikacije (razmjenjuje se jer se može i usput se iskoristi, ali nije nužno) (slučajan je svaki put)

PRIJAVA ZA RAD

⇒ obično se koristi:

KORISNIČKO IME + LOZINKA

↳ nedostaci:

- lozinka treba imati svojstva da se ne može lagano slomit grubom silom (no, kako pamtit duge lozinke?)

↳ dobra strana:

- ako je sustav kompromitiran, možemo promijenit lozinku što nije tako jednostavno kod provale u biometrijski sustav

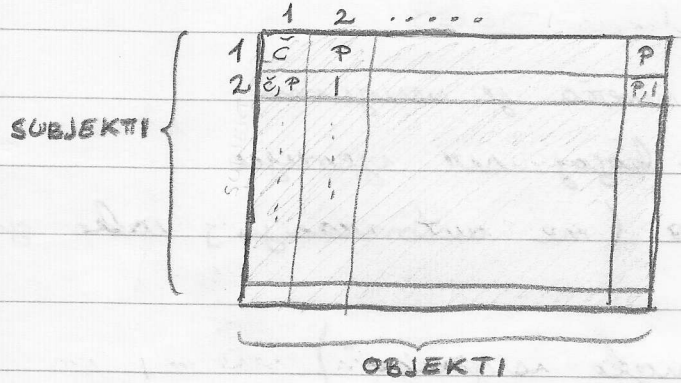
↳ lozinka se ne pohranjuje u čistom obliku, već se hashira ili enkriptira

AUTORIZACIJA

⇒ nakon prijave, slijedi autorizacija

↳ autorizacija je dodjela prava prijavljenom korisniku (npr. pisanje, čitanje, izvršavanje, ...)

⇒ postoji tablica (koja je rijetka):



OBJEKT: datoteke, direktive, ...

SUBJEKT: korisnik, proces, ...

↳ kako je tablica rijetka, bolje je koristit liste:

1) LISTA PRAVA PRISTUPA OBJEKTU

2) LISTA DOZVOLA ZA PRISTUP OBJEKTIMA

(engl. Capability Tickets)

AUTENTIFIKACIJSKI PROTOKOL „KERBEROS"

⇒ koristi se jako puno (npr. za login u operacijski sustav Windows u online modu)

⇒ razvijen 1978 godine na MIT-u

⇒ simetrični kriptosustav

⇒ PRETPOSTAVKE:

- računala su sigurna, mreža je nesigurna
- postoj treća strana kojoj svi vjeruju

⇒ služi samo za autentifikaciju (ne autorizaciju, iako ga je trivijalno proširiti)

⇒ samo 1 unos lozinke („single sign-on")

⇒ lozinka ne putuje mrežom!

- niti hashirana, niti kriptirana! BITNO!

⇒ ARHITEKTURA:

a) čvor klijenta

b) čvor poslužitelja

c) čvor KERBEROS poslužitelj

⇒ ima više klijenata i više poslužitelja

↳ samo jedan čvor Kerberos

→ o čvoru KERBEROS:

- mjerenu se vjeruje
- fizički zaštićen i nije spojen na Internet
- sastoj se od:

1) BAZA PODATAKA:
   ↳ id + lozinka
   ↳ id + K (tajni ključ)
2) AS
3) TGS - engl. „Ticket Granting Server"

⇒ shema protokola:

