Otpornost na otkaze i konsenzus algoritmi (nastavak)

Konsenzus sa proizvoljnim otkazima

- Pretpostavke:
  - Za kvarove usled pada ($2k+1$) procesa kako bi se došlo do konsenzusa u prisustvu k otkaza
  - Međutim, pretpostavka je da je kašnjenje ograničeno, tj. poruke stižu u nekom ograničenom vremenu.
    - Ali koliko je ovo ograničeno vreme?
  - Ako procesi ne rade u koračnom režimu (engl. *lock step mode*) tj. asinhroni su, tada je teško reći kolika je adekvatna latencija prilikom prijema poruka

Rezultat: FLP Impossibility

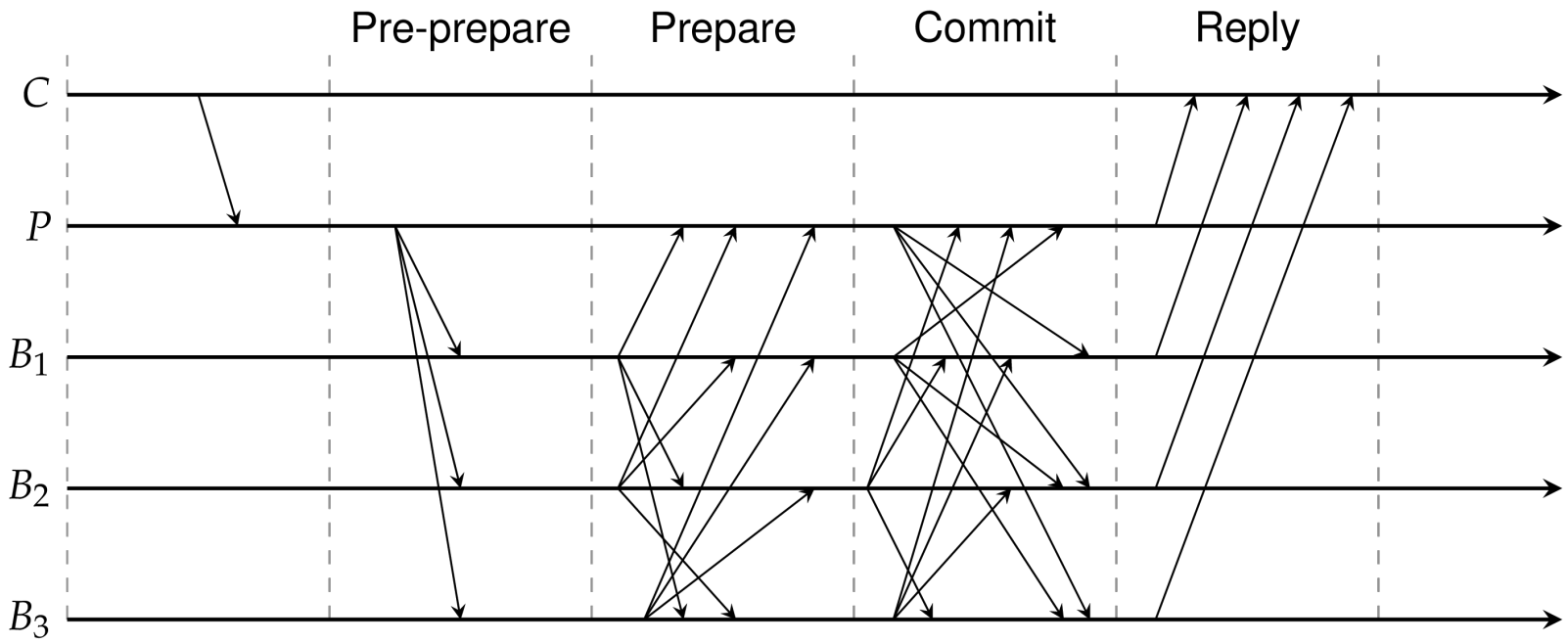
- U **asinhronom modelu**, **distribuirani konsenzus** je **nemoguć**, čak i ako **samo jedan proces** može da **otkaže**
- **F**ischer, **L**ynch, **P**aterson, "Impossibility of Distributed Consensus with One Faulty Process", J. ACM, vol. 32, no. 2 , pp. 374-382, 1985.
- Važi i za "slabi" konsenzus (samo neki proces treba da sazna, ali ne i svi)
- Važi i za samo dva stanja (0 i 1)
- Intuicija: ne možemo razlikovati otkaze od sporosti (možda nema odgovora od procesa koji ima odlučujući glas)
- Implikacija: Biramo između sigurnosti (engl. *safety*) i životnosti (engl. *liveness*) koja vodi dostupnosti (engl. *availability*)
- Kako da dobijemo oba svojstva? Neophodni su detektori kvara tj. **parcijalna sinhronost**

Algoritam praktične vizantijske tolerancije

- **Vizantijska otpornost na otkaze** bila je dugo slabo istraživana tema, delom zato što se ispostavilo da je **teško ostvariti** kombinaciju **sigurnosti** (engl. *safety*), **životnosti** (engl. *liveness*) i **praktičnih performansi**
- **Algoritam PBFT** (**Practical Byzantine Fault Tolerance**) predložili su Miguel Castro i Barbara Liskov 1999. (http://www.pmg.csail.mit.edu/papers/bft-tocs.pdf)
- PBFT ima primarni/bekap model sa $3k+1$ replikom, pretpostavke o okruženju:
  - ne pravi nikakve pretpostavke o ponašanju replika servera: pretpostavka je da će se server koji je otkazao proizvoljno ponašati
  - poruke mogu biti izgubljene, mogu kasniti i biti primljene van redosleda
  - pretpostavka je da se pošiljalac poruke može identifikovati
- Pod prethodnim pretpostavkama, dok god ne otkaže više od k servera, može se dokazati da je **algoritam praktične vizantijske tolerancije na otkaze** (PBFT) **siguran**, što znači da će klijent uvek primiti tačan odgovor
- Ako dodatno pretpostavimo **sinhronost**, što znači da su kašnjenje poruka i vreme odgovora ograničeni, takođe pruža **životnost**. Ovo praktično znači da **PBFT** pretpostavlja **parcijalno sinhroni model**, kod koga su neograničena kašnjenja izuzetak koji može, na primer, biti izazvan napadom

