UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika - raziskovalna smer

Tadej Novak

Uporaba Berlekamp-Masseyevega algoritma v kriptografiji in teoriji kodiranja

Magistrsko delo

Ljubljana, 2003

Kazalo

PROGRAM MAGISTRSKEGA DELA

Uporaba Berlekamp-Masseyevega algoritma v kriptografiji in teoriji kodiranja

Delo naj predstavi Berlekamp-Masseyev (BM) algoritem in njegovo uporabo v kriptografiji ter teoriji kodiranja (glej [Ber68] in [Mas69]). Poudarek naj bo na kriptoanalizi tekočih šifer ter BCH kodah za odpravljanje napak (kot npr. Reed-Salomonovih kodah). Za konec pa naj predstavi še dokaz novega determinističnega polinomskega algoritma za testiranje praštevilskosti.

LFSR-generator psevdonaključnih zaporedij je izredno učinkovit in enostaven za implementacijo, poleg tega pa ima zelo dobre statistične lastnosti (glej npr. Praprotnik [Pra01]). Vendar pa se da z BM-algoritmom za dano zaporedje učinkovito skonstruirati najkrajši LFSR-generator, ki generira to zaporedje. Zato LFSR-generator, kot samostojen generator, ni več uporaben v kriptografske namene. Kljub temu je z uvedbo nelinearizacije še vedno osnova večine uporabnih generatorjev. Če pri izhodnem zaporedju novega generatorja spremenimo nekaj določenih bitov, lahko zaporedje postane linearno. Odprto ostaja vprašanje, ali to velja pri vsaki osnovni nelinearizaciji. Z BM-algoritmom danes preverimo linearno zahtevnost vsakega zaporedja.

Izdela naj se tudi podroben opis BCH kod za odpravljanje napak, ki so primer cikličnih linearnih kod. Do leta 1968 so bile zanimive zgolj s teoretičnega vidika, saj je dekodiranje potekalo preko računsko prezahtevnega reševanja linearnega sistema enačb. Leta 1968 pa je Berlekamp [Ber68] sestavil hiter iterativen algoritem za reševanje ključne enačbe in s tem uvrstil BCH kode med ene najbolj uporabnih. Danes se za zapisovanje podatkov na zgoščenkah (CD) uporablja Reed-Solomonove kode.

[Ber68] E.R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York etc, 1968.

[BP94] D. Bini, V. Pan, *Polynomial and Matrix Computation, Fundamental Algorithms, Vol. 1*, Birkhauser, Boston-Basel-Berlin, 1994.

[HHL00] D. R. Hankerson, D. G. Hoffman, D. A. Leonard, C. C. Linder, *Coding Theory and Cryptography : The Essentials; 2 ed.*, M. Dekker, New York, Basel, 2000.

[Mas69] J. L. Massey, *Shift-Register Synthesis and BCH Decoding*, IEEE Transactions on Information Theory, **Vol. IT 15, No. 1** (January 1969), 122–127.

[Pra01] M. Praprotnik, *Kriptoanaliza urno-kontroliranega pomičnega registra*, diplomsko delo, 2001.

[Sti95] D. R. Stinson, *Cryptography, Theory and Practice*, CRC Press, 1995.

Mentor: prof. dr. Aleksandar Jurišić

Ljubljana, 5. marec 2003

Povzetek. Delo opisuje področji, za potrebe katerih je bil razvit Berlekamp-Masseyev algoritem. V prvih dveh poglavjih so predstavljene kode za odpravljanje napak. Največji poudarek je na dekodiranju BCH kod, kjer s pomočjo Berlekampovega algoritma rešimo takoimenovano ključno enačbo, ter na dekodiranju v sezname za Reed-Solomonove kode, kjer z Guruswami-Sudanovim algoritmom dosežemo večjo učinkovitost odpravljanja napak. V naslednjih treh poglavjih so obravnavane tokovne šifre, to je del področja simetrične kriptografije. Podrobno so opisane statistične lastnosti m-zaporedij, ki so poseben primer LFSR zaporedij. Te lastnosti so takoimenovani Golombovi postulati naključnosti. Podrobno je predstavljen tudi Berlekamp-Masseyev algoritem za sintezo LFSR. Definirani so potrebni pogoji za kriptografsko varnost psevdonaključnih zaporedij, kar je pravzaprav bistveni del kriptoanalize asinhronih tokovnih šifer, ter napadi na tokovne šifre in statistični testi. V zadnjem poglavju pa je predstavljen prvi determinističen algoritem polinomske časovne zahtevnosti za problem praštevilo, ki napoveduje umik verjetnostnih algoritmov za problem praštevilo iz algoritmov za iskanje naključnih praštevil, ki se uporabljajo v mnogih kriptografskih algoritmih.

Abstract. This dissertation describes two fields of coding theory where we can use Berlekamp-Massey algorithm. Error correcting codes are presented in the first and second chapter. The most important parts are presentations of the hardest part of decoding procedure for BCH codes, i.e. solving of key equation for which Berlekamp algorithm was developed, and Guruswami-Sudan list decoding algorithm for Reed-Solomon codes. With this algorithm the capacity of error-correction is improved. In the next three chapters the paper is dealing with stream ciphers, the part of symmetric cryptography. Properties of m-sequences, which are an example of LFSR sequences, are described in details. This are so called Golombs Randomness Postulates. Berlekamp-Massey algorithm for LFSR synthesis is fully described. There are also defined necessary conditions for cryptographic secure use of pseudo-random sequences, which is the most important part of cryptanalysis of asynchronous stream ciphers. In the fifth chapter is the list of some general possible attacks on stream ciphers and a collection of many statistical tests. In the last chapter the first deterministic algorithm for decision problem primes is presented. One can suspect the removing of randomized algorithms for problem primes from procedures for finding random prime numbers used in many cryptographic algorithms.

Math. Subj. Class. (2000): 11-04, 11A41, 11A51, 11T71, 94A55, 94A60, 94B05, 94B15, 94B20, 94B35, 94B70 94A55, 94A60, 94B05, 94B15, 94B20, 94B35, 94B65, 94B70

Ključne besede: kode za odpravljanje napak, ciklične kode, BCH kode, Reed-Solomonove kode, dekodiranje v sezname, problem rekonstrukcije polinoma, Berlekampov algoritem, ključna enačba, kriptografija, brezpogojna varnost, razbijanje kriptosistemov, napadi, tokovni tajnopisi/šifre, psevdonaključna zaporedja, PNZ, linearni povratni pomični registri, LFSR, m-zaporedja, statistični testi, Golombovi postulati naključnosti, Berlekamp-Masseyev algoritem, kriptoanaliza, NESSIE, problem praštevilo, AKS algoritem.

Key words: error correcting codes, cyclic codes, BCH codes, Reed-Solomon codes, list decoding, polynomial reconstruction problem, Berlekamp algorithm, key equation, cryptography, unconditional security, attacks, stream ciphers, pseudo-random sequences, PRS, linear feedback shift registers, LFSR, m-sequences, statistical tests, Golomb randomness postulates, Berlekamp-Massey algorithm, cryptanalysis, NESSIE, problem primes, AKS algorithm.

Zahvalil bi se ženi Barbari, ki me je bodrila pri raziskovanju in vnetem pisanju naloge, prof. dr. Aleksandru Jurišiću za vse nasvete in popravke pri pisanju tega dela, podjetju Cifra d.o.o. za spoznavanje kriptografije ter možnost testiranja algoritmov v praksi ter centru CACR (Centre for Applied Cryptographic Research) na univerzi v Waterlooju, Kanada, kjer sem preživel nepozabne raziskovalne dneve, še posebej pa prof. Guang Gong in prof. Alfredu J. Menezesu za vso pomoč. Da sem lahko brezskrbno delal, pa sem hvaležen tudi svojim in Barbarinim staršem ter vsem sodelavcem na IMFM in FMF.

Uvod

Če hočemo najti najprestižnejši element, ki na skrivnosten način močno povezuje vsa področja teorije kodiranja, je to prav gotovo Berlekamp-Masseyev algoritem. Teorija kodiranja je sestavljena iz treh področij, ki jih bomo na kratko opisali.

Kode za odpravljanje napak nam omogočajo odpravljanje napak povzročenih pri prenosu ali shranjevanju podatkov. Uporabljamo jih na področjih telekomunikacij, pri čemer je velik del razvoja narekoval razvoj odkrivanja vesolja in pošiljanja podatkov na zemljo. Razvite so bile posebne kode z veliko redundance in s čimvečjo stopnjo odkrivanja napak pri dani dolžini kodne besede.

Tako je misija Mariner leta 1972 pošiljala na Zemljo slike z Marsa s pomočjo **Reed-Mullerjeve kode** [93, Chapter 4.2]. Slike so bile črno-bele s $64 = 2^6$ stopnjami sivine. Posamezna točka na sliki je torej predstavljala 6 bitov informacij. Pred prenosom ji je bilo dodanih 26 bitov redundance in pri prejetju teh 32 bitov informacije o točki smo lahko pravilno določili barvo točke, če se pri prenosu ni pojavilo več kot 7 napak.

Ko je Voyager leta 1979 pošiljal z Jupitra barvne slike z $4096 = 2^{12}$ barvnimi odtenki, je uporabljal Golayevo kodo [93, Chapter 4.3]. Kodna beseda vsebuje 24 bitov, torej 12 bitov redundance, sposobna pa je odpraviti največ 3 napake.

Pri shranjevanju podatkov pa je razlog za nepogrešljivost uporabe kod v tem, da je mnogo ceneje za uporabo medijev implementirati kode za odpravljanje napak, kot pa narediti 100% zanesljiv vir. Tako jih uporabljamo pri zgoščenkah, trdih diskih, DVD-jih in drugje.

Kompresija podatkov je področje na katerem iščemo algoritme, s katerimi se da dane informacije shraniti tako, da zavzamejo čim manj prostora na podatkovnem mediju. Pri tem upoštevamo, da se pri določenih zaporedjih elementov, ponavadi bitov, določeni vzorci pojavljajo pogosteje, kot bi se pri neodvisnih metih poštenega kovanca. Določeni statistični testi za testiranje naključnosti temeljijo na tem, da se naključnega zaporedja v povprečju ne da kompresirati.

Kriptografija je seveda področje, ki se je močno razvilo v drugi polovici prejšnjega stoletja. Temu je botrovalo najprej obdobje hladne vojne, v zadnjem desetletju pa razcvet interneta, predvsem na področju e-trgovine in e-bančništva. Vzpostavitev varne komunikacije po javnem kanalu sta s svojo idejo in primerom algoritma prva omogočila Diffie in Hellman [21]. Ko je komunikacija vzpostavljena pa uporabljamo običajne bločne ali pa tokovne šifre, prve so ponavadi varnejše, druge pa mnogo hitrejše.