IDS ⇒ „id" usluge

↳ nakon M_1: AS računa ključ

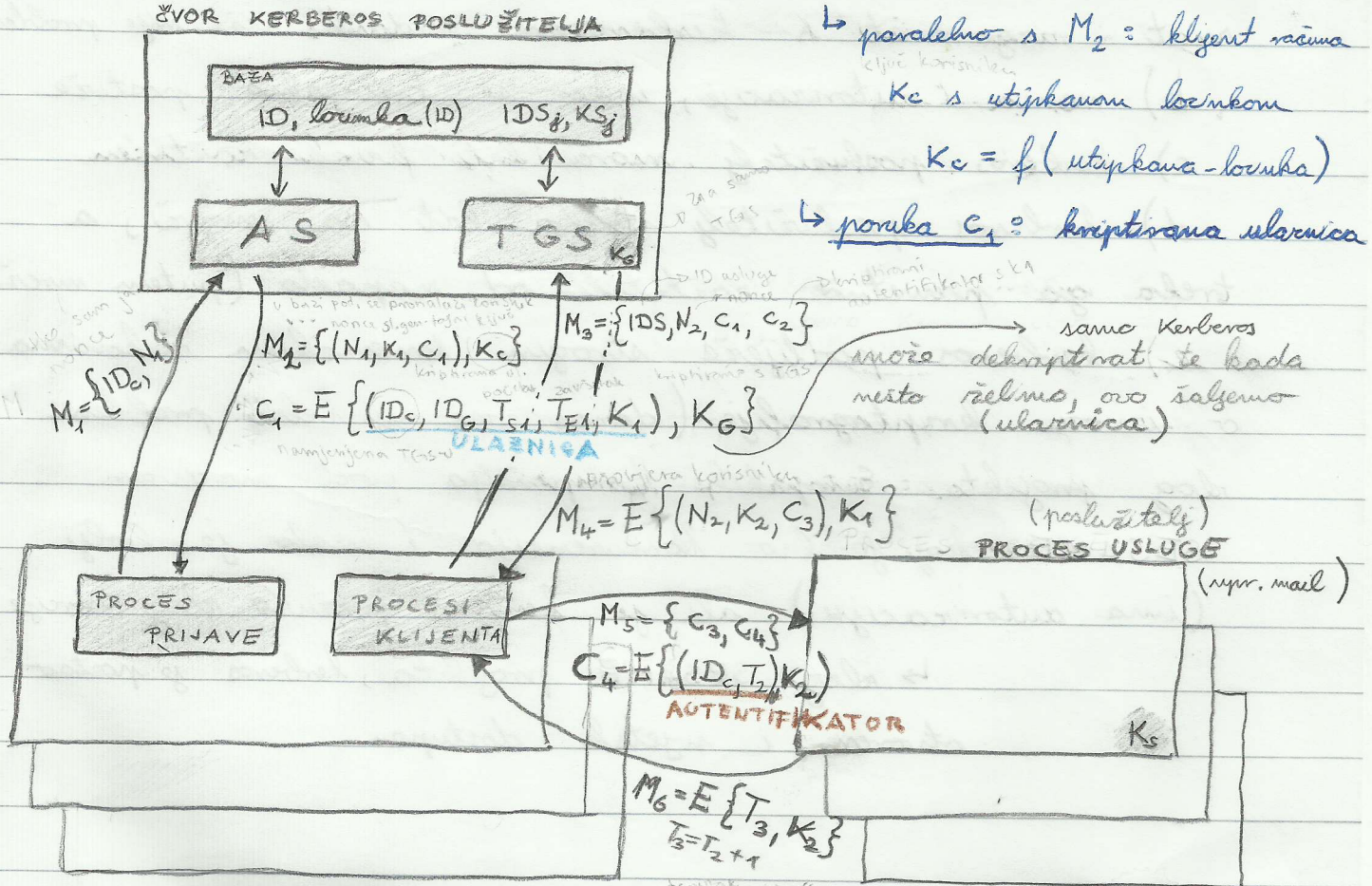
$K_c = f(\text{lozinka}(ID_c))$

↳ paralelno s M_2: klijent računa

K_c s utipkanom lozinkom

$K_c = f(\text{utipkana-lozinka})$

↳ poruka C_1: kriptirana ulaznica



↳ poruka $C_2 = E\{(ID_G, T_1), K_1\}$

↳ poruka C_3: kriptirana dozvola $C_3 = E\{(ID_C, IDS, T_{S2}, T_{E2}, K_2), K_S\}$ DOZVOLA

⇒ KERBEROS PRUŽA 3 NIVOA ZAŠTITE:

1. autentifikacija samo na početku } poruke do usluge nisu kriptirane
   ↳ najniža razina
2. sigurne poruke
   ↳ provjerava se je li poruka u intervalu
   (usluga se smije koristit samo u nekom radnom intervalu
3. kriptirane poruke (privatne poruke)
   ↳ same poruke do usluge su kriptirane

⇒ NEDOSTACI:

a) svaki program treba bit „kerberiziran" (treba znat raspravljat s kerberosom - danas više nije problem)
b) nema autorizacije, iako se trivijalno postiže
c) kerberos poslužitelj mora biti fizički zaštićen
d) kerberos poslužitelj treba biti na mreži, a treba ga pametno zaštititi od napada (putem mreže)
e) kerberos podliježe strogim Američkim zakonima o izvozu kriptografije (danas više nije veliki problem

dolaz projekta Europskih kompanija pod nazivom SESAME koji je bio konkurencija i malo je bolji (ima autorizaciju), ali je kerberos preživio dolaz konkurencije)
↳ dolaz SESAME projekta, kerberos je postao otvoren i svjetski dostupan

INFRASTRUKTURA JAVNIH KLJUČEVA

⇒ skup tehnologija, protokola, normi i usluga koj zajedno omogućuju sigurnu komunikaciju javnim ključevima na nesigurnoj mreži

⇒ PKI: „Public Key Infrastructure"

⇒ DVIJE ZADAĆE: BITNO!

(1) nedvojbeno povezivanje javnih ključeva s korisnicima i provjera jesu li on valjeći

(2) offline provjera certifikata

⇒ DIGITALNI CERTIFIKAT:

- svjedodžba koja potvrđuje da je korisnik u trenutku izdavanja certifikata posjedovao odgovarajući privatni ključ
- cilj je povezati javni ključ s korisnikom (time se izbjegne podmetanje javnog ključa)

⇒ certifikacijsko tijelo stoga generira javni i privatni ključ, nama daje privatni (na novce), a on čuva naš javni i privatni ključ (moramo tom tijelu JAKO vjerovati jer ono može s tim ključevima sve što i mi !)

⇒ drugi zahtjev se realizira tako da se povremeno download-a update svih certifikata

- stoga, trebamo biti što više online
- da smo cijelo vrijeme online, ne trebamo nam certifikat

⇒ **SADRŽAJ CERTIFIKATA:**

1. identifikator certifikata
2. nositelj (primatelj) certifikata
3. vrijeme i datum izdavanja
4. vrijeme do kada vrijedi
5. klasa certifikata
6. identitet izdavanja certifikata
7. digitalni potpis izdavatelja ←
8. javni ključ nositelja ←
9. namjena javnog ključa

⇒ norme koje propisuju sadržaj certifikata (ima ih dosta):

↳ najpoznatija je X.509

⇒ DIJELOVI PKI SUSTAVA:

1. KORISNICI
2. CERTIFIKATOR (CA)
3. REGISTRATOR (RA) - tamo gdje se ratvaruju certifikati
4. BAZA CERTIFIKATA
   - ↳ vaređi certifikati
   - ↳ opozvani certifikati (CRL - Certificate Revocation List)
5. SUSTAV ZA UPRAVLJANJE CERTIFIKATIMA
6. SUSTAV ZA REKONSTRUKCIJU IZGUBLJENIH KLJUČEVA
7. SUSTAV ZA POUZDANO VREMENSKO OŽNAČAVANJE (TSA)