Primer: PBFT

- Različite faze u **algoritmu praktične vizantijske tolerancije na otkaze**. *C* je klijent, *P* je primarni proces i B_1, B_2 i B_3 su bekapi. Pretpostavka je da je B2 otkazao:

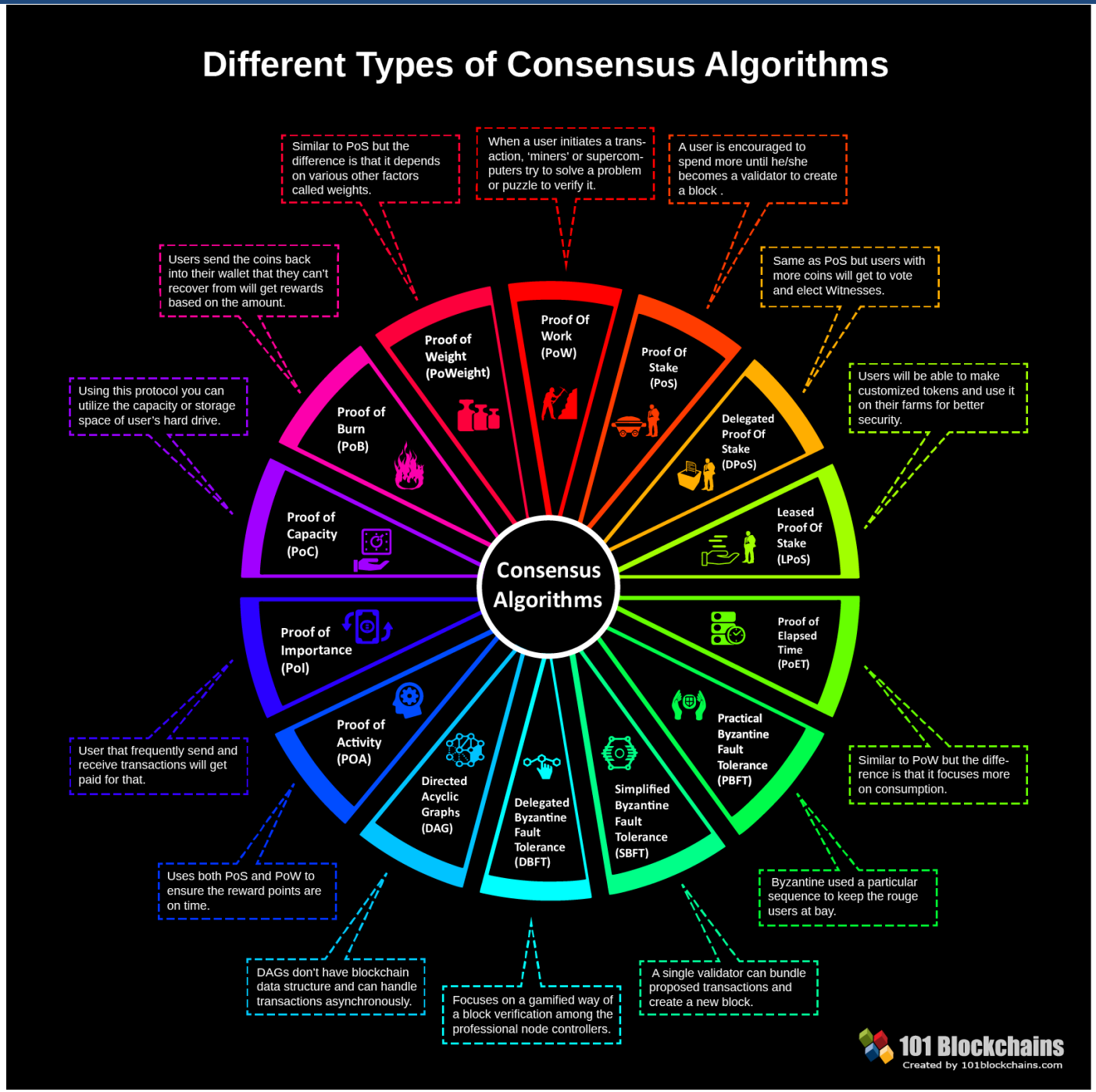


Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Primer: HotStuff

- Yin et al. (2019): "HotStuff: BFT Consensus with Linearity and Responsiveness"
  - BFT protokol za replikaciju zasnovan na lideru za **parcijalno sinhrone sisteme** – kada je mrežna komunikacija sinhrona, omogućava ispravnom lideru da vodi ka konsenzusu tempom stvarnog, a ne maksimalnog kašnjenja u mreži (svojstvo **odzivnost** – engl. *responsiveness*) i sa obimom komunikacije koji **linearno** zavisi od broja replika.
  - Protokol može poslužiti kao osnova za pravljenje **replikacionih servisa na velikoj skali**
  - **Replikacija konačnih automata** (engl. *state machine replication* – SMR) – sistem kao celina pruža repliciran servis čije stanje je preslikano na n deterministčkih replika
  - BFT SMR protokol omogućava da se, u sistemu sa n replika, ispravne replike slože oko redosleda izvršavanja komandi, uprkos delovanju f neispravnih replika. Ovim se osigurava da $n-f$ ispravnih replika izvršavaju komande u identičnom redosledu i daju isti odgovor na istu komandu

Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

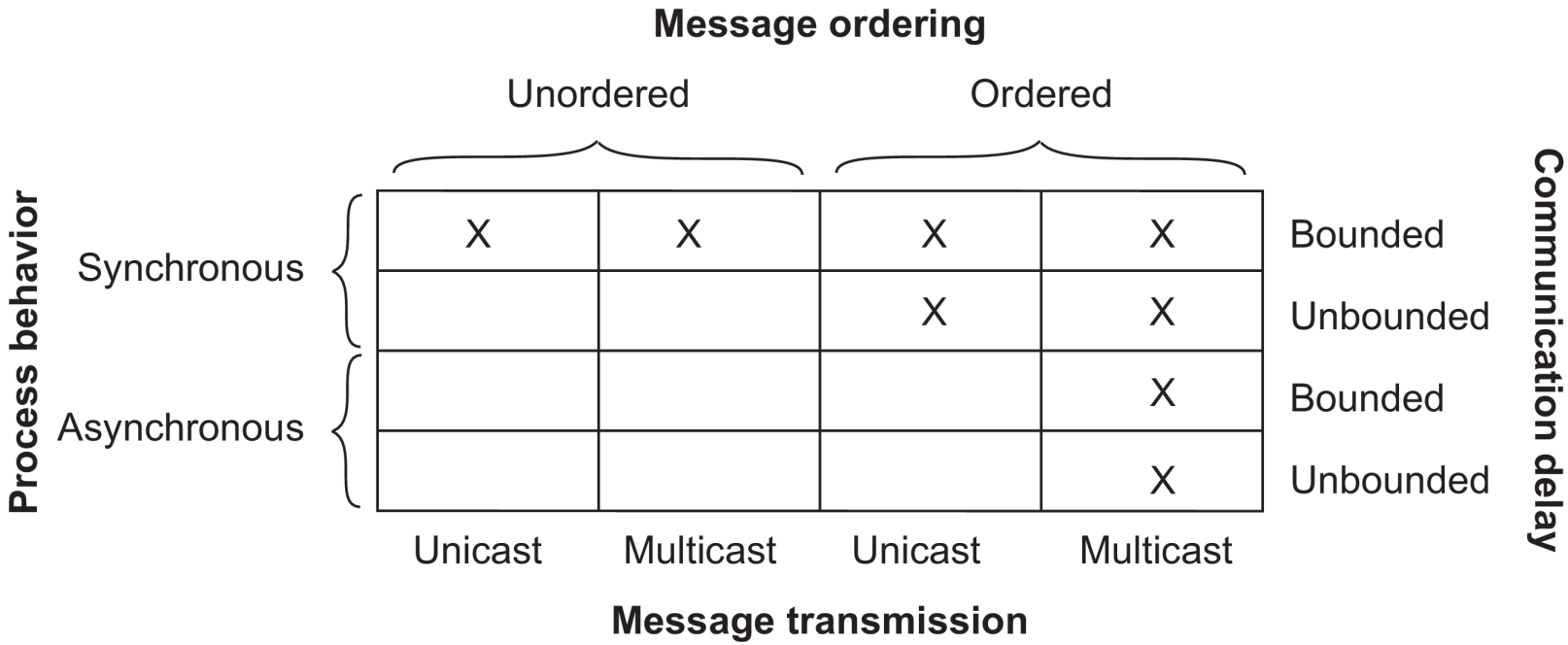

Different Types of Consensus Algorithms
Consensus Algorithms
Proof Of Work (PoW)
When a user initiates a transaction, 'miners' or supercomputers try to solve a problem or puzzle to verify it.
Proof Of Stake (PoS)
A user is encouraged to spend more until he/she becomes a validator to create a block .
Delegated Proof Of Stake (DPoS)
Same as PoS but users with more coins will get to vote and elect Witnesses.
Leased Proof Of Stake (LPoS)
Users will be able to make customized tokens and use it on their farms for better security.
Proof of Elapsed Time (PoET)
Similar to PoW but the difference is that it focuses more on consumption.
Practical Byzantine Fault Tolerance (PBFT)
Byzantine used a particular sequence to keep the rouge users at bay.
Simplified Byzantine Fault Tolerance (SBFT)
A single validator can bundle proposed transactions and create a new block.
Delegated Byzantine Fault Tolerance (DBFT)
Focuses on a gamified way of a block verification among the professional node controllers.
Directed Acyclic Graphs (DAG)
DAGs don't have blockchain data structure and can handle transactions asynchronously.
Proof of Activity (POA)
Uses both PoS and PoW to ensure the reward points are on time.
Proof of Importance (PoI)
User that frequently send and receive transactions will get paid for that.
Proof of Capacity (PoC)
Using this protocol you can utilize the capacity or storage space of user's hard drive.
Proof of Burn (PoB)
Users send the coins back into their wallet that they can't recover from will get rewards based on the amount.
Proof of Weight (PoWeight)
Similar to PoS but the difference is that it depends on various other factors called weights.
101 Blockchains
Created by 101blockchains.com

Realizacija otpornosti na otkaze

- S obzirom da su članovi **procesne grupe otporne na otkaze** (engl. *fault-tolerant process group*) vrlo čvrsto spregnuti, može doći do ozbiljnih problema po pitanju performansi ili čak do situacija u kojima realizacija otpornosti na otkaze u distribuiranom sistemu jednostavno nije moguća
- Pitanja:
  - Da li postoje ograničenja u smislu onoga što se može postići vezano za realizaciju otpornosti na otkaze?
  - Šta je neophodno kako bi se postigao konsenzus u distribuiranim sistemima?
  - Šta se dešava kada su grupe u distribuiranim sistemima razdvojene (engl. *partitioned*)?

Distribuirani konsenzus algoritmi

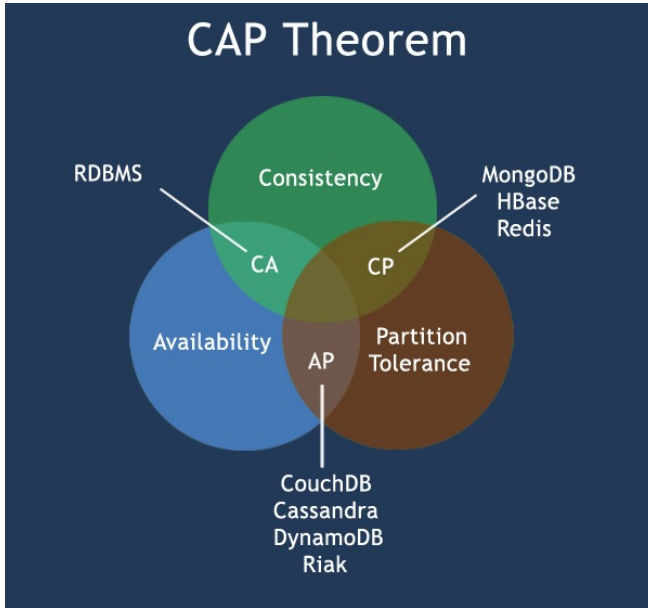
- **Formalni zahtevi** za **konsenzus u distribuiranim sistemima**:
  - **procesi** proizvode **iste izlazne vrednosti**
  - svaki **izlazna vrednost** mora biti **validna**
  - svaki **proces** mora na kraju **proizvesti izlaz**



Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

CAP teorema

- **Uslovi** pod kojima jeste i nije moguće **postići konsenzus** su **usko povezani** sa **uslovima** kada se može **ostvariti konzistentnost**
- Konzistentnost u ovom smislu znači da kada imamo **procesnu grupu** kojoj **klijent šalje zahteve** onda su **odgovori** koje dobija nazad **tačni**
- Ovde je reč o **svojstvu sigurnosti** (engl. *safety property*): ovo svojstvo tvrdi da se **ništa loše neće desiti** (engl. *nothing bad will happen*)
- Tipovi operacija koje posmatramo se izvršavaju od procesnih grupa u redosledu u kome mogu izdržati otkaz *k* članova grupe
- Dostupnost (engl. *availability*) je tipično **svojstvo životnosti** (engl. *liveness property*): **nešto će se dobro desiti na kraju** (engl. *eventually, something good will happen*)



Izvor: https://stackoverflow.com/questions/47453975/as-per-cap-theorem-which-category-does-memcached-fall-into

CAP teorema

- **CAP teorema** (Fox i Brewer 1999, dokaz Gilbert i Lynch 2002.):
  - Svaki **umreženi sistem** koji **pruža deljene podatke** može zadovoljiti **samo dva od sledeća tri svojstva**:
    - **C**: **konzistentnost** (engl. *consistency*), na osnovu koga se deljeni i replicirani podatak pojavljuje kao jedinstvena i ažurna (engl. up-to-date) kopija
    - **A**: **dostupnost** (engl. *availability*), na osnovu koga će se ažuriranja uvek na kraju izvršiti
    - **P**: **tolerantnost na particionisanje** (engl. *partitioning*), tolerancija na podeljenost procesne grupe
- Zaključak:
  - U **mreži** u kojoj su **mogući komunikacioni kvarovi**, **nemoguće** je **realizovati atomičnu čitaj/piši deljenu memoriju** koja može **garantovati odgovor na svaki zahtev**

Detekcija otkaza

- Kako možemo pouzdano detektovati da je **proces zaista pao**?
- **Opšti model**:
  - svaki proces je opremljen sa modulom za detekciju kvarova
  - proces *P* ispituje drugi proces *Q* kako bi video reakciju
  - ako *Q* reaguje: smatra se da je *Q* živ (iz tačke gledišta *P*)
  - ako *Q* ne reaguje tokom *t* vremenskih jedinica: sumnja se da je *Q* pao
- Kod sinhronih sistema, kada se sumnja na pad, onda se pad i desio
- **Detekcija otkaza** se praktično može **implementirati** na sledeći način:
  - ako *P* nije dobio otkucaj (engl. heartbeat) od *Q* tokom vremena *t*: *P* sumnja na *Q*
  - ako *Q* potom pošalje poruku koju primi *P*:
    - *P* prestane da sumnja na *Q*
    - *P* uvećava vrednost tajmauta *t*
  - ako se *Q* zaista srušio, *P* će nastaviti da sumnja na *Q*

Distribuirani komit

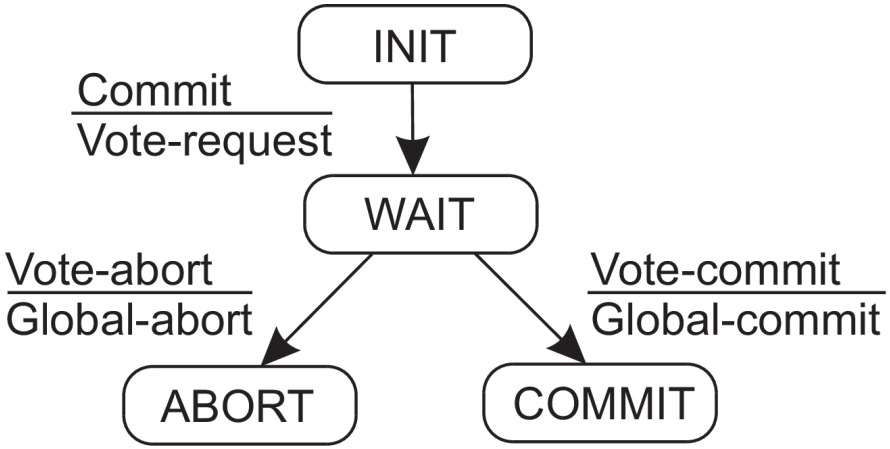
- **Distribuirani komit** podrazumeva da **operaciju izvrše ili svi članovi procesne grupe ili nijedan**
  - u slučaju **pouzdanog multikastinga**, operacija je isporučivanje poruke svim prijemnicima
  - kod **distribuiranih transakcija**, operacija može biti komit transakcije na jednom mestu koja je deo cele transakcije
- Najprostija šema je **komit protokol sa jednom fazom** (engl. *one-phase commit protocol*) kod koga koordinator kaže ostalim procesima da li da lokalno izvedu operaciju, mana: ako neki od procesa ne može da izvede operaciju, nema načina da o tome obavesti koordinatora
- Zbog toga se u praksi koriste **komit protokoli sa dve i tri faze** (engl. *two-phase and three-phase commit* – 2PC i 3PC), 2PC ne može efikasno rešavati otkaz koordinatora