Razviti so tudi drugi algoritmi, na primer digitalni podpisi, ki omogočajo, da lahko vsakdo preveri identiteto podpisovalca ter preprečujejo ponarejanje podpisov. S tem lastnik podpisa ne more zanikati. Več o kriptografiji, predvsem pa o tokovnih šifrah, pri katerih igra Berlekamp-Masseyev [60] algoritem veliko vlogo, bomo povedali v 3. poglavju.

Pri uporabi zgoščenk se uporabljajo Reed-Solomonove kode, ki so poseben primer BCH kod, katere bomo podrobneje predstavili v 2. poglavju. Najtežji in časovno najzehtevnejši del pri uporabi kod je dekodiranje, postopek s katerim odpravimo napake, ki se pojavijo pri prenosu. Dekodirni postopek za BCH kode temelji na reševanju tako imenovane ključne enačbe. Berlekamp [7] je leta 1968 predstavil učinkovit algoritem za reševanje le te in BCH kode so sedaj zaradi hitrosti dekodiranja in možnosti najrazličnejših konfiguracij med najbolj razširjenimi. Seveda pa so zaradi nizke stopnje vsebnosti informacije v kodnih besedah neprimerne na primer za prenos slik iz vesolja.

Berlekampovo idejo reševanja je nato učinkovito uporabil Massey [60], ki je predstavil za malenkost drugačen algoritem za LFSR sintezo. To je sedaj tako imenovan Berlekamp-Masseyev (BM) algoritem, ki nam za dano zaporedje nad poljubnim obsegom poišče najkrajšo linearno rekurzivno enačbo, s pomočjo katere lahko dobimo to zaporedje. Če je dolžina zaporedja enaka n, dolžina dobljene enačbe pa m, potem je časovna zahtevnost algoritma enaka $O(nm)$. Število m imenujemo linearna zahtevnost zaporedja in če je $m = 100$ in želimo preveriti linearno zahtevnost zaporedja v 10 MB veliki datoteki (to je približno 100 M bitov), potem je zahtevnost algoritma približno 10 giga binarnih operacij. Na 1 GHz računalniku se to delo izvede v 10 sekundah. Če pa bi uporabljali katerokoli metodo hitrosti $O(n^2)$, bi bilo delo opravljeno v 100 M sekundah, kar je več kot 3 leta in to je dovolj zgovoren rezultat o učinkovitosti algoritma. Zahteva po visoki linearni zahtevnosti je tako poleg primerne avtokorelacije postala ena najpomembnejših pri konstrukciji kriptografsko varnih psevdonaključnih zaporedij. Seveda linearna zahtevnost ne sme biti previsoka. To je pokazal Rueppel [80] leta 1986 na primeru zaporedja, katerega varnost je temeljila na teži problema nahrbtnika. Če hočemo, da zaporedje izgleda naključno, se lahko njegova linearna zahtevnost razlikuje od pričakovane, to je $n/2$, za največ 1 ali 2, kar bomo pokazali v 5. poglavju.

Za razumevanje dela je potrebno znanje o končnih obsegih, ki ga lahko dobimo v Vidav [92], lahko pa tudi v vsakem učbeniku iz teorije kod za odpravljanje napak, kot so to Berlekamp [7], Vanstone in Oorschot [93] ali pa Hankerson et al. [39]. V slednjih je povdarek na znanju, potrebnem za študij teh kod, najdemo pa tudi algoritme za iskanje minimalnih polinomov, nerazcepnih in primitivnih polinomov ipd. V delu bomo vsak dokaz zaključili z $\blacksquare$, vsak primer z $\blacktriangle$, literaturo pa bomo navajali z zaporedno številko v oglatih oklepajih.

Magistrsko delo je razdeljeno na šest poglavij:

1. poglavje nam na kratko poda nekaj splošnih dejstev in definicij iz teorije kod za odpravljanje napak. Predstavi nam lastnosti kod, kot so stopnja, razdalja, koliko napak lahko odpravijo in druge. Prikazane so tudi možne strategije dekodiranja. Izpeljane so ocene, ki povezujejo te parametre in iz katerih lahko vidimo, kako dobre kode lahko konstruiramo. Podanih je nekaj konstrukcij kod, ki dosegajo obravnavane optimalnosti. Nekatere od teh kod lahko dekodiramo v polinomskem času. Opisane so linearne kode, nato pa še podrazred cikličnih kod, kjer dobimo znanje za razumevanje njihovega najrazširjenega predstavnika, to je BCH kod, ki so opisane v drugem poglavju. Dobimo občutek za (pre)zahtevnost (časovno ali pa spominsko) postopka dekodiranja s sindromi.

2. poglavje nam izpelje razred BCH kod ter predstavi njihove karakteristike. Opisan je Petersonov algoritem za dekodiranje, ki je prvi algoritem za dekodiranje BCH kod v polinomskem času. Ta algoritem je Berlekamp bistveno pospešil s hitrim iterativnim algoritmom za iskanje rešitve ključne enačbe. Do leta 1968 so bile BCH kode pomembne bolj iz teoretičnega vidika, saj je bil najtežji del dekodiranja, to je reševanje ključne enačbe s pomočjo reševanja linearnega sistema, računsko prezahteven. Nato so predstavljene še Reed-Solomonove (RS) kode, ki so poseben primer BCH kod. Razdalja RS kod ne more biti večja od konstrukcijske, saj se operacije pri dekodiranju odvijajo v obsegu simbolov kodnih besed. Prvotna definicija RS kod je bila z uporabo problema rekonstrukcije polinoma. Dokazana je ekvivalentnost definicij ter Guruswami-Sudanov algoritem iz leta 1999 za dekodiranje v sezname za RS kode. To je prvi algoritem polinomske časovne zahtevnosti, ki pri dekodiranju RS kod lahko pri poljubni stopnji kode z intervala $(0, 1)$ popravi več napak kot Petersonov (Berlekampov) algoritem.

3. poglavje opisuje področje kriptografije. Na kratko predstavi možnosti uporabe ter stopnje možnih napadov na kriptosisteme. Opiše nam brezpogojno in računsko varnost ter predstavi tokovne tajnopise ter pogoje za njihovo kriptografsko varnost, če jih preučujemo kot generatorje psevdonaključnih zaporedij bitov. Zaporedje je psevdonaključno, če zanj ne obstaja niti razlikovalec od porazdelitve neodvisnih metov poštenega kovanca niti napovedovalec naslednjega bita. Lastnosti sta si zelo podobni (univerzalnost napovednega testa).

4. poglavje vsebuje opis linearnih povratnih pomičnih registrov (LFSR). Izhodno zaporedje LFSR generatorja ima določene statistične lastnosti zelo lepe (Golombovi postulati naključnosti), zato je ta generator mnogokrat uporabljen kot osnova hitrih tokovnih šifer, ki temeljijo na generiranju psevdonaključnih zaporedij. Opisan je tudi Berlekamp-Masseyev algoritem za izračun profila linearne zahtevnosti zaporedja, s pomočjo lastnosti algoritma pa izpeljemo tudi pričakovano vrednost in varianco za linearno zahtevnost. Na koncu opišemo nekaj tehnik nelinearizacije LFSR-jev, ki je potrebna za kriptografsko varnost.

5. poglavje nam predstavlja povzetek osnovnih napadov na tokovne šifre. Možnih napadov je še več, bolj podrobne opise in predvsem uporabne reference pa lahko dobimo v priročniku za kriptografijo [64] ali pa v zadnjem poročilu [69] kriptografskega natečaja NESSIE evropske unije. Kateri algoritmi so bili izbrani kot priporočeni s strani predstavnikov tega triletnega projekta z udeleženci s celotnega sveta je bilo objavljeno marca 2003 v razlagi [70]. V tem poglavju so opisani tudi najpomembnejši statistični testi za preverjanje varnosti.

6. poglavje je pregled algoritmov za rešitev odločitvenega problema `PRAŠTEVILO`. Podrobno je opisan in dokazan prvi determinističen algoritem polinomske časovne zahtevnosti za rešitev problema. To je AKS algoritem iz leta 2002. Do sedaj so se v algoritmih za kriptografijo z javnimi ključi uporabljali predvsem verjetnostni algoritmi za določanje naključnih praštevil. AKS algoritem jih zaradi svoje počasnosti ne bo izpodrinil, vendar pa s svojo preprostostjo obeta nove rezultate na področju preoblikovanja verjetnostnih algoritmov iz teorije števil v deterministične. Največji optimisti pa ugibajo tudi o obstoju podobnega algoritma za problem faktorizacije.

Trenutno največji izziv na področju statističnega testiranja pa predstavlja iskanje algoritma za izračun linearne zahtevnosti s k napakami poljubnega zaporedja (s poljubno periodo) ter algoritma za izračun kvadratične zahtevnosti, pri kateri še ni izpeljana formula za pričakovana vrednost. Ti problemi so opisani v sklepu naloge.

V četrtem poglavju je opisana tudi avtokorelacija zaporedij. Uporaba zaporedij z dvo-vrednostno avtokorelacijo (ponavadi največja možna nekorelacija) je prisotna tudi v razvoju komunikacij, saj nam omogoča, da podatkovni kanal razdelimo v več kanalov s pomočjo kodiranja (CMDA kanal), kar ima določene prednosti od metode izbora druge frekvence (FMDA kanal) oz. delitve kanala s pomočjo delitve časovnega intervala na podintervale (TMDA kanal). Za varno uporabo teh komunikacij, so se v zadnjem desetletju razvijali predvsem sistemi za digitalni prenos informacij, kot so GSM omrežja, internet, intranet, navidezna zasebna omrežja (VPN - Virtual Private Network) in drugi. Dejavnike, ki so vplivali na razvoj le teh, lahko razdelimo v dve skupini. V prvi so dejavniki, ki glede na svoje potrebe s svojo finančno in politično močjo narekujejo razvoj:

1. **uporabniki aplikacij** (finance,...),

2. **politika** (prepoved izvoza kriptosistemov,...),

3. **pravo** (patenti, podpisi, jamstva,...),

4. **družba** (želja po zasebnosti),

V drugi skupini pa so znanost in ljudje, ki s svojim znanjem in sredstvi ta razvoj dejansko izvajajo:

1. **matematika** (teorija števil,...),

2. **računalništvo** (analiza algoritmov) in

3. **elektrotehnika** (hardware).

Pomemben projekt, ki se je na ta način izvajal na področju kriptografije in se tudi ravno končal, je projekt NESSIE [69] (New European Schemes for Signatures, Integrity and Encryption) katerega del, ki temelji na pogojih izbora tokovne šifre kot priporočeno za uporabo, je opisan v zadnjem poglavju. Od petih kandidatov noben ni ustrezal danim zahtevam. Lahko pa upamo, da bo razvoj pripeljal do odkritja tokovne (hitre) šifre, katere varnost ne bo zaostajala za bločnimi. Poročilo o projektu je seveda idealen priročnik za pregled trenutnega stanja na področju razvoja kriptografije.