⇒ KLASE CERTIFIKATA:

a) najniža klasa - nositelj je identificiran uz pomoć e-mail adrese

b) srednja razina - nositelj osobno podnio podatke

c) viša razina - nositelj je autentificiran uz pomoć isprava

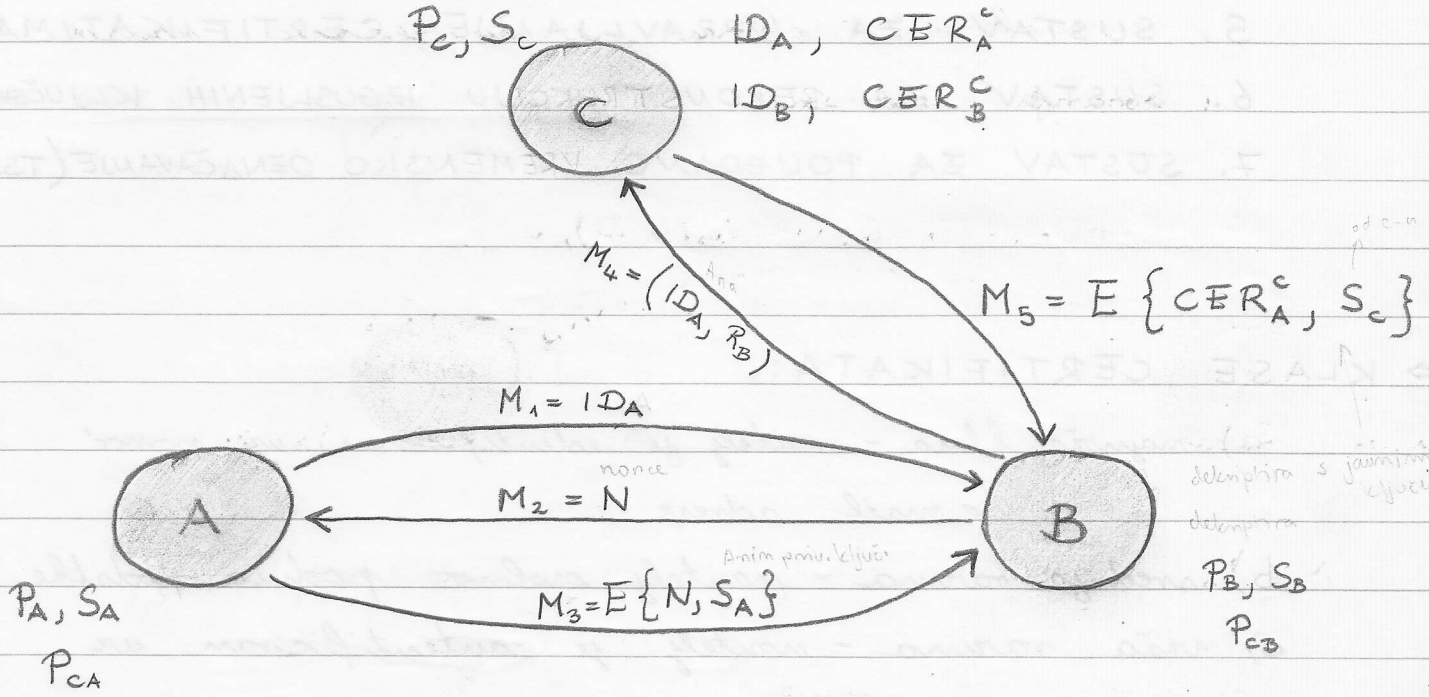
d) najviša razina - nositelj je autentificiran uz pomoć isprava i biometrije

⇒ provjera certifikata:

- lanac se mora spojit od Ane do Branka (ovo je analogno DNS poslužiteljima u mreži)

JEDNOSTRANA AUTENTIFIKACIJA U PKI SUSTAVU

⇒ cilj je isti kao i prije, samo sada imamo CERTIFIKACIJSKI CENTAR (C)



PREPORUČENI X.509 AUTENTIFIKACIJSKI PROTOKOLI

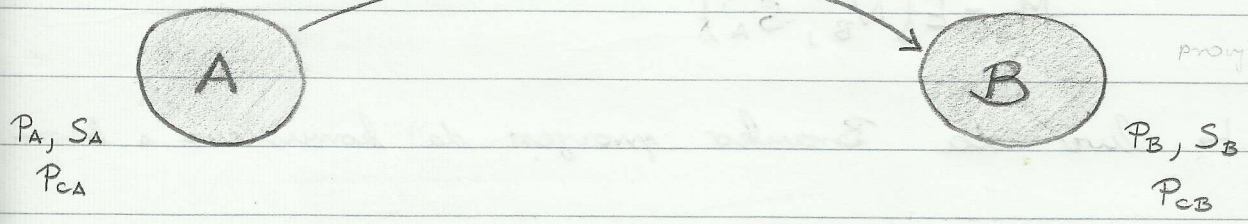
⇒ **Protokol s jednom porukom** (,,one-way protocol")

↳ autentificiraju se oba sudionika (i A i B)

↳ osigurava se integritet poruke $A \rightarrow B$

↳ vremenskom oznakom se sprječava napad ponavljanja poruke

$$M_1 = \{CER_A^{CA}, E[(ID_B, T_A, N_A, D), S_A]\}$$



N_A, T_A

$CER_B^{CB} \rightarrow P_B$

$P_B = P_{CA} \cdot (A \rightarrow B) \cdot CER_B^{CB}$

} Ana prvine provjerava Branka tako da ratvori lanac certifikata

Branko po primitku M_1 radi isto ono što je Ana na početku i time je autentificira

⇒ **Protokol s dvije poruke** („two-way protocol")

↳ dodaje se samo jedna poruka kao proširenje „one-way protocola" koja ide od Branka prema Ani:

$$M_2 = E\left[(ID_A, T_B, N_A, N_B, D), S_B\right]$$

↳ služi da Ana provjeri da komunicira s Brankom

⇒ **Protokol s tri poruke** („three-way protocol")

↳ proširenje „two-way protokola" s trećom porukom koja ide od Ane prema Branku:

$$M_3 = E(N_B, S_A)$$

↳ služi da Branko provjeri da komunicira s Anom

SIGURNOSNI ŠTIT

(engl. „ firewall ")

⇒ služi za zaštitu privatne mreže od napada iz javne mreže

⇒ <u>3 razine zaštite</u> :

a) „ packet filter " (filtrira pakete)

b) „ proxy server " (prikriva IP adresu)

c) „ full server " (služi za realizaciju FTP protokola)