2PC protokol

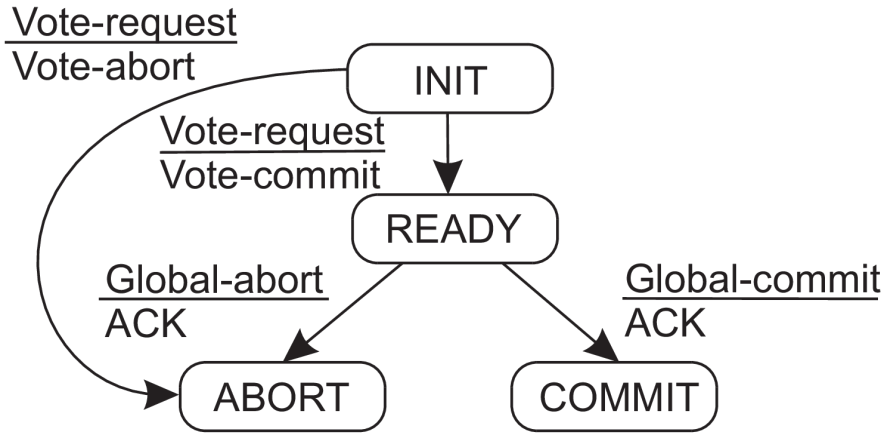
- 2PC protokol uveo Gray 1978, posmatra se **distribuirana transakcija** koja uključuje **veći broj procesa** koji se izvršavaju na **različitim mašinama**, pretpostavka je da **nema** kvarova, klijent koji je inicirao izračunavanja je **koordinator**, procesi koji treba da izvrše komit su **učesnici**
- **Protokol** se sastoji od sledeće dve faze od kojih svaka ima po dva koraka:
  - **faza 1a**: koordinator šalje *VOTE-REQUEST* učesnicima (nazive se i korak pre upisa – pre-write)
  - **faza 1b**: kada učesnik primi *VOTE-REQUEST* poruku, vraća ili *VOTE-COMMIT* ili *VOTE-ABORT* poruku koordinatoru. Ako pošalje *VOTE-ABORT*, prekida lokalno izračunavanje
  - **faza 2a**: koordinator prikuplja glasove od svih učesnika, ako su svi *VOTE-COMMIT*, šalje poruku *GLOBAL-COMMIT* svim učesnicima, u suprotnom šalje *GLOBAL-ABORT*
  - **faza 2b**: svaki učesnik čeka na *GLOBAL-COMMIT* ili *GLOBAL-ABORT* i ponaša se u skladu sa primljenom porukom

2PC protokol – konačni automati

(a) konačni automat za koordinatora



(a) konačni automat za učesnika



Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Bezbednost u distribuiranim sistemima i osnovi kriptografije

Bezbednost u distribuiranim sistemima

- **Bezbednost** je jedan od **najteže ostvarivih principa** pošto mora **prožimati ceo distribuirani sistem**. **Jedna greška** u projektovanju bezbednosnih komponenti može učiniti **sve bezbedonosne mere beskorisnim**
- Bezbednost je usko povezana sa zavisnošću (engl. *dependability*), koja uključuje dostupnost, pouzdanost, sigurnost i održivost
- Ako je potrebno da imamo poverenje u distribuirani računarski sistem, onda su veoma važne osobine i **poverljivost** (engl. *confidentiality*) i **integritet** (engl. *integrity*)
  - **poverljivost** se odnosi na svojstvo računarskog sistema koji svoje informacije otkriva samo autorizovanim stranama
  - **integritet** je svojstvo da se izmene u dobrima (engl. *asset*) sistema mogu vršiti samo na autorizovani način. Drugim rečima, nedozvoljene izmene u bezbednim računarskim sistemima mogu se detektovati i oporaviti

Polise i mehanizmi bezbednosti

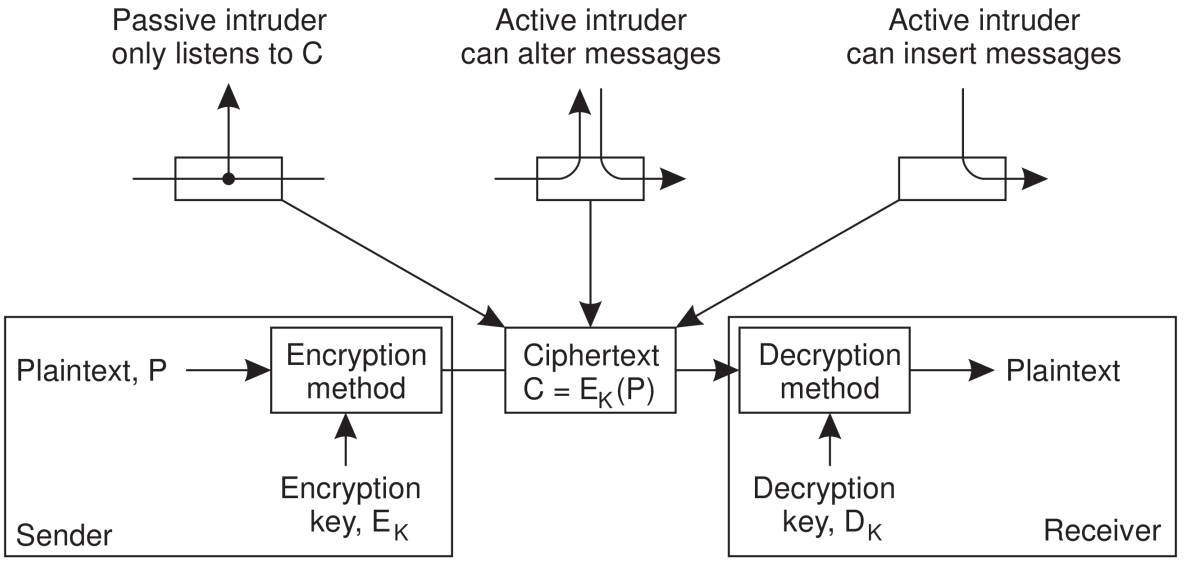
- Bezbedan računarski sistem se ne može izgraditi prostom tvrdnjom da treba biti u mogućnosti da se zaštiti od svih mogućih bezbedonosnih pretnji
- Prvo je potreban **opis bezbednosnih zahteva** koji se naziva **polisa (politika) bezbednosti** (engl. *security policy*). **Polisa bezbednosti** precizno opisuje koje **akcije entiteti u sistemu smeju**, odnosno **ne smeju preduzeti**. Entiteti uključuju korisnike, servise, podatke, mašine, itd.
- Kada se postavi polisa bezbednosti, mogu se kreirati **mehanizmi bezbednosti** (engl. *security mechanisms*) pomoću kojiih se polisa može sprovesti. Važni mehanizmi bezbednosti su:
  1. **šifrovanje** (engl. ***encryption***)
  2. **autentifikacija** (engl. ***authentication***), tj. provera identiteta
  3. **autorizacija** (engl. ***authorization***)
  4. **revizija** (engl. ***auditing***)