Za neljube napake, ki so ostale v delu, se bralcu opravičujem. Vse, ki jih bom uspel najti, pa bom objavil na spletni strani
`http://fmf.uni-lj.si/~novakta/research/BM-mag.htm`

Tadej Novak

`tadej.novak@fmf.uni-lj.si`

Poglavje 1

Kode za odpravljanje napak

Za izboljšanje zanesljivosti prenosa podatkov po kanalih, oziroma njihovega zapisa na medijih, kjer lahko pride do napak zaradi motenj, šibkosti signala in podobno, uporabljamo teorijo odkrivanja in odpravljanja napak. Preprosta primera uporabe sta enotna matična številka občana (EMŠO[1]) ali pa številka na bančni kartici. V obeh primerih imamo na koncu kontrolne številke, pri EMŠO pa je kontrola integrirana že v številki sami. Bolj zahtevna uporaba teorije, kot je kontrola napak pri pretipkavanju, pa se pojavi pri kontroli prenosa podatkov po radijskih kanalih, na primer iz vesoljskih plovil, ali pa pri zagotavljanju zanesljivosti zapisa v RAM-u (pomnilnik z naključnim dostopom), na trdem disku ali pa na zgoščenkah. V zvokovni tehnologiji lahko inovacijo le teh primerjamo s fonografom, ki ga je leta 1877 odkril Thomas Edison.

Osnove o kodah za odpravljanje napak si lahko preberemo v uvodnem učbeniku Vanstone in Oorschot [93] ali pa Hankerson et al. [39], mnogo obsežnejši pa je učbenik MacWilliams in Sloane [56]. Med najnovejše reference sodi Sudan [89], pregledni članek, s katerim lahko z nekaj predznanja dobimo dober občutek za stanje in trenutne odprte probleme v razvoju teorije kod. Uporaben pa je seveda tudi priročnik za teorijo kodiranja [42] (handbook). V slovenskem jeziku pa si za uvod lahko ogledamo članka Klavžar [54] in Jurišić [45].

V tem poglavju bomo najprej predstavili osnovne definicije in shemo uporabe kod za odpravljanje napak. Najpogosteje uporabljena strategija dekodiranja je dekodiranje k najbližjemu sosedu, opisali in primerjali pa bomo tudi ostale strategije. Najnovejši dosežki so bili narejeni za dekodiranje s seznami, čemur bomo več pozornosti namenili v naslednjem poglavju. V tem pa bomo opisali še linearne kode, ki so osnova za izpeljavo mnogih razredov kod. Dekodiramo jih s pomočjo sindromov in standardne tabele, pri čemer ima pomembno vlogo kontrolna matrika.

Najtežji del pri uporabi kod je dekodiranje, ki je z uporabo sindromov in standardne tabele eksponentne časovne zahtevnosti. Prvi algoritem polinomske časovne zahtevnosti je leta 1960 predstavil Peterson [74] za dekodiranje BCH kod, opis najdemo tudi v njegovem učbeniku [75]. Postopek je temeljil na dekodiranju v dveh korakih, iskanju lokacij napak in nato določitvi vrednosti napak. Bistven korak je bil pretvorba nelinearnega sistema enačb v linearnega. Berlekamp [7] pa je v svojem učbeniku predstavil hitri iterativni algoritem brez uporabe matrik,

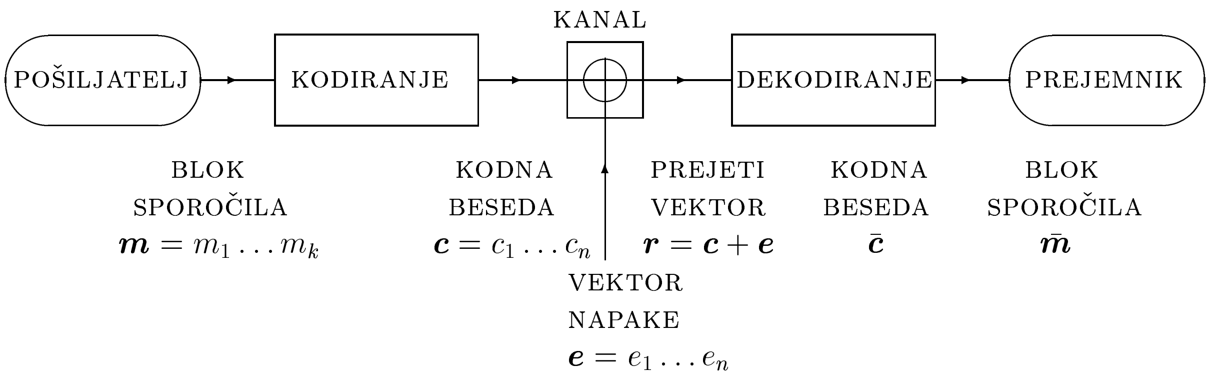
[1] EMŠO je sestavljena iz številk, ki predstavljajo rojstni datum, številko registra (50), spol in zaporedno številko, ki ji sledi kontrolna številka, taka da je vsota $7a_1+6a_2+5a_3+4a_4+3a_5+2a_6+7a_7+6a_8+5a_9+4a_{10}+3a_{11}+2a_{12}+a_{13}$ deljiva z 11, kjer so a_i cifre EMŠO. Kako se EMŠO določi, si lahko preberemo v *Uredbi o načinu določanja osebne identifikacijske številke*:
`http://objave.uradni-list.si/bazeul/URED/1999/008/B/52345854.htm`

ki poišče polinom lokatorja napake brez reševanja sistema enačb. Vse te algoritme bomo predstavili v naslednjem poglavju.

Poleg hitrosti dekodiranja je pri izbiri kod pomembno tudi, koliko napak lahko popravimo po končanem prenosu kodne besede. Primerjamo torej stopnjo vsebnosti sporočila v kodni besedi s stopnjo odpravljanja napak ali pa z relativno razdaljo. Več o teh stopnjah in ocenah optimalnosti pri konstrukciji kod bomo povedali v jedru poglavja, zaključili pa ga bomo s posebnim primerom linearnih kod, to je s cikličnimi kodami. Tu lahko kodne besede predstavimo kot polinome, podprostor vseh kodnih besed v vektorskem prostoru vseh možnih n-teric pa kot ideal v kolobarju polinomov asociiranih n-tericam. Za razdaljo cikličnih kod dobimo lep rezultat v primeru BCH kod, ki so opisane v naslednjem poglavju.

1.1 Tehnike kodiranja in dekodiranja

Predpostavimo, da preko nezanesljivega podatkovnega kanala pošiljamo neko sporočilo. Ker pri prenosu prihaja do motenj, ki povzročijo napake, želimo razviti mehanizem, ki bi nam te napake odkril in po možnosti tudi odpravil. Sporočilo razdelimo na bloke dolžine k, nato pa vsak blok $\boldsymbol{m} = m_1 \dots m_k$ posebej **zakodiramo** z injektivno funkcijo f v kodno besedo $\boldsymbol{c} = c_1 \dots c_n$ dolžine n ter jo pošljemo. Pri prenosu se pojavi napaka $\boldsymbol{e} = e_1 \dots e_n$ in prejemnik sprejme vektor $\boldsymbol{r} = \boldsymbol{c}+\boldsymbol{e}$. **Dekodiranje** poteka v dveh korakih. Najprej poiščemo kodno besedo $\bar{\boldsymbol{c}}$, ki je na nek način najbližja prejetemu vektorju (le ta ni nujno kodna beseda), nato pa s pomočjo funkcije $f^{-1}_{Im(f)}$ iz te kodne besede izluščimo blok sporočila $\bar{\boldsymbol{m}}$ in zanj upamo, da je bilo poslano.



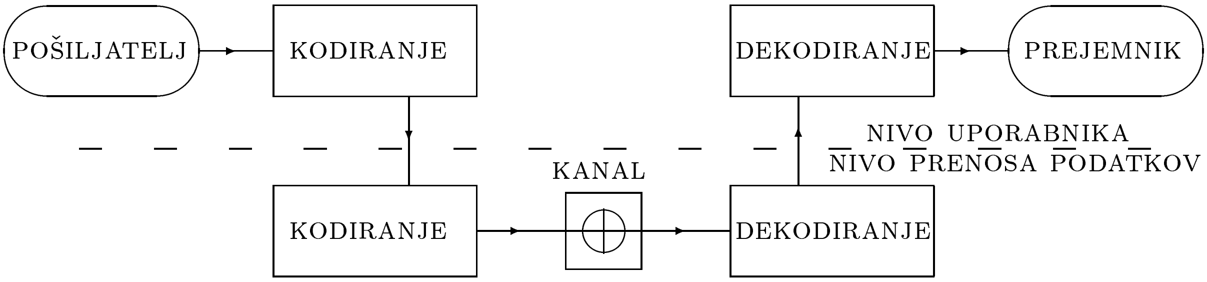
Slika 1.1: Proces pošiljanja bloka sporočila po nezanesljivem kanalu

Opisana procedura je predstavljena na sliki 1.1, njen algoritmično najtežji del pa ponavadi predstavlja prvi korak dekodiranja. Mnogokrat z besedo dekodiranje mislimo zgolj ta korak. Gledano bolj splošno, s stališča računalniških struktur, glej npr. učbenik Vidmar [94], potekata kodiranje in dekodiranje v več plasteh oz. nivojih. Na uporabniškem nivoju imamo sporočilo sestavljeno iz črk običajne abecede. To sporočilo na različnih nivojih spremenimo v binarni zapis, kompresiramo, šifriramo, kodiramo ipd. Na strani prejemnika ponovimo v obratnem vrstnem redu ustrezne inverzne postopke. Nas bosta zanimala le postopka kodiranja in dekodiranja za odpravljanje napak, ki se pojavijo pri prenosu. Ne glede na to pa po učbeniku [93] povzemimo en splošen primer.

Primer. Predpostavimo, da preko podatkovnega kanala pošiljamo informacijo, sestavljeno iz znakov iz množice $\{A, B, C, D\}$. Na nivoju uporabnika zapišemo znake v binarni zapis po naslednjem pravilu: $A \to 00$, $B \to 10$, $C \to 01$ in $D \to 11$. Naj nam posamezni znaki predstavljajo bloke sporočila. Kodiranje na nivoju prenosa podatkov pa naj bo predstavljeno s pravilom $00 \to 00000$, $10 \to 10110$, $01 \to 01011$ in $11 \to 11101$. Celotna shema je torej

$$\begin{aligned} A &\to 00 \to 00000\,, \\ B &\to 10 \to 10110\,, \\ C &\to 01 \to 01011\,, \\ D &\to 11 \to 11101\,. \end{aligned}$$

Če prejemnik prejme vektor 01000, lahko sklepa, da je pri prenosu prišlo do napake. Če predvideva, da do napak prihaja naključno in pri vsakem bitu z enako verjetnostjo (manjšo od 1/2), potem je najbolj verjetno, da je bil poslan niz 00000 in je pri prenosu prišlo do napake pri drugem bitu. Proces pošiljanja lahko ponazorimo s shemo, prikazano na sliki 1.2. ▲



Slika 1.2: Shema dvoplastnega procesa pošiljanja

Kakšno vrsto kod za odpravljanje napak bomo uporabili v posameznem primeru je odvisno od zanesljivosti prenosa podatkov, vrste napak, ki se pojavljajo (sprememba bitov, nerazpoznavanje bitov ipd), in tudi načina pojavljanja le teh, saj se mnogokrat pojavi več napak skupaj, npr. zaradi fizične poškodbe medija. Od tega je odvisna količina redundance ter uporaba določenih tehnik, od katerih je najbolj znana prepletanje, glej Vanstone in Oorschot [93, Chapter 7].