VIŠEDISKOVNI ZALIHOSNI SPREMNICI (engl. RAID)

⇒ RAID: „Redundant Array of Independant Disks"

⇒ možemo dobiti dvije stvari:

a) veću brzinu

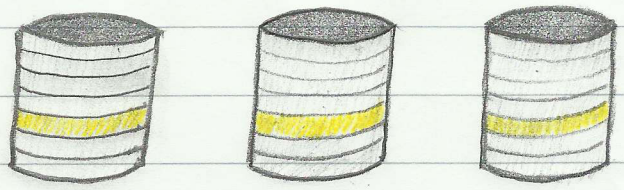
b) veću pouzdanost

⇒ MTTF: „Mean-Time-To-Failure" (očekivano vrijeme do pojave kvara)

⇒ osnovna jedinica adresiranja u RAID sustavu: „data stripe"

⇓

POJASNA ORGANIZACIJA



⇒ POJASNA ORGANIZACIJA:

a) sitno-zrnata

↳ možemo čak na bajtovima (ili bitovima) dijelit datoteku (UBRZANJE JE DOBRO?)

b) krupno-zrnata

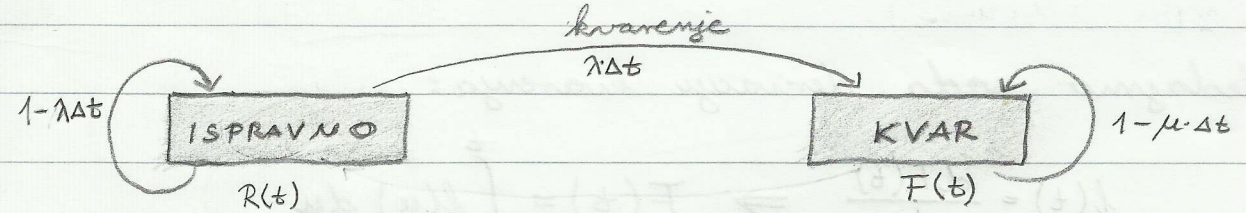
↳ ako datoteka nije veća od sektora, nemamo ubrzanje jer se dijeli po sektorima

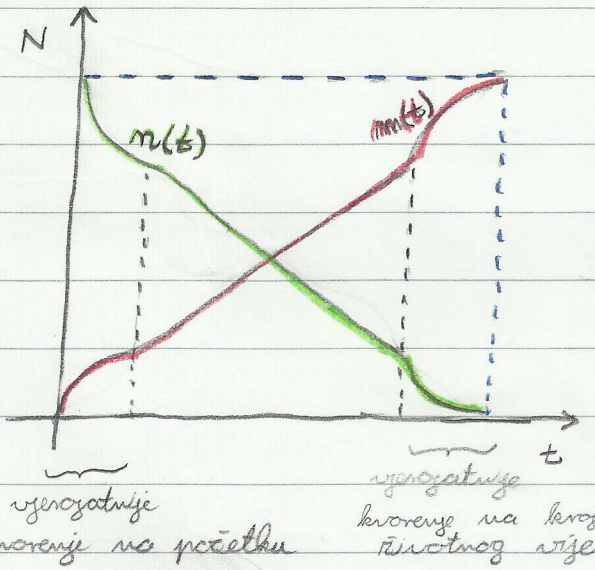
⇒ ZALIHOST: uvodimo paritetni disk koj ukolbiko dođe do kvarenja jednog diska služi za rekonstrukciju podataka pokvarenog diska, no gubi se kapacitet

↓

POVEĆAVA POUZDANOST, ALI SMANJUJE KAPACITET I BRZINU PRISTUPA

⇒ o stanjima NEPOPRAVLJIVE komponente





$n(t)$ ⇒ broj ispravnih komponenti

$m(t)$ ⇒ broj pokvarenih komponenti

$$n(t) + m(t) = N$$

⇒ POUZDANOST ⇒ vjerojatnost nepopravljive komponente da je u trenutku „t" u ispravnom stanju

$$R = \frac{m(t)}{N}$$

⇒ NEPOUZDANOST ⇒ vjerojatnost nepopravljive komponente da je u trenutku „t" u stanju kvara

$$F = \frac{m(t)}{N}$$

⇒ prihvatiti da vrijedi: $R + F = 1$

⇒ pogledajmo sada derivaciju kvarenja:

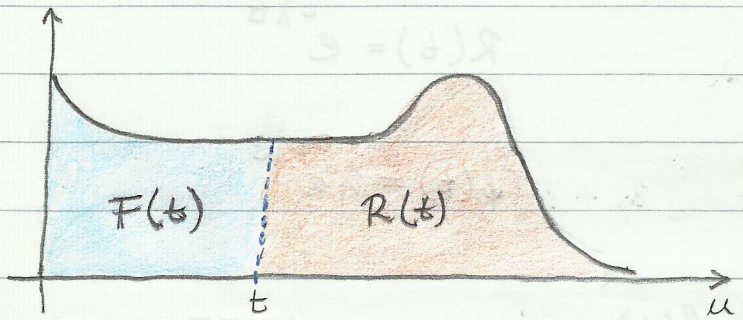
$$f(t) = \frac{dF(t)}{dt} \Rightarrow F(t) = \int_0^t f(u)\,du$$

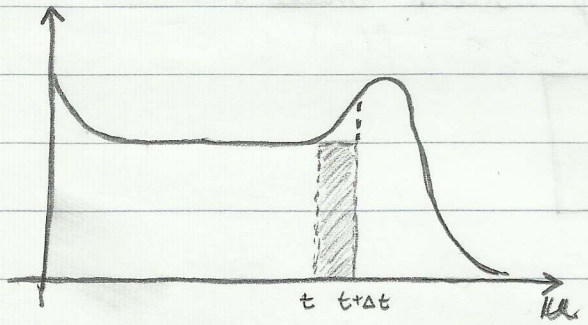
$$E(u) = MTTF = \int_0^t u\,f(u)\,du$$

KVARENJE

⇒ FAILURE RATE:

- verjetnost pojava kvara do trenutka „t" na populacijo
- oznaka $r(t)$
- gostoča kvarenja je drug naziv





$$r(t)\cdot\Delta t \approx \frac{m(t+\Delta t)-m(t)}{m(t)} \quad /:N \quad /:N$$

$$r(t)\cdot\Delta t \approx \frac{F(t+\Delta t)-F(t)}{R(t)}$$

↳ za dovolj mali Δt:

$$F(t+\Delta t)-F(t) = f(t)\cdot\Delta t$$

$$r(t)\cdot\Delta t = \frac{f(t)\cdot\Delta t}{R(t)} = \frac{f(t)\cdot\Delta t}{1-F(t)}$$

$$r(t) = \frac{f(t)}{1-F(t)} = \frac{\frac{dF(t)}{dt}}{1-F(t)} = -\frac{d}{dt}\ln\left[1-F(t)\right]$$

↳ sada izračunamo integral od $\tau(t)$:

$$\int_0^t \tau(u)\,du = -\int_0^t \frac{d}{du} \ln\left[1-F(u)\right] du = -\ln\left[1-F(u)\right] \Big|_0^t = -\ln\left[1-F(t)\right]$$

$$-\int_0^t \tau(u)\,du = \ln\left[1-F(t)\right] \Rightarrow F(t) = 1 - e^{-\int_0^t \tau(u)\,du}$$

$$F(t) = 1 - e^{-\lambda t}$$

$$R(t) = e^{-\lambda t}$$

$$f(t) = \lambda e^{-\lambda t}$$

↳ ako uvrstimo ovaj $f(t)$ u izraz za MTTF s integralom dobivamo sljedeći izraz koj treba znati:

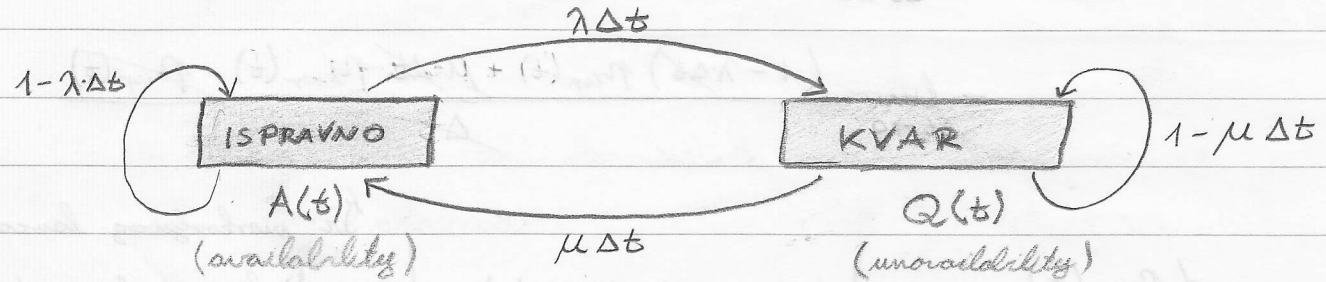
$$\boxed{MTTF = \frac{1}{\lambda}}$$

⇒ analogno izvodimo srednje vrijeme do popravka:

$$\boxed{MTTR = \frac{1}{\mu}}$$

MODEL PONAŠANJA POPRAVLJIVE KOMPONENTE S KONSTANTNIM BRZINAMA KVARENJA I POPRAVLJANJA

⇒ vratimo se na model s početka:



⇒ RASPOLOŽIVOST: vjerojatnost da je popravljiva komponenta u trenutku „t" u ispravnom stanju (A(t))

⇒ NERASPOLOŽIVOST: vjerojatnost da je popravljiva komponenta u trenutku „t" u kvarnom stanju (Q(t))

⇒ <u>uvijek vrijedi</u>:

$$A(t) \geq R(t) \qquad Q(t) \leq F(t)$$

⇒ za nepopravljive komponente i za $t=0$ popravljivih komponenti vrijedi:

$$A(t) = R(t)$$

BITNO!

⇒ odredimo vjerojatnost ispravnog stanja i sve ono isto kao i prije za nepopravljive komponente:

$$p_{isp}(t+\Delta t) = (1 - \lambda \cdot \Delta t) \cdot p_{isp}(t) + \mu \cdot \Delta t \cdot p_{kvar}(t)$$

$$\frac{d\,p_{isp}(t)}{dt} = \lim_{\Delta t \to 0} \frac{p_{isp}(t+\Delta t) - p_{isp}(t)}{\Delta t} =$$

$$= \lim_{\Delta t \to 0} \frac{(1 - \lambda \cdot \Delta t)\, p_{isp}(t) + \mu \cdot \Delta t \cdot p_{kvar}(t) - p_{isp}(t)}{\Delta t}$$

$$\frac{d\,p_{isp}(t)}{dt} = -\lambda\, p_{isp}(t) + \mu\, p_{kvar}(t)$$

$$\frac{d\,p_{kvar}(t)}{dt} = \lambda\, p_{isp}(t) - \mu\, p_{kvar}(t)$$

Iz markovljevog lanca:
↳ ako strelica ide van, onda je minus, a ako unutra, onda je plus!