Mehanizmi bezbednosti

1. **Šifrovanje transformiše podatke u oblik koji napadači ne mogu razumeti**, tj. pruža način za implementaciju poverljivosti podataka. Šifrovanje dodatno omogućava proveru da li su podaci modifikovani, tj. proveru integriteta
2. **Autentifikacija** se koristi za **verifikaciju identiteta** korisnika, klijenta, servera ili drugog entiteta. U slučaju klijenta, osnovna premisa je da pre nego što servis počne da radi bilo šta u ime klijenta, servis mora znati identitet klijenta. Tipično, korisnici se autentifikuju putem lozinki (engl. *passwords*)
3. **Autorizacija** – nakon autentifikacije klijenta, neophodno je proveriti **da li je klijent autorizovan da izvrši traženu akciju**. Npr. zavisno od toga ko pristupa bazi podataka, dozvola može biti dana za čitanje zapisa, modifikaciju određenih polja u zapisu ili dodavanje ili brisanje celih zapisa
4. **Revizija** – alati za reviziju koriste se za **praćenje koji klijenti pristupaju čemu i na koji način**. Iako sama revizija ne pruža zaštitu od bezbedonosnih pretnji, revizioni logovi mogu biti veoma korisni za analizu bezbedonosnih propusta i preduzimanje mera protiv uljeza. Zbog toga se napadači trude da ne ostave nikakve tragove koji bi doveli do otkrivanja njihovog identiteta. U ovom smislu, postojanje logova čini napade na računarske sisteme riskantnijim

Kriptografija

- **Kriptografija** je **umetnost** i **nauka čuvanja bezbednosti poruka** (Schneier)
- **Pošiljalac** (engl. *sender*) *S* želi da pošalje **poruku** (engl. *message*) *m*, **primalac** (engl. *receiver*) *R*. Kako bi zaštitio poruku od bezbedonosnih pretnji, *S* prvo **šifruje** *m* u nerazumljivu poruku *m′* i potom šalje m′ primaocu R. R mora **dešifrovati** primljenu poruku u originalnu formu *m*. Šifrovanje i dešifrovanje se postižu primenom kriptografskih metoda parametrizovanih ključevima. Originalni oblik poruke naziva se **otvoren tekst** (engl. ***plaintext***), šifrovana forma je **šifrat** (engl. ***ciphertext***)



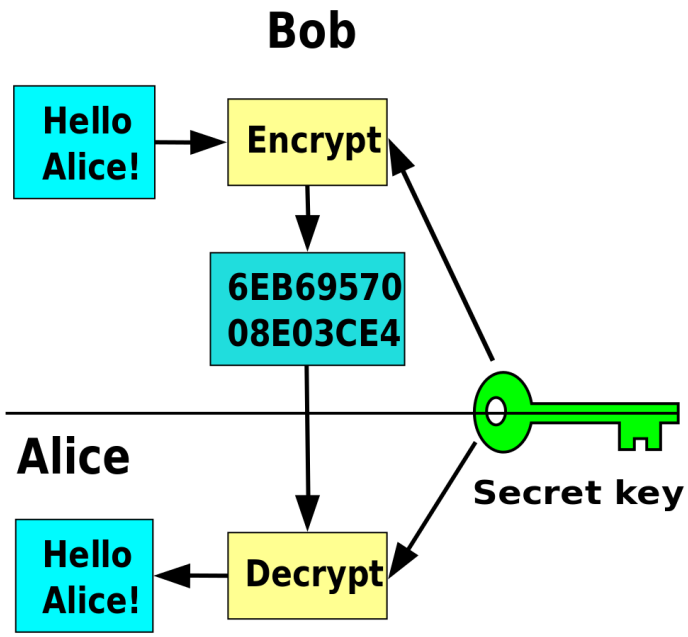
Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Simetrični i asimetrični kriptosistemi

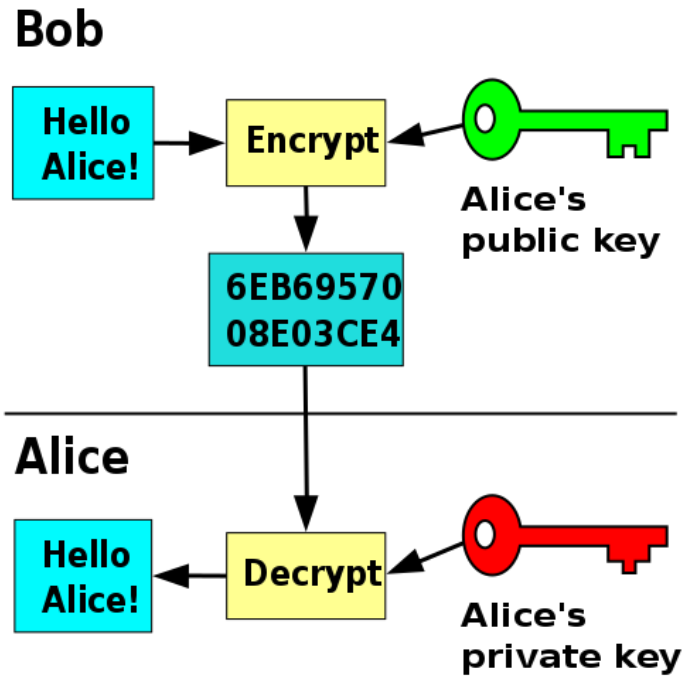
- U **simetričnom kriptosistemu**, isti ključ se koristi i za šifrovanje i za dešifrovanje poruke: $P = D_k(E_k(P))$
- Simetrični kriptosistemi su poznati i kao **sistemi sa tajnim** (engl. ***secret-key***) ili **deljenim ključem** (engl. ***shared-key***), zato što se zahteva da pošiljalac i primalac dele isti ključ, da bi se osiguralo da zaštita radi, **deljeni ključ** se mora održati **tajnim**, pa komunikacija ide isključivo kroz **bezbedne kanale** (engl. *secure channels*). Primeri: DES, AES
- U **asimetričnom kriptosistemu**, ključevi za šifrovanje i dešifrovanje se razlikuju, ali zajedno čine **jedinstveni par**. Drugim rečima, postoji poseban **ključ za šifrovanje** K_E i **dešifrovanje** K_D, tako da je $P = D_{K_D}(E_{K_E}(P))$
- Jedan od ključeva u asimetričnom kriptosistemu se čuva kao privatan, dok se drugi objavljuje **javno**. S toga se asimetrični kriptosistemi nazivaju i **sistemi sa javnim ključem** (engl. ***public-key systems***). Primeri: Diffie-Hellman, RSA, ECC, digitalni potpisi