Postavimo kode za odpravljanje napak v matematične okvire. Naj bo Σ abeceda, sestavljena iz q simbolov, $q \geq 2$, na primer $\Sigma = \{a, b, \ldots, z\}$ ali pa $\Sigma = \{0, 1\}$. **Blokovna koda** C dolžine $n \geq 1$ z M kodnimi besedami nad abecedo Σ je množica M n-teric elementov iz Σ. Imenujemo jo tudi $[n, M]$**-koda nad** Σ, njene elemente pa **kodne besede**. Ponavadi je koda C injektivna slika **prostora sporočil** $\mathcal{M} = \Sigma^k$, $k \leq n$, torej je $M = q^k$. Če poteka kodiranje tako, kot je prikazano na sliki 1.1, kjer kodiramo sporočila dolžine k znakov iz abecede Σ v kodne besede dolžine n znakov na način, da sporočilu dodamo $r = n - k$ **nadzornih znakov**, lahko definiramo **stopnjo vsebnosti sporočila** $[n, M]$-kode ali na kratko **stopnjo** kot

$$R = \frac{k}{n}\,.$$

Omenjenemu postopku pravimo **razširitev sporočila**, količino $r = n - k$ pa **dodatek** kode. V splošnem razširitev sporočila ni nujno dodajanje znakov, takrat lahko definicijo stopnje posplošimo na $R = (\ln M)/n$. V primeru s strani 12 je $k = 2$, $n = 5$ in $q = 2$, saj je koda binarna.

Ker bomo pri dekodiranju na kanalu šteli napake v kodnih besedah, je smiselno definirati metriko na kodi C kot Hammingovo razdaljo. **Hammingova razdalja $d(\boldsymbol{x}, \boldsymbol{y})$ med dvema kodnima besedama** je definirana kot število koordinat, pri katerih se besedi razlikujeta. Tako je razdalja med besedama $\boldsymbol{x}$ = (abccd) in $\boldsymbol{y}$ = (cbcca) enaka 2, saj se besedi razlikujeta na prvem in petem mestu. Da je Hammingova razdalja metrika ni težko preveriti.

Definirajmo še pojem, ki ga bomo največkrat potrebovali, to je **Hammingova razdalja kode C**, ki je definirana kot

$$d = \min\{d(\boldsymbol{x}, \boldsymbol{y}) \mid \boldsymbol{x}, \boldsymbol{y} \in C, \boldsymbol{x} \neq \boldsymbol{y}\} .$$

Če jo primerjamo z dolžino kode, govorimo o **relativni razdalji** $\delta = d/n$.

Ko prejemnik na kanalu sprejme kodno besedo $\boldsymbol{r}$, jo mora po določeni strategiji dekodirati, saj je pri prenosu lahko prišlo do napake. Smiselne odločitve so:

1. ni prišlo do napak, sprejmi $\boldsymbol{r}$ kot kodno besedo,
2. prišlo je do napak, popravi $\boldsymbol{r}$ v kodno besedo $\boldsymbol{c}$ ali
3. prišlo je do napak, napak ni možno popraviti.

Če v dekodirni shemi dopuščamo uporabo odločitve 3, jo imenujemo **nepopolna dekodirna shema**, v nasprotnem primeru pa **popolna dekodirna shema**. Če je prejemnik prejel $\boldsymbol{r} \notin C$, je seveda ugotovil, da je pri prenosu prišlo do napak. Naštejmo nekaj strategij, ki jih lahko uporabimo pri dekodiranju.

Dekodiranje k najbližjemu sosedu (NND – nearest neighbour decoding) je strategija, pri kateri prejeto n-terico $\boldsymbol{r}$ dekodiramo v enolično (če obstaja) kodno besedo $\boldsymbol{c}$, ki minimizira razdaljo $d(\boldsymbol{r}, \boldsymbol{c})$.

Dekodiranje z največjim verjetjem (MLD – maximum likelihood decoding) je strategija, pri kateri prejeto zaporedje $\boldsymbol{r}$ dekodiramo v enolično (če obstaja) kodno besedo $\boldsymbol{c}$, ki maksimizira pogojno verjetnost $P(\boldsymbol{r}|\boldsymbol{c})$, to je verjetnost, da prejmemo zaporedje $\boldsymbol{r}$, če smo poslali kodno besedo $\boldsymbol{c}$.

Dekodiranje z najmanjšo napako (MED – minimum error decoding) je strategija, pri kateri prejeto zaporedje $\boldsymbol{r}$ dekodiramo v enolično (če obstaja) kodno besedo $\boldsymbol{c}$, ki minimizira pogojno verjetnost $P(\boldsymbol{c}|\boldsymbol{r})$, to je verjetnost, da je bila poslana kodna beseda $\boldsymbol{c}$, če je bilo prejeto zaporedje $\boldsymbol{r}$.

Dekodiranje v sezname (list decoding) je strategija, pri kateri si izberemo **mejo napake** e, nato pa pri dekodiranju vrnemo seznam vseh kodnih besed, ki so od prejete oddaljene za največ e, $e > 0$.

Dekodiranje do omejene razdalje (bounded distance decoding) je strategija, pri kateri si izberemo mejo napake e, nato pa pri dekodiranju vrnemo enolično (če obstaja) kodno besedo, ki je od prejete oddaljene za največ e, $e > 0$.

Nedvoumno dekodiranje (unambiguous decoding) je dekodiranje do omejene razdalje s parametrom $e = (d-1)/2$.

V nadaljevanju bomo pokazali, da so si strategije NND, MLD in MED ob določenih (smiselnih) predpostavkah ekvivalentne. Intuitivno bi rekli, da je najprimernejša strategija NND. Strategija MLD je bolj smiselna za pošiljatelja, ki ve, kaj je poslal, strategija MED pa je bolj smiselna za prejemnika, ki ve, kaj je prejel. Do sedaj ni bila še nobena od strategij NND, MLD oz. MED učinkovito implementirana v kateremkoli netrivialnem primeru. Učinkovito pomeni obstoj dekodirnega algoritma polinomske časovne zahtevnosti, netrivialni primer pa pomeni kodo, pri kateri ni moč prešteti niti vseh kodnih besed, niti vseh možnih vektorjev napak v polinomskem času.

Pri klasičnem študiju kod se ukvarjamo predvsem z nedvoumnim dekodiranjem, saj nam, kot bomo videli v trditvi 3, pri predpostavki, da je prišlo do največ e napak, zagotavlja obstoj iskane enolične kodne besede, oddaljene od prejete za največ e. Pri strategiji dekodiranja do omejene razdalje je Peterson [74] prvi predstavil dekodirni algoritem polinomske časovne zahtevnosti in sicer za BCH kode. Parameter je bil $e = (\delta - 1)/2$, pri čemer je e konstrukcijska razdalja kode, glej razdelek 2.1. Algoritem lahko sedaj, ko vemo, da so RS kode podrazred BCH kod z minimalno razdaljo enako konstrukcijski, interpretiramo kot prvi učinkovit algoritem za nedvoumno dekodiranje. Strategije dekodiranja v sezname pa se do sedaj ni pretirano raziskovalo, kljub temu, da je bila predstavljena že leta 1957. O tej strategiji si lahko več preberemo v Sudan [89] in Guruswami in Sudan [36]. Prvi rezultati, boljši od klasičnega dekodiranja BCH kod, segajo v leto 1989, trenutno najodmevnejši pa je rezultat, ki sta ga predstavila avtorja Guruswami in Sudan [35]. Predstavila sta prvi algoritem, ki pri katerikoli stopnji vsebnosti sporočila R, $0 < R < 1$, doseže strogo boljšo **stopnjo odpravljanja napak** $\varepsilon = e/n$, kot jo doseže Berlekampov algoritem. Njun rezultat si bomo podrobneje ogledali v razdelku 2.4 na strani 65.

Če privzamemo, da je koda C definirana nad abecedo s q simboli in da je verjetnost, da pride do napake pri prenosu simbola po kanalu, enaka p, $0 < p < 1 - 1/q$, ter, če pride do napake, je vsaka napaka enako verjetna, torej $p/(q-1)$, potem takemu kanalu pravimo **q-tiški simetrični kanal** z verjetnostjo napake p.

Trditev 1. *Strategija dekodiranja z največjim verjetjem (MLD) je pri uporabi q-tiškega simetričnega kanala ekvivalentna strategiji dekodiranja k najbližjemu sosedu (NND).*

Dokaz. Naj bosta zaporedji $\boldsymbol{r}$ in $\boldsymbol{c}$ na razdalji $d(\boldsymbol{r}, \boldsymbol{c}) = d$. Potem je

$$P(\boldsymbol{r}|\boldsymbol{c}) = (1-p)^{n-d} \left(\frac{p}{q-1}\right)^d ,$$

saj na $n - d$ mestih ni prišlo do napake, na ostalih d mestih pa je prišlo do točno določene napake izmed $q - 1$ možnih za vsako mesto.