⇓ zapišemo u matričnom obliku

$$\begin{bmatrix} p_{isp}(t)' \\ p_{kvar}(t)' \end{bmatrix} = \begin{bmatrix} -\lambda & \mu \\ \lambda & -\mu \end{bmatrix} \begin{bmatrix} p_{isp} \\ p_{kvar} \end{bmatrix}$$

↳ rješavamo preko svojstvenih vrijednosti:

$$|\lambda U - A| = \begin{vmatrix} \lambda + \lambda & -\mu \\ -\lambda & \lambda + \mu \end{vmatrix} = 0 \quad \Rightarrow \quad \begin{matrix} \lambda_1 = 0 \\ \lambda_2 = -(\lambda + \mu) \end{matrix}$$

$$p_{isp}(t) = C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t} = C_1 + C_2 e^{-(\lambda+\mu)t}$$

$$p_{kvar}(t) = C_3 \cdot e^{\lambda_1 t} + C_4 \cdot e^{\lambda_2 t} = C_3 + C_4 \cdot e^{-(\lambda+\mu)t}$$

↳ konstante C_1 i C_2 odredujemo iz početnih uvjeta:

• na početku su obje komponente u ispravnom stanju (rade po očekivanju)

$$\frac{d p_{isp}(0)}{dt} = -\lambda \cdot 1 + \mu \cdot 0 = -(\lambda+\mu) \cdot C_2$$

$$C_2 = \frac{\lambda}{\lambda+\mu}$$

↳ iz izvora, suma konstanti mora biti jednaka 1, pa je C_1 slijedeći:

$$C_1 = \frac{\mu}{\lambda+\mu}$$

⇒ sada imamo obje konstante, pa smo dobili IZRAZ ZA RASPOLOŽIVOST:

$$p_{isp}(t) = A(t) = \frac{\mu}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu} e^{-(\lambda+\mu)t}$$

⇒ analogno, IZRAZ ZA NERASPOLOŽIVOST:

$$p_{kvar}(t) = Q(t) = \frac{\lambda}{\lambda+\mu} - \frac{\lambda}{\lambda+\mu} e^{-(\lambda+\mu)t}$$

⇒ primjeti da vrijedi: $A(t) + Q(t) = 1$

⇒ nama su najbitniji limesi u beskonačnost:

$$\lim_{t\to\infty} Q(t) = \frac{\lambda}{\lambda+\mu} = \frac{MTTR}{MTTR+MTTF}$$

$$\lim_{t\to\infty} A(t) = \frac{\mu}{\lambda+\mu} = \frac{MTTF}{MTTR+MTTF}$$

⇒ **ZAKLJUČAK OVA DVA IZRAZA:**

- za popravljive komponente, MTTR je malen (par sati dok ne zamijenimo disk, na primjer), a MTTF je reda veličine nekoliko godina
- stoga, limes za $Q(t)$ teži u nulu (nerasploživost teži u nulu), a limes za $A(t)$ teži u jedan (sustav je uvijek raspoloživ)

⇒ **REZIME:** postupak izračuna (OPĆENITI PRISTUP)

1. nacrtaj Markovljev lanac
2. iz Markovljevog lanca, postavi sustav diferencijalnih jednadžbi
3. riješi sustav: dobiš rješenja: $p_0(t) = C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t}$
   $p_1(t) = C_3 e^{\lambda_1 t} + C_4 e^{\lambda_2 t}$
4. protumači (interpretiraj) rezultat tako da odrediš veličine $A(t)$, $Q(t)$ i krajnje:

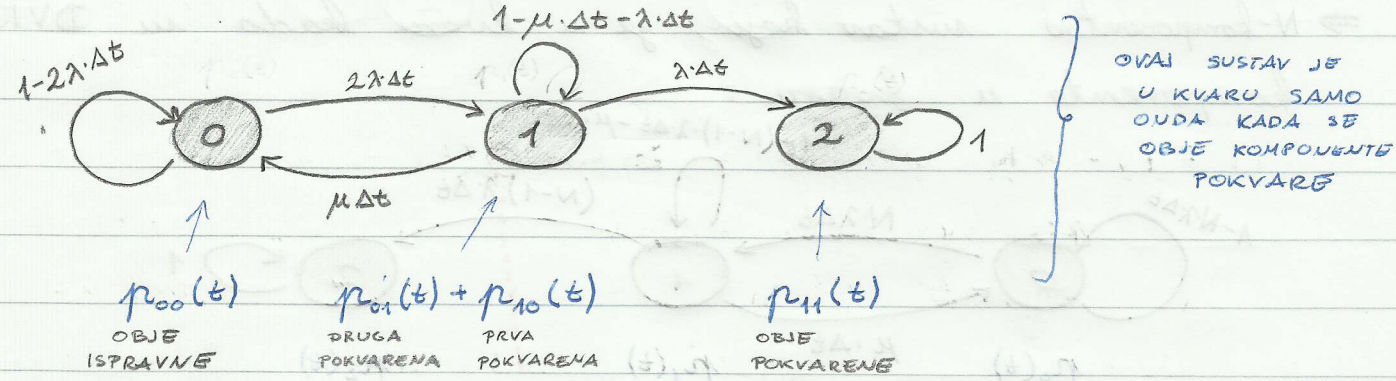
$$MTTF_S = \int_0^\infty u\, g(u)\, du$$

↓

MTTF cijelog sustava

DVOKOMPONENTNI SUSTAV

⇒ broj stanja označava broj pokvarenih komponenti:



⇒ NAPOMENA: kada na strelicama imamo izraze koj nisu pomnoženi sa Δt, onda oni označe brzina prijelaza umjesto stanja

↳ ako imamo izraze sa Δt, onda su to vjerojatnost prijelaza

⇒ za VIŠEKOMPONENTNI sustav koj je u stanju kvara kada se JEDNA komponenta pokvari, $MTTF_S$ je:

$$MTTF_S = \frac{1}{N} MTTF$$

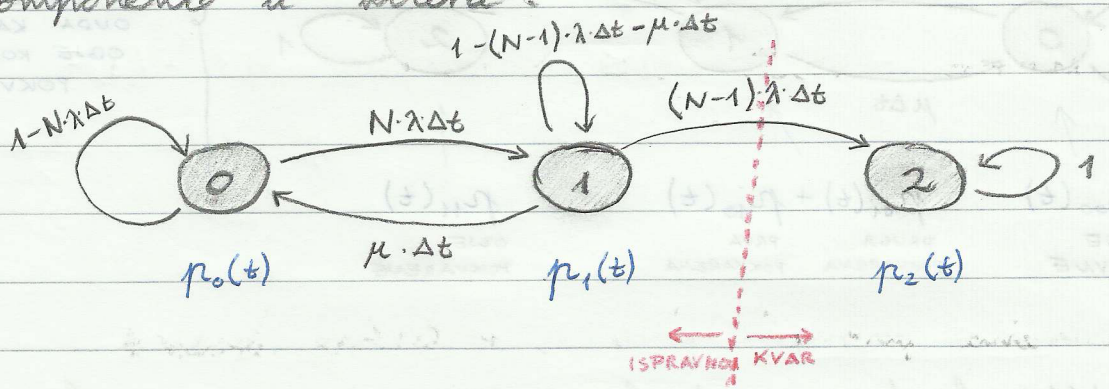
BITNO!