Simetrični i asimetrični kriptosistemi

(a) **simetrični kriptosistem**



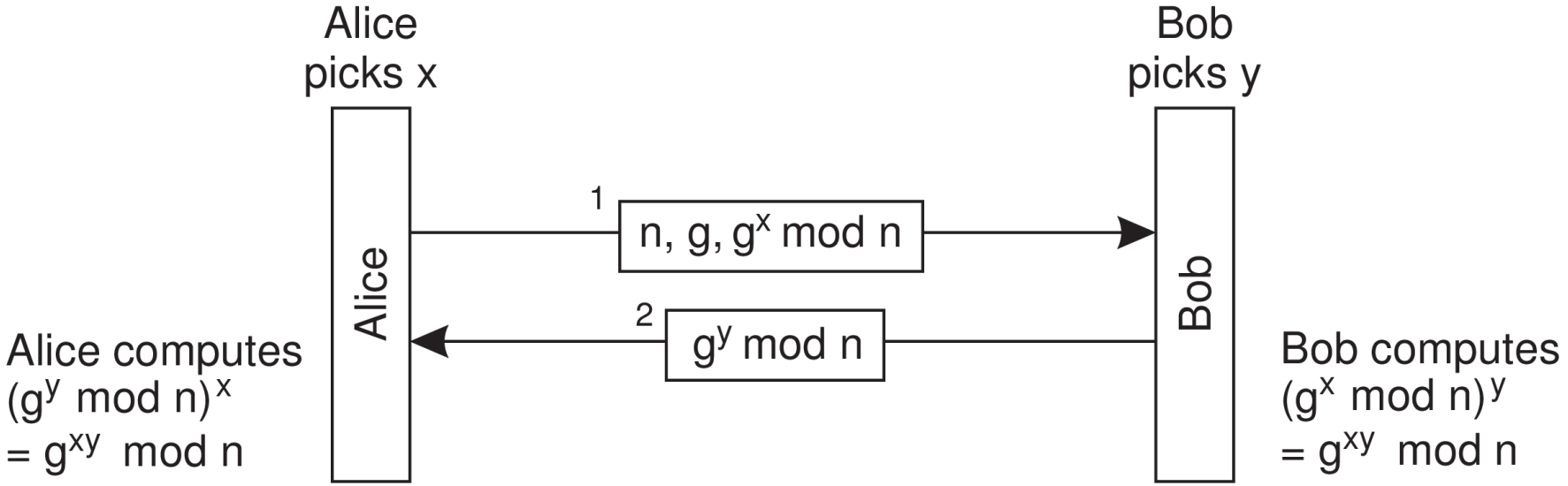
(b) **asimetrični kriptosistem**



Izvor: https://en.wikipedia.org/wiki/Cryptography

Diffie-Hellman razmena ključeva

- Princip **Diffie-Hellman razmene ključeva**:
  - Alisa i Bob se dogovore o dva velika broja *n* i *g*, koji su oba javno poznati
  - Alisa bira slučajni broj *x* koji čuva kao tajnu (privatni ključ), Bob bira *y*
  - Nakon koraka 1 i 2 sa slike, isključivo Alisa i Bob će imati tajni ključ g^{xy} mod *n*



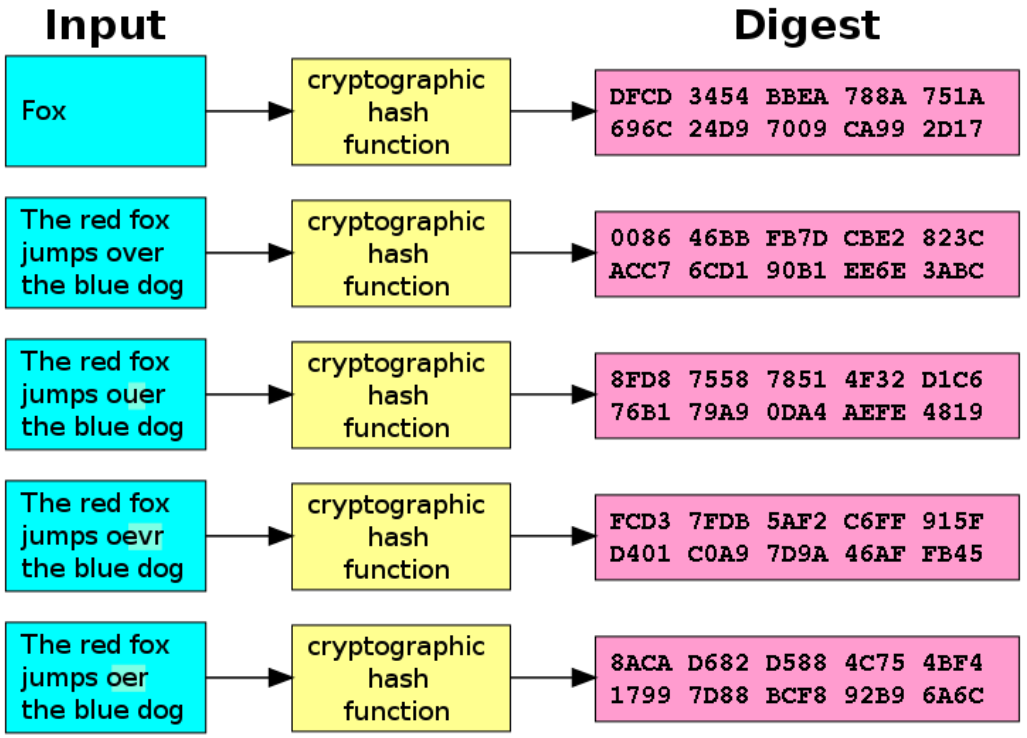
Izvor: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017