Vzemimo kodni besedi $\boldsymbol{c}_1$ in $\boldsymbol{c}_2$ ter predpostavimo, da je bilo prejeto zaporedje $\boldsymbol{r}$. Pokažimo, da velja

$$P(\boldsymbol{r}|\boldsymbol{c}_1) > P(\boldsymbol{r}|\boldsymbol{c}_2) \qquad \Longleftrightarrow \qquad d(\boldsymbol{r}, \boldsymbol{c}_1) < d(\boldsymbol{r}, \boldsymbol{c}_2) .$$

Označimo $d_1 = d(\boldsymbol{r}, \boldsymbol{c}_1)$ ter $d_2 = d(\boldsymbol{r}, \boldsymbol{c}_2)$ ter iz formule za izračun $P(\boldsymbol{r}|\boldsymbol{c})$ izpeljimo

$$P(\boldsymbol{r}|\boldsymbol{c}_1) > P(\boldsymbol{r}|\boldsymbol{c}_2) \quad \Longleftrightarrow \quad (1-p)^{n-d_1} \left(\tfrac{p}{q-1}\right)^{d_1} > (1-p)^{n-d_2} \left(\tfrac{p}{q-1}\right)^{d_2} \quad \Longleftrightarrow$$

$$\Longleftrightarrow \quad (1-p)^{d_2-d_1} > \left(\tfrac{p}{q-1}\right)^{d_2-d_1} \quad \Longleftrightarrow \quad \left(\tfrac{p}{(1-p)(q-1)}\right)^{d_2-d_1} < 1 .$$

Ker velja $p < 1 - 1/q$, je $\frac{p}{(1-p)(q-1)} < 1$. Torej je

$$\left(\tfrac{p}{(1-p)(q-1)}\right)^{d_2-d_1} < 1 \quad \Longleftrightarrow \quad d_2 - d_1 > 0$$

iz česar sledi trditev. ■

Trditev 2. *Če privzamemo, da je verjetnost, da pošljemo posamezno kodno besedo po kanalu, konstantna, potem sta tudi strategiji MLD in MED ekvivalentni.*

Dokaz. Naj bo $\boldsymbol{c}$ oznaka za kodno besedo, ki jo pošiljamo, in $P(\boldsymbol{c})$ verjetnost, da pošljemo kodno besedo $\boldsymbol{c}$. Po predpostavki je $P(\boldsymbol{c})$ konstantna za vsak $\boldsymbol{c} \in C$. Pri fiksnem zaporedju $\boldsymbol{r}$, ki ga prejmemo, je

$$P(\boldsymbol{c}|\boldsymbol{r}) = \frac{P(\boldsymbol{c})}{P(\boldsymbol{r})} P(\boldsymbol{r}|\boldsymbol{c})\,.$$

Ker je ulomek v enačbi konstanten, je $\max_{\boldsymbol{c}\in C} P(\boldsymbol{c}|\boldsymbol{r})$ dosežen pri istem $\boldsymbol{c}$, kot je dosežen $\max_{\boldsymbol{c}\in C} P(\boldsymbol{r}|\boldsymbol{c})$. ■

Za kodo pravimo, da **popravi** e **napak**, če dekoder pri uporabi določene strategije lahko popravi katerikoli prejeti vektor, ki se od poslane kodne besede razlikuje pri največ e koordinatah. Pokažimo, kako na sposobnost odkrivanja in popravljanja napak vpliva Hammingova razdalja kode.

Trditev 3. *Naj bo C $[n, M]$-koda z razdaljo d in naj bo $e = \lfloor (d-1)/2 \rfloor$. Potem lahko C pri uporabi sheme NND popravi e napak. Če kodo uporabljamo le za odkrivanje napak, pa lahko odkrije $d-1$ napak.*

Dokaz. Naj bo $\boldsymbol{c}$ poslana kodna beseda in $\boldsymbol{x}$ prejeto zaporedje, tako da je $d(\boldsymbol{c}, \boldsymbol{x}) \le e$. Naj bo $\boldsymbol{c}_1$ poljubna kodna beseda. Potem iz trikotniške neenakosti $d(\boldsymbol{c}, \boldsymbol{c}_1) \le d(\boldsymbol{c}, \boldsymbol{x}) + d(\boldsymbol{x}, \boldsymbol{c}_1)$ lahko izpeljemo

$$d(\boldsymbol{x}, \boldsymbol{c}_1) \ge d(\boldsymbol{c}, \boldsymbol{c}_1) - d(\boldsymbol{c}, \boldsymbol{x}) \ge d - e \ge (2e+1) - e = e + 1\,,$$

kar pomeni, da je $\boldsymbol{c}$ najbližja kodna beseda zaporedju $\boldsymbol{x}$. Če kodo uporabljamo le za odkrivanje napak in je število napak, ki se pojavijo pri prenosu kodne besede $\boldsymbol{c}$ kvečjemu $2e$, dekoder to zazna, saj je razdalja kode C vsaj $2e+1$, kar pomeni, da nobena kodna beseda ni od $\boldsymbol{c}$ oddaljena za manj kot $2e+1$. ■

Seveda pa se nam pri dekodiranju samo po sebi postavi vprašanje, kolikšna je pri shemi NND ob uporabi q-tiškega simetričnega kanala z verjetnostjo napake p (pri prenosu posameznega simbola) sploh verjetnost pravilnega dekodiranja. To pomeni, da se nam pri prenosu kodne besede $\boldsymbol{c}$ pojavi kvečjemu toliko napak, da se prejeto zaporedje $\boldsymbol{r}$ še kar dekodira v $\boldsymbol{c}$. Recimo, da uporabljamo strategijo nedvoumnega dekodiranja s parametrom e. Naj bo $S_{\boldsymbol{c}}$ krogla z radijem e in s središčem v kodni besedi $\boldsymbol{c}$ ter $P(S_{\boldsymbol{c}}|\boldsymbol{c})$ verjetnost, da prejmemo zaporedje $\boldsymbol{r} \in S_{\boldsymbol{c}}$ ob predpostavki, da smo poslali kodno besedo $\boldsymbol{c}$. Iskana verjetnost je potem enaka

$$P(S_{\boldsymbol{c}}|\boldsymbol{c}) = \sum_{\boldsymbol{r}\in S_{\boldsymbol{c}}} P(\boldsymbol{r}|\boldsymbol{c}) = \sum_{i=0}^{e} \binom{n}{i} p^i (1-p)^{n-i}\,.$$

V zvezi s tem je zanimiv rezultat, ki ga je predstavil Shannon [81], povzemamo pa ga dopolnjenega iz učbenika Berlekamp [8, Chapter 13.8]. Naj bo $P_e(n, M)$ verjetnost napačnega dekodiranja pri uporabi najboljše kode z M kodnimi besedami dolžine n. Potem obstaja konstanta C_0, imenovana **kapaciteta kanala**, odvisna le od lastnosti kanala in nič od izbire kode, za katero velja, da če fiksiramo stopnjo kode na R, je

$$\lim_{n\to\infty} P_e(n, e^{nR}) = \begin{cases} 0 & ; \quad R < C_0 \\ 1 & ; \quad R > C_0 \end{cases}\,.$$

Znano je tudi, da verjetnost napake $P_e(n, e^{nR})$ pada pri fiksnem R eksponentno glede na n. S kakšnim parametrom eksponentno pa je eno od osnovnih eksistenčnih vprašanj Shannonove teorije.

V teoriji kodiranja sta orala ledino Shannon [81] s statističnimi modeli, z odkrivanjem eksistenčnih izrekov in s skrajno nekonstrukcijskimi rezultati ter Hamming [38] s kombinatornim pristopom in konstrukcijskimi dokazi. Shannonov pristop smo že prikazali, zapišimo še osnovno vprašanje Hammingove teorije:

> Kolikšna je največja možna razdalja kode C pri dani velikosti q abecede Σ ter parametroma n in k, to je dolžini kodnih besed $\boldsymbol{c} \in \Sigma^n$ ter dolžini sporočil $\boldsymbol{m} \in \Sigma^k$, ki jih kodiramo?

Za majhne q glede na n je tudi Hammingovo vprašanje še vedno odprto. V razdelku 1.3 bomo zbrali ocene, ki povezujejo parametre d, n in k poljubno izbrane linearne kode. Hamming se je iskanja odgovora lotil drugače. Pri fiksnem d je za izbran n konstruiral kodo z največ možnimi kodnimi besedami. To je storil za $d = 1$ (trivialen primer, kjer je $C = \Sigma^n$), $d = 2$ (pri binarnih kodah dodamo parnostni bit), $d = 3$ (bistven rezultat sedaj t.i. Hammingove kode, ki jih bomo opisali na strani 26) ter $d = 4$ (splošna konstrukcija, kako s povečanjem dolžine kode za 1 iz kode z liho minimalno razdaljo dobimo tudi minimalno razdaljo povečano za 1).

1.2 Linearne kode

Če za abecedo Σ vzamemo končni obseg $\mathbb{F} = \mathrm{GF}(q)$, potem nam Σ^n predstavlja vektorski prostor, ki ga označimo z $V_n(\mathbb{F})$. Pravimo, da je C **linearna** (n, k)**-koda** nad $\mathbb{F}$, če je definirana kot k-dimenzionalni podprostor vektorskega prostora $V_n(\mathbb{F})$. Okrogli oklepaji so rezervirani za linearne, oglati pa za splošne kode; parametra znotraj oklepajev imata različna pomena glede na vrsto oklepajev.

Orodje, s pomočjo katerega določamo razdaljo linearne kode C, je **Hammingova teža vektorja** $\boldsymbol{v} \in V_n(\mathbb{F})$, ki jo označimo z $w(\boldsymbol{v})$ in je enaka številu neničelnih koordinat vektorja $\boldsymbol{v}$. **Hammingovo teža** (n, k)**-kode** C pa definiramo kot

$$w(C) = \min\{w(\boldsymbol{x}) \mid \boldsymbol{x} \in C \setminus \{\boldsymbol{0}\}\}\,.$$

Trditev 4. [93, Theorem 3.1] *Naj bo d razdalja (n, k)-kode C. Potem je*

$$d = w(C)\,.$$

Dokaz. Ker je $d(\boldsymbol{x}, \boldsymbol{y}) = w(\boldsymbol{x} - \boldsymbol{y})$, velja

$$\begin{aligned} d &= \min\{d(\boldsymbol{x}, \boldsymbol{y}) \mid \boldsymbol{x}, \boldsymbol{y} \in C, \boldsymbol{x} \neq \boldsymbol{y}\} = \min\{w(\boldsymbol{x} - \boldsymbol{y}) \mid \boldsymbol{x}, \boldsymbol{y} \in C, \boldsymbol{x} \neq \boldsymbol{y}\} = \\ &= \min\{w(\boldsymbol{z}) \mid \boldsymbol{z} \in C \setminus \{\boldsymbol{0}\}\} = w(C)\,, \end{aligned}$$

saj je C vektorski podprostor prostora $V_n(\mathbb{F})$. ■

Naj vrstični vektorji $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ tvorijo bazo podprostora C. Potem matriko $G \in \mathbb{F}^{k \times n}$, definirano kot

$$G = \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_k \end{bmatrix},$$

imenujemo **generatorska matrika** kode C. Pravimo tudi, da je koda C **generirana** z matriko G. Če izvajamo na matriki G elementarne operacije na vrsticah (množenje vrstice s skalarjem, prištevanje vrstici večkratnik druge vrstice, menjava vrstic), dobimo matriko, ki prav tako generira kodo C. Za matriko G oblike

$$G = [I_k \; A]$$

pravimo, da ima **standardno obliko**. Za kodo C ne obstaja vedno generatorska matrika v standardni obliki. Gotovo pa obstaja ekvivalentna koda, ki ima generatorsko matriko v standardni obliki. Pravimo, da sta si (n, k)-kodi C in C' nad obsegom $\mathbb{F}$ **ekvivalentni**, če obstajata generatorski matriki G in G' za kodi C in C' ter permutacijska matrika P velikosti $n \times n$, tako da velja

$$G' = GP\,.$$

Permutacijska matrika pravzaprav premeša koordinate v kodnih besedah kode C in s tem tvori novo kodo C'.