(ovo je neveranu uz gornju sliku jer ovaj gore ide u kvar kada se OBJE komponente pokvare)

↳ ovo je intuitivno

VIŠEKOMPONENTNI SUSTAV

⇒ N-komponentni sustav koji je u kvaru kada su DVIJE komponente u kvaru:



$$A(t) = p_0(t) + p_1(t) \qquad Q(t) = p_2(t)$$

$$q_1(t) = p_2'(t)$$

↳ napišimo sada sustav:

$$p_0'(t) = -N\lambda\, p_0(t) + \mu \cdot p_1(t)$$

$$p_1'(t) = N\lambda\, p_0(t) - [\mu + (N-1)\lambda]\, p_1(t)$$

$$p_2'(t) = (N-1)\lambda \cdot p_1(t) = q_1(t)$$

↳ ovo je jednostavno jer treća ovisi o prve dvije:

$$\begin{bmatrix} p_0' \\ p_1' \end{bmatrix} = \begin{bmatrix} -N\lambda & \mu \\ N\lambda & -\mu-(N-1)\lambda \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \end{bmatrix}$$

$$\alpha_1 \alpha_2 = \lambda^2 N$$

$$\alpha_1 + \alpha_2 = (N-1)\lambda + \mu$$

↳ iz ovoga (uz neke matematičke) dobivamo:

$$p_2'(t) = q_1(t) = (N-1)\cdot\lambda\cdot p_1(t) = \frac{N(N-1)\lambda^2}{\lambda_1-\lambda_2}\left(e^{\lambda_1 t} - e^{\lambda_2 t}\right)$$

↳ integriramo i sredimo to ispada:

$$MTTF_S = \frac{(N-1)^2}{N}\cdot\frac{1}{\lambda} + \frac{N-1}{N}\cdot\frac{\mu}{\lambda^2}$$

$\frac{1}{\lambda}$ → MTTF

$\frac{\mu}{\lambda^2}$ → $\frac{MTTF^2}{MTTR}$

↳ konačna formula koju je potrebno znati:

$$\boxed{MTTF_S = \frac{1}{N(N-1)}\cdot\frac{MTTF^2}{MTTR} + \frac{2N-1}{N(N-1)}\cdot MTTF}$$

PONAVLJANJE

Nepopravljive komponente

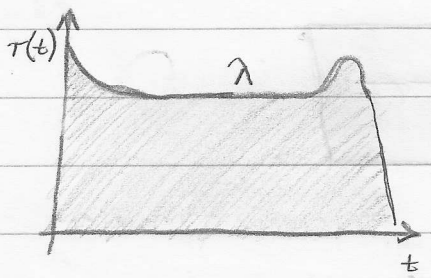
Popravljive komponente

$R(t)$ - pouzdanost $\leq$ $A(t)$ - raspoloživost

$F(t)$ - nepouzdanost $\geq$ $Q(t)$ - neraspoloživost

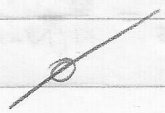
$$\underbrace{\frac{n_0(t)}{N}}_{R(t)} + \underbrace{\frac{n_1(t)}{N}}_{F(t)} = 1$$

1 komponenta



$$MTTF = \int_0^\infty u\, f(u)\, du = \frac{1}{\lambda}$$

1 komponenta



ANALOGNO

$$MTTR = \frac{1}{\mu}$$

$$\lim_{t\to\infty} R(t) = 0$$

$$\lim_{t\to\infty} A(t) = const = \frac{MTTF}{MTTF + MTTR}$$

$$\lim_{t\to\infty} F(t) = 1$$

$$\lim_{t\to\infty} Q(t) = const = \frac{MTTR}{MTTF + MTTR}$$

N komponenti

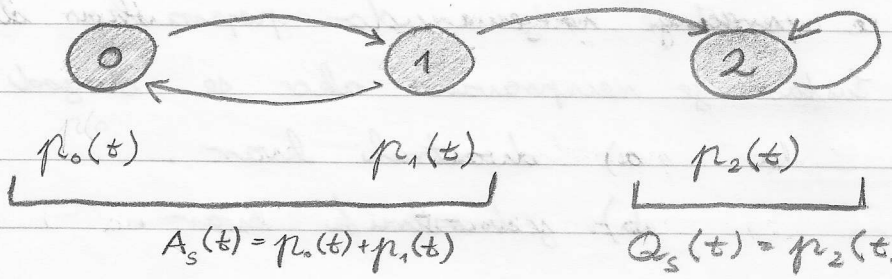
↳ za sustav N komponenti

gledamo samo popravljive komponente

N komponenti

$$MTTF_S = \int_0^\infty u\, g_\Delta(u)\, du$$

↳ nacrtamo Markovljev lanac za N komponentni sustav (s podnošenjem jednog kvara):



↳ dva slučaja:

a) sustav ispravan kada su sve komponente ispravne:

$$MTTF_S = \frac{MTTF}{N}$$

b) sustav ispravan kada su sve komponente ispravne il' jedna komponenta neispravna (ovo je gore nacrtano u Markovljevom lancu):

$$A_s(t) = p_0(t) + p_1(t)$$
$$Q_s(t) = p_2(t)$$

$$MTTF_S = \frac{1}{N(N-1)} \cdot \frac{MTTF^2}{MTTR} + \frac{2N-1}{N(N-1)} \cdot MTTF$$

⇒ NAPOMENA: „Mean-time-to-data-loss" (MTTDL) je drugi naziv za $MTTF_S$

ZAD: Sustav ima 57 istovrsnih diskova. Koliko iznosi srednje vrijeme do pojave kvara svakog diska ako je srednje vrijeme do pojave kvara u sustavu dvije godine, a srednje vrijeme do popravka diska 7 dana.