Kriptografske heš funkcije

- **Heš funkcija** *H* uzima kao ulaz poruku *m* proizvoljne dužine i proizvodi kao izlaz alfanumerički string *h* fiksne dužine: $h = H(m)$
- Izlazni string se naziva **vrednost heša**, **sažetak poruke** (engl. *message digest*), **sažetak** ili **kontrolna suma** (engl. *checksum*)
- **Osobine idealne kriptografske heš funkcije**:
  - **jednosmernost** (engl. *one-way function*) označava da je neizvodljivo sa stanovišta složenosti izračunavanja (engl. *computationally infeasible*) pronaći ulaz *m* koji odgovara poznatom izlazu *h*. S druge strane, računanje *h* na osnovu *m* je lako. Schneier naziva jednosmerne heš funkcije „radnim konjima“ savremene kriptografije, primeri kriptografskih heš algoritama SHA-1/2/3, MD5
  - **slaba koliziona otpornost** (engl. *weak collision resistance*) označava da je, za dati ulaz *m* i njemu pridruženi izlaz $h = H(m)$, neizvodljivo sa stanovišta složenosti izračunavanja pronaći neki drugi ulaz $m' \neq m$ tako da je $H(m) = H(m')$
  - **jaka koliziona otpornost** (engl. *strong collision resistance*) označava da, kada je poznato samo *H*, nije izvodljivo sa stanovišta složenosti izračunavanja da se pronađu bilo koja dva različita ulaza *m* i *m'*, tako da je $H(m) = H(m')$

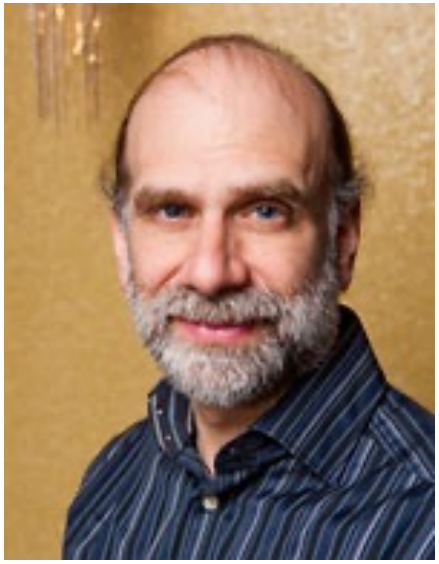
Primer: kriptografske heš funkcije

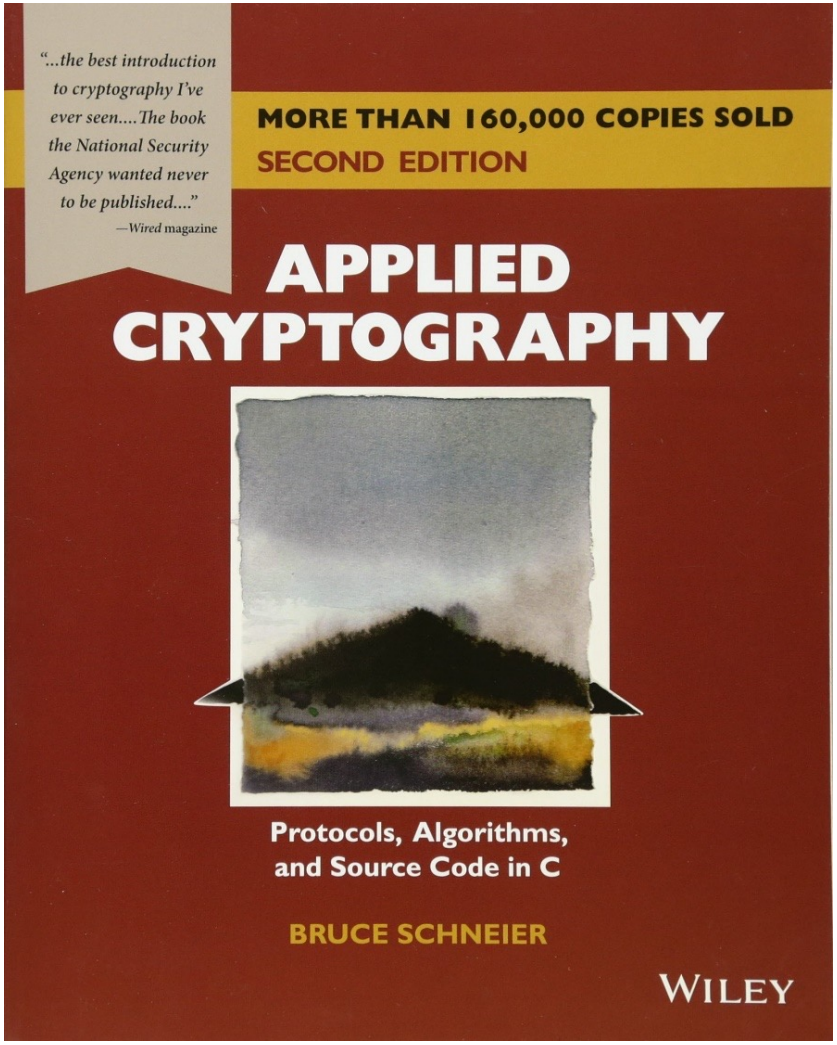
- Čak i **male promene u ulaznim vrednostima** (u datom primeru kod reči "over") dovode do **drastičnih promena u rezultujućim izlazima**, tzv. **efekat lavine** (engl. *avalanche effect*):

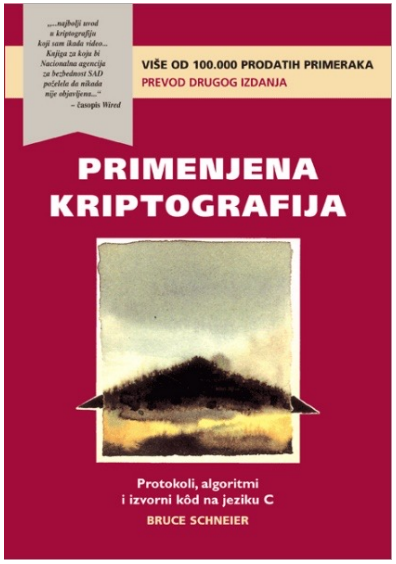


Izvor: https://simple.wikipedia.org/wiki/Cryptographic_hash_function

Literatura







Izvor: https://www.schneier.com/books/applied_cryptography/