Naj bosta $\boldsymbol{x} = (x_1x_2\dots x_n)$ in $\boldsymbol{y} = (y_1y_2\dots y_n)$ vektorja iz prostora $V_n(\mathbb{F})$. **Notranji produkt** med njima definiramo kot

$$\boldsymbol{x} \cdot \boldsymbol{y} = \sum_{i=1}^{n} x_i y_i\,.$$

To seveda ni skalarni produkt, saj zanj ne velja, da iz $\boldsymbol{x} \cdot \boldsymbol{x} = 0$ sledi $\boldsymbol{x} = 0$. Za primer lahko vzamemo $\boldsymbol{x} = (101) \in (\mathbb{Z}_2)^3$. Ni pa težko preveriti, da je komutativen,

$$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{y} \cdot \boldsymbol{x} \qquad \text{za poljubna } \boldsymbol{x}, \boldsymbol{y} \in C, \tag{1.1}$$

in linearen,

$$(\boldsymbol{x} + \boldsymbol{y}) \cdot \boldsymbol{z} = \boldsymbol{x} \cdot \boldsymbol{z} + \boldsymbol{y} \cdot \boldsymbol{z} \qquad \text{za poljubne } \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in C \qquad \text{in} \tag{1.2}$$

$$(\lambda \boldsymbol{x}) \cdot \boldsymbol{y} = \lambda(\boldsymbol{x} \cdot \boldsymbol{y}) \qquad \text{za poljubna } \boldsymbol{x}, \boldsymbol{y} \in C \text{ in poljuben } \lambda \in \mathbb{F}\,. \tag{1.3}$$

Pravimo, da sta si vektorja $\boldsymbol{x}$ in $\boldsymbol{y}$ **ortogonalna** (pravokotna), če velja $\boldsymbol{x} \cdot \boldsymbol{y} = 0$. Sedaj lahko definiramo **ortogonalni komplement** kode C

$$C^{\perp} = \{\boldsymbol{x} \in V_n(\mathbb{F}) \mid \boldsymbol{x} \cdot \boldsymbol{y} = 0 \text{ za vsak } \boldsymbol{y} \in C\}\,,$$

ki mu pravimo tudi kodi C **dualna koda**. Generatorski matriki kode $C^{\perp}$ pravimo tudi **kontrolna matrika** kode C (nekateri avtorji ji pravijo nadzorna matrika). Če je kontrolna matrika oblike $[I_k \; A]$, potem prvih k simbolov kodne besede $\boldsymbol{c} = (x_1x_2\dots x_n)$ predstavlja sporočilo $\boldsymbol{m} = (x_1x_2\dots x_k)$, simbole $x_{k+1}, \dots, x_n$ pa imenujemo **kontrolni simboli**. Ime je posledica točke 3 naslednje trditve.

Trditev 5. [93, Theorem 3.3, 3.4] *Naj bo C (n, k)-koda nad obsegom $\mathbb{F}$. Potem velja*

1. *$C^{\perp}$ je $(n, n-k)$-koda nad obsegom $\mathbb{F}$,*
2. *če je $G = [I_k \; A]$ generatorska matrika kode C, potem je $H = [-A^T \; I_{n-k}]$ generatorska matrika kode $C^{\perp}$ (kontrolna matrika kode C) in*
3. *če je H kontrolna matrika kode C, potem je $\boldsymbol{x} \in V_n(\mathbb{F})$ kodna beseda natanko tedaj, ko je $H\boldsymbol{x}^T = 0$.*

Dokaz. Dokazali bomo pravilnost točke 1, dokaz ostalih dveh točk pa je posledica dejstev iz dokaza prve. Pokazati moramo, da je $C^\perp$ vektorski podprostor prostora $V_n(\mathbb{F})$, torej da je zaprt za seštevanje in množenje s skalarjem. Naj bosta $\boldsymbol{x}, \boldsymbol{y} \in C^\perp$. Lastnost (1.2) notranjega produkta nam zagotovi, da je za vsak $\boldsymbol{z} \in C$

$$(\boldsymbol{x}+\boldsymbol{y})\cdot\boldsymbol{z} = \boldsymbol{x}\cdot\boldsymbol{z} + \boldsymbol{y}\cdot\boldsymbol{z} = 0+0 = 0\,,$$

torej je tudi $\boldsymbol{x}+\boldsymbol{y} \in C^\perp$. Podobno dobimo iz lastnosti (1.3) zaprtost za množenje s skalarjem.

Sedaj moramo pokazati še, da je $C^\perp$ $(n-k)$-dimenzionalen podprostor prostora $V_n(\mathbb{F})$. Lastnost, ki jo preučujemo, se pri ekvivalentnih kodah (razlikujejo se le v vrstnem redu zapisa koordinat) ohranja. Zato lahko brez škode za splošnost privzamemo, da ima koda C generatorsko matriko G standardne oblike, $G = [I_k\ A]$. Naj bo $\mathrm{Lin}(H)$ podprostor napet na vrsticah matrike $H = [-A^T\ I_{n-k}]$. Ker je

$$HG^T = [-A^T\ I_{n-k}][I_k\ A]^T = -A^T I_k^T + I_{n-k}A^T = -A^T + A^T = \mathbf{0}_{(n-k)\times k}\,,$$

so vse vrstice matrike H ortogonalne na generatorje kode C, torej na celoten C, saj je notranji produkt linearen. Zato je $\mathrm{Lin}(H) \subset C^\perp$.

Pokažimo še, da je $C^\perp \subset \mathrm{Lin}(H)$. Vzemimo poljuben $\boldsymbol{x} \in C^\perp$, $\boldsymbol{x} = (x_1x_2\dots x_n)$ in tvorimo

$$\boldsymbol{y} = \boldsymbol{x} - \sum_{i=1}^{n-k} x_{i+k}\boldsymbol{r}_i\,,$$

kjer je $\boldsymbol{r}_i$ i-ta vrstica matrike H. Ker $\boldsymbol{x} \in C^\perp$ in ker so tudi $\boldsymbol{r}_i \in C^\perp$, $1 \le i \le k$, saj smo pokazali $\mathrm{Lin}(H) \subset C^\perp$, je potem tudi $\boldsymbol{y} \in C^\perp$. Iz oblike matrike H in definicije vektorja $\boldsymbol{y}$ ni težko sklepati, da ima $\boldsymbol{y}$ obliko $\boldsymbol{y} = (y_1y_2\dots y_k0\dots0)$. Prav tako iz oblike matrike G in dejstva, da je $\boldsymbol{y} \in C^\perp$, kar pomeni $G\boldsymbol{y}^T = \mathbf{0}$, lahko sklepamo, da ima $\boldsymbol{y}$ obliko $\boldsymbol{y} = (0,\dots,0,y_{k+1},y_{k+2},\dots,y_n)$. Torej je $\boldsymbol{y} = \mathbf{0}$ in

$$\boldsymbol{x} = \sum_{i=1}^{n-k} x_{i+k}\boldsymbol{r}_i\,,$$

kar pomeni, da je $C^\perp \subset \mathrm{Lin}(H)$. Torej je $C^\perp = \mathrm{Lin}(H)$ in ker so vrstice matrike H linearno neodvisne, je dimenzija kode $C^\perp$ enaka $n-k$. ■

Zopet ni odveč opomba, da notranji produkt ni skalarni produkt in $C^\perp$ ni ortogonalni komplement, od koder bi lahko uporabili napačni razmislek, da je vsota njunih dimenzij enaka dimenziji celega prostora, in dejali, da je polovica zadnjega dokaza nepotrebna. Naj omenimo, da obstajajo **sebi-ortogonalne** linearne kode, to so tiste, za katere velja $C \subset C^\perp$, in **sebi-dualne** linearne kode, to so tiste, za katere velja $C = C^\perp$.

Če želimo definirati kodo C, lahko to storimo tako, da določimo njeno generatorsko matriko ali pa njeno kontrolno matriko. V obeh primerih je koda enolično določena. Pri uporabi kod moramo vedno implementirati kodiranje ter dekodiranje. Eden od preprostih (ne nujno najučinkovitejši) načinov za kodiranje je, da sporočilo $(m_1m_2\dots m_k)$ zgolj razširimo v kodno besedo

$$\boldsymbol{c} = (\underbrace{m_1m_2\dots m_k}_{\text{sporočilo}}, \underbrace{x_1x_2\dots x_{n-k}}_{\text{dodatek}})\,.$$

Simbole $x_1 x_2 \dots x_{n-k}$ imenujemo **kontrolni simboli** in jih enolično določimo iz matrične enačbe

$$Hc^T = 0\,,$$

kjer je H kontrolna matrika kode C.

Ugotavljanje razdalje kode po definiciji razdalje ali pa z direktnim izračunom teže kode je za kode z mnogo koordinatami računsko zelo zahtevno (eksponentno, glede na dolžino besed). Ker se razdaljo izračuna le enkrat, je lahko tak napor upravičen. Vendar pa bi si želeli imeti orodje, ki nam to stori v polinomskem času. Pri linearnih kodah tako orodje obstaja in ga bomo opisali v naslednji trditvi.