Sustav je nesprovan ako se dogodi:

a) dvostruki kvar.

b) jednostruki kvar.

$N = 57$

$MTTF_S = 2 \cdot 365$ dana

$MTTR = 7$ dana

a) $$MTTF_S = \frac{1}{N(N-1)} \cdot \frac{MTTF^2}{MTTR} + \frac{2N-1}{N(N-1)} \cdot MTTF$$

↳ kvadratna jednadžba (nije teško):

$MTTF = 10$ godina i 12 dana

b) $$MTTF_S = \frac{MTTF}{N}$$

$$MTTF = N \cdot MTTF_S \approx 14 \text{ godina}$$

NAČINI ŽALIHOSNE ORGANIZACIJE DISKOVA

RAID 0

↳ neredundantna organizacija

↳ pretpostavka: diskovi su jednaki (ako nisu, kapacitet svakog je jednak kapacitetu najmanjeg)

↳ ne gubi se kapacitet

↳ željeno ubrzanje ?! (nadamo se N puta)

RAID 1

↳ zrcaljena organizacija

↳ najčešći u praksi (nema kamo na brzinu)

↳ svaki disk ima kopiju podataka (gledamo samo paran broj diskova)

↳ **gubi se pola ukupnog kapaciteta**

↳ broj mogućih kvarenja: $\sum_{i=1}^{N-1} i$

↳ za RAID 1 sa 6 diskova, to je 15 mogućih kvarenja od kojih su 3 loša (pokvari se disk i njegova kopija)

RAID 2

↳ koristi se Hammingov kod za korekciju pogreške

↳ ne koristi se jer je suvišna kompilacija (XOR je bolji), a nismo dobili puno u odnosu na RAID 1 - trebamo 3 paritetna diska za 4 podatkovna

↳ ako bilo koja dva diska uvuče, došlo je do gubitka podataka

RAID 3

↳ također jednostruka korekcija

↳ koristi XOR, pa mora biti sitno-crnata pojasnost

↳ problem je ragušenost radnjeg (paritetnog diska)

RAID 4

↳ isto kao RAID 3, ali na krupnoj crnatosti, pa ima isti problem ragušenosti

RAID 5

↳ isto kao RAID 4, ali paritetni disk nije centralizran već su paritetni podaci raspršeni po svim diskovima

↳ dopuštava jednostruki kvar (kao i svi prije)

↳ pogledaj primjer 12.2 na vježbi!

↳ na razliku od RAID-a 2, 3, 4, ovaj RAID 5 se koristi u praksi dosta često (skoro ko RAID 1)

↳ gubi se kapacitet jednog diska (bolje od RAID 1)

RAID 6

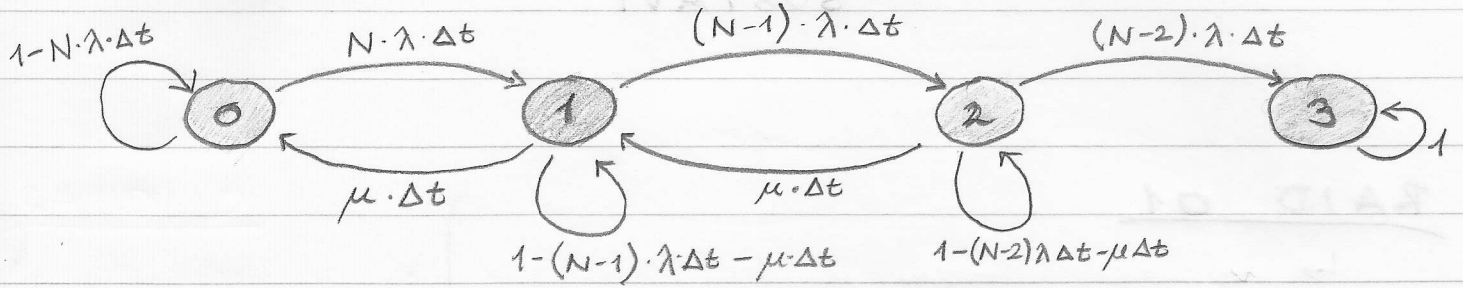
↳ jedini koji dopuštava dvostruki kvar

* koristi Reed-Solomonove kodove

↳ za 4 diska trebamo 2 redundantna diska (ali ti su također raspodijeljeni po svim diskovima)

BITNO ISPIT

PR: Nacrtaj Markovljev lanac za RAID 6



$$A(t) = p_0(t) + p_1(t) + p_2(t)$$
$$Q(t) = p_3(t)$$

PR: Odredi $MTTF_S$ ako su zadane diferencijalne jednadžbe $p_0' = \dots$, $p_1' = \dots$, $p_2' = \dots$ ISPIT

↳ treba izračunati p_3' i onda jednostavno integral:

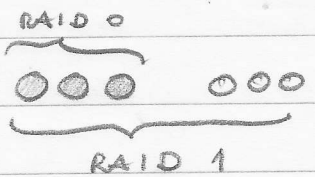
$$MTTF_S = \int_0^\infty u \cdot p_3'(u)\, du$$

(početni uvjet je $p_0(0) = 1$, $p_1(0) = p_2(0) = p_3(0) = 0$)

↳ NAPOMENA: primjeti da na gornjem Markovljevom lancu, p_0' neće ovisiti o p_3 jer nema te veze prema njoj (isto za p_1' i p_2')

VIŠERAZINSKI RAID SUSTAVI

RAID 01



nema neke pretjerane razlike

RAID 10