Trditev 6. [93, Theorem 3.5] *Naj bo H kontrolna matrika (n, k)-kode C nad obsegom $\mathbb{F}$. Potem je vsak nabor $s-1$ stolpcev matrike H linearno neodvisen natanko tedaj, ko ima koda C razdaljo vsaj s.*

Dokaz.
($\Rightarrow$) Predpostavimo, da je vsak nabor $s-1$ stolpcev matrike H linearno neodvisen nad $\mathbb{F}$. Naj bo $\boldsymbol{c} = (c_1 c_2 \dots c_n)$ neničelna kodna beseda in naj bodo $\boldsymbol{h}_1, \boldsymbol{h}_2, \dots, \boldsymbol{h}_n$ stolpci matrike H. Ker je H kontrolna matrika, je $H\boldsymbol{c}^T = 0$, kar lahko zapišemo kot

$$H\boldsymbol{c}^T = \sum_{i=1}^{n} c_i \boldsymbol{h}_i = 0\,.$$

Če je $w(\boldsymbol{c}) \leq s-1$ imamo zapisano netrivialno linearno kombinacijo $s-1$ ali manj vektorjev, kar se po predpostavki ne more zgoditi. Torej je $w(\boldsymbol{c}) \geq s$ za vsak neničeln $\boldsymbol{c} \in C$ in zato je $w(C) \geq s$. Z upoštevanjem trditve 4 dobimo, da ima C razdaljo vsaj s.
($\Leftarrow$) S protislovjem. Predpostavimo sedaj, da ima koda C razdaljo vsaj s in da obstaja $t \leq s-1$ stolpcev matrike H, recimo $\boldsymbol{h}_{i_1}, \boldsymbol{h}_{i_2}, \dots, \boldsymbol{h}_{i_t}$, ki so linearno odvisni. Potem obstajajo skalarji $\lambda_{i_1}, \lambda_{i_2}, \dots, \lambda_{i_t}$, ki niso vsi enaki 0, taki da je

$$\sum_{j=1}^{t} \lambda_{i_j} \boldsymbol{h}_{i_j} = 0\,.$$

Definirajmo vektor $\boldsymbol{c}$, ki ima na koordinatah i_j, $1 \leq j \leq t$, elemente λ_{i_j}, drugje pa ničle. Vektor $\boldsymbol{c}$ je neničeln in je kodna beseda iz C, saj iz zadnje enačbe sledi $H\boldsymbol{c}^T = 0$. Ker je $w(\boldsymbol{c}) = t \leq s-1$, smo prišli v protislovje s predpostavko, da je razdalja kode C vsaj s. Torej ne obstaja nabor manj kot s stolpcev matrike H, ki bi bili linearno odvisni. ■

1.3 Vrste kod, konstrukcije in ocene

V tem razdelku bomo naredili pregled najbolj poznanih kod. Povedali bomo, kako jih konstruiramo ter kako lahko z njihovimi kombinacijami dobimo nove kode. Posamezne konstrukcije, kakor tudi kombinacije, so pomembne zato, ker z njimi dosežemo nekatere optimalne lastnosti. Ker se določene lastnosti izključujejo med seboj in ker je tudi mnogo povezav med lastnostmi še neodkritih, ni nekega splošnega postopka, s katerim bi dobili optimalno kodo. Nekaj ocen, ki povezujejo dolžino kode, dimenzijo prostora sporočil, razdaljo kode in velikost abecede (parametri n, k, d in q), bomo navedli ob koncu razdelka. Poudarek bomo dali na asimptotične ocene, to so relacije med $\delta = d/n$ in $R = k/n$, ko gre $n \to \infty$.

Konstrukcije novih kod iz že poznanih

Po MacWilliams in Sloane [56, Chapter 1.9] bomo povzeli osnovne konstrukcije novih kod iz že poznanih. S pomočjo zadnje bomo konstruirali t. i. razred asimptotično dobrih kod.

Dodajanje nadzornega bita (Adding an overall parity check). Če v binarni linearni (n, k)-kodi C z liho razdaljo d obstaja kodna beseda z liho težo, potem lahko dobimo novo kodo, tako da vsaki kodni besedi $(x_1 \dots x_n)$ dodamo bit x_{n+1}, tako da velja $x_1 + \cdots + x_n + x_{n+1} = 0$. Ker je teža vseh kodnih besed sedaj soda, je razdalja nove kode enaka $d + 1$.

Prebod kode z izbrisom koordinat (Puncturing a code by deleting coordinates). Če v linearni kodi vsem kodnim besedam izbrišemo i-to koordinato za nek izbran i, dobimo novo linearno kodo z boljšo stopnjo. Če smo imeli srečo, se razdalja d ni spremenila.

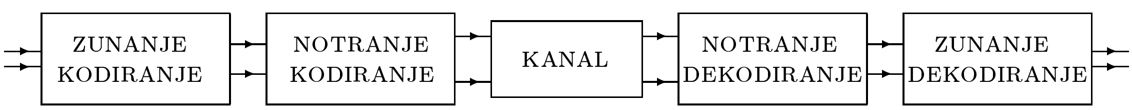
Iztrebljanje kode z odstranjevanjem kodnih besed (Expurgating by throwing away codewords). Če je v binarni linearni (n, k)-kodi C z razdaljo d odstranimo vse kodne besede z liho težo, dobimo $(n, k-1)$-kodo s ponavadi večjo razdaljo (npr. če je razdalja d liha).

Razširjanje kode z dodajanjem kodnih besed (Augmenting by adding new codewords). Linearni (n, k)-kodi C z razdaljo d lahko dodamo kodno besedo $\mathbf{1} = (1 \dots 1)$ (če je seveda že ne vsebuje) in dobimo $(n, k+1)$-kodo $C \cup (\mathbf{1} + C)$. Razdalja nove kode je potem enaka $\min\{d, n-d\}$.

Podaljševanje z dodajanjem novih koordinat (Lengthening by adding message symbols). Linearni kodi lahko dodamo kodno besedo $\mathbf{1}$, tako kot v prejšnji točki, nato pa vsem kodnim besedam dodamo še en kontrolni bit. S postopkom simuliramo dodajanje novega simbola sporočila.

Krajšanje kode z vzetjem preseka (Shortening a code by taking a cross-section). Iz linearne kode vzamemo vse kodne besede, ki se začnejo z 0, ter jim izbrišemo prvi simbol.

Združevanje kod (Concatenated codes). Postopek za združevanje kod je prvi predstavil Forney [25], predstavljen pa je na sliki 1.3. Naj bo notranja koda linearna (n_1, k_1)-koda nad abecedo s q_1 simboli in z razdaljo d_1 ter zunanja koda linearna (n_2, k_2)-koda nad abecedo s $q_2 = q_1^{k_1}$ simboli in z razdaljo d_2. Torej je število simbolov zunanje kode ravno enako številu kodnih besed notranje kode. Zato lahko kodiranje izvedemo na način, da najprej kodiramo blok sporočila z zunanjo kodo, nato pa zakodiramo še vsak dobljen simbol. Združena koda je potem linearna $(n_1 n_2, k_1 k_2)$-koda nad abecedo s q_1 simboli in z razdaljo $d_1 d_2$.



Slika 1.3: Lepljenje kod

Obstoj asimptotično dobrih razredov kod

Da bi čimbolje izkoristili kapaciteto prenosnega kanala, si želimo imeti pri dani stopnji kode (delež koristnih informacij) čimvečjo zmožnost odkrivanja in odpravljanja napak, torej čimvečjo relativno razdaljo kode. Naj bo pri dani dolžini n in stopnji R koda $C_{n,R}$ koda z največjo možno relativno razdaljo $\delta_{n,R}$. Naj bo C koda dolžine $2n$ definirana kot $C = \{\boldsymbol{c}_1\boldsymbol{c}_2 \,|\, \boldsymbol{c}_1, \boldsymbol{c}_2 \in C_{n,R}\}$. Stopnja kode C je očitno enaka R, relativna razdalja pa $\delta/2$, saj če sta kodni besedi $\boldsymbol{c}_1$ in $\boldsymbol{c}_2$ na minimalni razdalji v kodi $C_{n,R}$, potem sta kodni besedi $\boldsymbol{c}_1\boldsymbol{c}_1$ in $\boldsymbol{c}_1\boldsymbol{c}_2$ na minimalni razdalji v kodi C. Pri takšni konstrukciji kod večje dolžine gre relativna razdalja proti 0.

Zastavlja se vprašanje, ali pri poljubnem n in R obstaja koda z dolžino strogo večjo od n ter s stopnjo in relativno razdaljo vsaj tolikšno, kot jo ima koda $C_{n,R}$. S tem bi radi poiskali kodo, ki ima pri dani stopnji čim večjo relativno razdaljo, in tako pri dani stopnji zagotovili optimalen prenos podatkov po kanalu. Malo manj zahtevno vprašanje je vprašanje o obstoju razreda asimptotično dobrih kod. Za razred kod $\mathcal{C} = \{C_\lambda | \lambda \in \mathbb{N}\}$ pravimo, da je **asimptotično dober**, če je

$$\inf_{\lambda\in\mathbb{N}} R_\lambda > 0 \qquad \text{in} \qquad \inf_{\lambda\in\mathbb{N}} \delta_\lambda > 0\,,$$

kjer je R_λ stopnja kode C_λ, δ_λ pa relativna razdalja kode C_λ, $\lambda \in \mathbb{N}$.

Obstoj razreda asimptotično dobrih kod je pokazal Gilbert s konstrukcijo naključnih kod, ki jih bomo spoznali v naslednjem razdelku. Forney [25] pa je prvi predstavil konstrukcijo asimptotično dobrih kod s kodiranjem in dekodiranjem v polinomskem času, vendar pa tega ni omenil eksplicitno. Svoj rezultat je predstavil kot algoritmično verzijo (Hammingov stil) Shannonovega izreka. Pri združevanju kod je za zunanjo kodo vzel Reed-Solomonovo (le te bomo opisali v razdelku 2.4) ter napisal algoritem polinomske časovne zahtevnosti za iskanje optimalne notranje kode. Tudi algoritma za kodiranje in dekodiranje razreda teh kod sta bila polinomske časovne zahtevnosti, verjetnost napake pri dekodiranju posamezne kode pa se je z njeno dolžino eksponentno zmanjševala.

Prvo eksplicitno konstrukcijo razreda asimptotično dobrih kod je predstavil Justesen [46], ki je nadgradil Fourneyevo idejo. Konstrukcijo bomo povzeli po učbeniku [56], dokaz o velikosti relativnih razdalj Justesenovih kod pa si bralec lahko pogleda v [56, Chapter 10.11]. Naj bo $C_{n,k}$ Reed-Solomonova koda (definirali jo bomo v razdelku 2.4) dolžine n in dimenzije k nad $\mathbb{F} = \mathrm{GF}(2^m)$. **Justesenova koda** $J_{n,k}$ je množica kodnih besed, ki jih dobimo tako, da kodno besedo $\boldsymbol{a} = (a_0 \dots a_{n-1})$ iz kode $C_{n,k}$ preslikamo v vektor

$$\boldsymbol{b} = (a_0, a_0; a_1, \alpha a_1; a_2, \alpha^2 a_2; \dots; a_{n-1}, \alpha^{n-1} a_{n-1})\,,$$

kjer je α primitivni element v $\mathbb{F}$ (generator multiplikativne grupe $\mathbb{F}\backslash\{0\}$), nato pa posamezne elemente vektorja $\boldsymbol{b}$ zamenjamo z binarnimi m-tericami. Dobljena koda $J_{n,k}$ je primer združevanja kod z uporabo n različnih notranjih kod. To je binarna linearna $(2mn, mk)$-koda in je stopnje $R_m = k/2n$.

Če pri fiksnem R, $0 < R < \frac{1}{2}$, za vsak $m \in \mathbb{N}$ vzamemo $k = \lceil R\,2n \rceil = \lceil R\,2(2^m - 1) \rceil$ in definiramo $C_m = J_{n,k}$, dobimo razred binarnih linearnih kod stopenj $R_m \geq R$ ter z relativnimi razdaljami $d_m \geq (H_2^{-1}(0.5) - o(m))\,(1 - 2R) \approx 0.110\,(1 - 2R)$, kjer je H_2 funkcija entropije definirana za $0 \leq x \leq 1$ kot $H_2(x) = -x \log_2 x - (1 - x) \log_2(1 - x)$. Razred kod $\{C_m | m \in \mathbb{N}\}$ je asimptotično dober.

Shannonova kapaciteta kanala

Že Shannon [81] je v svojem delu nakazal, da je **naključna koda**, ki jo bomo s konstrukcijo definirali v naslednjem odstavku, precej dobra, vendar pa tega ni pokazal eksplicitno. To je leta 1952 storil Gilbert, kasneje pa je njegovo oceno Varshamov dosegel s pomočjo naključne linearne kode.

Naj bo $V_q(n,r)$ **volumen krogle** z radijem r v Σ^n, kjer je $|\Sigma| = q$. Če je $(K-1)V_q(n,d) < q^n$, potem lahko s pomočjo požrešne metode konstruiramo kodo C s $K = q^k$ elementi in razdaljo d. Na vsakem koraku izberemo vektor, ki še ni v C in je na razdalji vsaj d od poljubne kodne besede iz C. S tem na vsakem koraku izločimo (ne več kot) $V_q(n,d)$ vektorjev iz Σ^n, ki jih ne bomo mogli dati v C v naslednjih korakih (Gilbertova konstrukcija). Varshamov je pokazal, da lahko na podoben način konstruiramo linearno kodo. Če velja $(K-1)V_q(n,d) < q^n$, potem je

$$q^k\, V_q(n,d) \le q^n\,. \tag{1.4}$$

Poglejmo, kaj nam pove zgornja ocena za velike n. Če za $0 < \delta < 1$ definiramo

$$H_q(\delta) = \delta \log_q\left(\frac{q-1}{\delta}\right) + (1-\delta)\log_q\left(\frac{1}{1-\delta}\right),$$

lahko s pomočjo Stirlingove formule dobro aproksimiramo $V_q(n,\delta n) \approx q^{H_q(\delta)n}$. Ta aproksimacija in Gilbertova (Varshamova) konstrukcija skupaj z oceno (1.4), ki jo preoblikujemo v $k \ge n - \log_q(V_q(n,d))$, nam povesta, da obstaja (linearna) koda z dolžino n, razdaljo δn in dolžino sporočila $k \approx n(1 - H_q(\delta))$. Za velike n torej obstaja koda stopnje

$$R \ge 1 - H_q(\delta) \qquad\qquad \text{(Gilbert-Varshamova ocena)}\,,$$

ki doseže relativno razdaljo δ in torej popravi največ $\delta n/2$ napak.

Drugačno interpretacijo zgornjega rezultata je dobil že Shannon [81], glej [56, Chapter 1.6, Theorem 7], s verjetnostnimi metodami in brez konstrukcij. Definiral je kapaciteto q-tiškega simetričnega kanala z verjetnostjo napake p pri prenosu posameznega simbola kot

$$C_0(p) = 1 - H_q(p) \qquad\qquad \text{(Shannonova kapaciteta kanala)}$$

in pokazal, da za vsak $\varepsilon > 0$ in vsak R, $0 < R < C_0(p)$, obstaja (n,k)-koda s stopnjo $k/n > R$ in vrednostjo $P_{err} < \varepsilon$, kjer je P_{err} povprečna verjetnost napake pri prenosu posameznih simbolov.

Neeksistenčni rezultati

Naj bo C poljubno izbrana koda nad abecedo velikosti q dolžine n, velikosti $M = q^k$ in z razdaljo d. Povzeli bomo nekaj najbolj znanih neenakosti, ki povezujejo parametre n, k, d in q. Singleton [85] je predstavil preprosto oceno, ki jo dobimo, če kodo projeciramo na $k-1$ koordinat. Ker je dimenzija kode enaka k, se po Dirichletovem principu vsaj dve kodni besedi preslikata v isto. Torej sta se taki dve kodni besedi prej razlikovali pri največ $n-k+1$ koordinatah, torej velja

$$d \le n - k + 1 \qquad\qquad \text{(Singletonova ocena)}\,.$$

Singletonova ocena za razdaljo kode je preprosta, pa tudi zelo dobra, saj jo nekatere kode dosežejo, npr. Reed-Solomonove kode. Kode, ki dosežejo enakost v Singletonovi oceni, imenujemo **maksimalno razdaljno separabilne** (MDS - maximum distance separable). Ocena je neodvisna od parametra q, njena slabost pa je, da jo kode običajno dosežejo le pri velikih q.

Poiščimo še kako oceno za majhne q. Hamming [38] je upošteval, da se krogle z radijem $d/2$ okrog kodnih besed ne sekajo, torej je

$$\begin{aligned} q^k\, V_q(n, d/2) &\leq q^n && \text{(Hammingova ocena) in} \\ R &\leq 1 - H_q(\delta/2) && \text{(asimptotično)}\,. \end{aligned}$$

Asimptotično oceno smo dobili analogno kot Gilbert-Varshamovo. Obstajajo kode, pri katerih je dosežena enakost v neasimptotični Hammingovi oceni. Takšne kode imenujemo **popolne kode**. Trivialni primeri so $C = \{\mathbf{0}\}$, $C = \Sigma^n$ in za lihe n tudi $C = \{\mathbf{0}, \mathbf{1}\}$. Edine (poleg ekvivalentnih) netrivialne popolne linearne kode nad končnimi obsegi so:

1. Hammingove $\left(\frac{q^m - 1}{q - 1}, n - m\right)$-kode, $m \in \mathbb{N}$, nad GF(q) z razdaljo $d = 3$ (opisali jih bomo na koncu tega razdelka),
2. binarna Golayeva $(23, 12)$-koda z razdaljo $d = 7$ in
3. Golayeva $(11, 6)$-koda nad GF(3) z razdaljo $d = 5$.

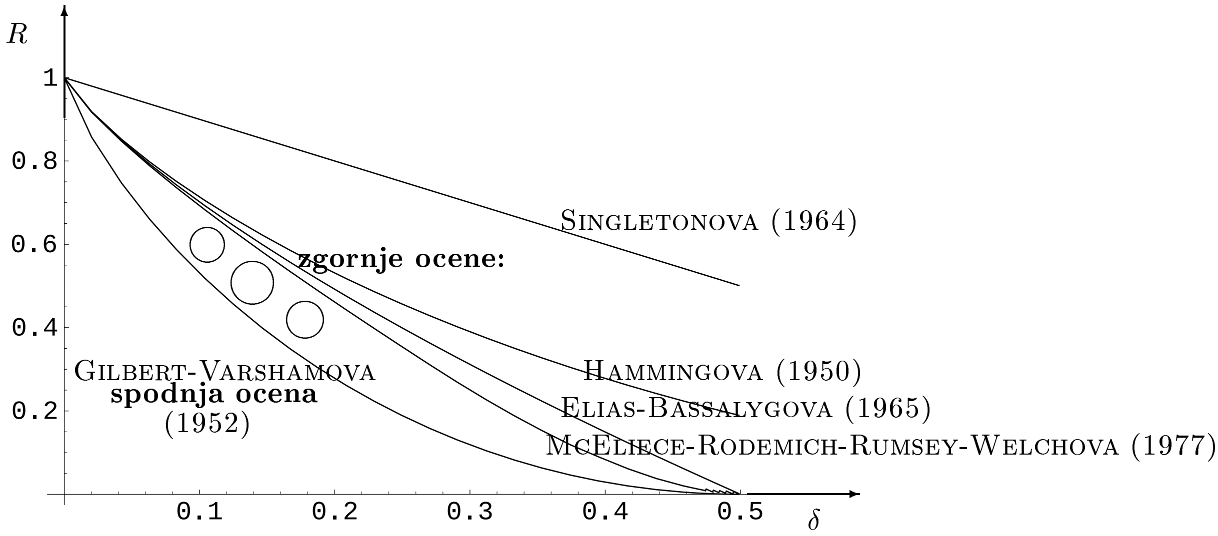
Parametri (n, k, d, q) poljubne netrivialne nelinearne popolne kode nad GF(q) so enaki parametrom ene od ravnokar omenjenih linearnih kod. To klasifikacijo je začel dokazovati že Van Lint, Tietäväinen pa je leta 1973 dokaz zaključil. Dokaz je preobsežen za to poglavje, najdemo ga v učbeniku [56, Chapter 6.10]. Iz povedanega lahko sklepamo, da je pogoj $M = q^k$ eden od potrebnih pogojev za popolnost kode.

Hammingova ocena je dobra le za majhne d. Enakost v oceni lahko dosežemo, kot smo že omenili, le za $d = 1$, $d = 3$, $d = 5$ in $d = 7$. Asimptotično gledano je obravnavanje primera, ko gre relativna razdalja δ proti 0, trivialno, saj nam naključne pa tudi Hammingove kode zagotavljajo obstoj kod, za katere gre stopnja R kode proti 1. Torej se pri $\delta \to 0$ zgornja in spodnja meja ujemata, glej sliko 1.4. Kaj pa za večje δ? Spodnja meja za R, to je Gilbert-Varshamova ocena, je pri $\delta = (q-1)/q$ enaka 0. Spodnja meja nam garantira eksistenco neke kode stopnje R, torej pri $\delta = (q-1)/q$ ne obstaja naključna koda s stopnjo večjo od 0. Ali obstaja kaka druga koda pri tem δ s stopnjo $R > 0$? Odgovor je negativen, kar sta prva pokazala Elias in Bassalygo, neodvisno drug od drugega. S pomočjo preproste ideje, da je v poljubni krogli z radijem $d/2$ okoli poljubnega vektorja iz Σ^n kvečjemu ena kodna beseda, sta pokazala, da je v poljubni krogli z radijem $r = \frac{q-1}{q}\left(n - \sqrt{n(n - \frac{q}{q-1}d)}\right)$ okoli poljubnega vektorja iz Σ^n kvečjemu qn kodnih besed. Torej, če za vsako kodno besedo preštejemo vse vektorje, ki so okoli nje na razdalji največ r, smo poljuben vektor iz Σ^n šteli največ qn-krat, kar pomeni $q^k\, V_q(n, r) \leq qnq^n$. Oceno lahko spremenimo tudi v asimptotično obliko, podroben dokaz obeh pa si lahko ogledamo v Bassalygo [6]. Oceni sta

$$\begin{aligned} q^k\, V_q(n, r) &\leq qnq^n && \text{(Elias-Bassalygova ocena) in} \\ R \leq 1 - H_q\!\left(\left(1 - \frac{1}{q}\right)\left(1 - \sqrt{1 - \frac{q}{q-1}\delta}\right)\right) & && \text{(asimptotično)}\,. \end{aligned}$$

Če pogledamo oceno pri $\delta = (q-1)/q$, vidimo, da se ujema s spodnjo oceno. V učbeniku [56, Chapter 17] je zastavljen odprt raziskovalni problem:

> Kakšna je natančna zgornja meja za stopnjo R kot funkcijo relativne razdalje δ, ko gre $n \to \infty$? Ali limita $\lim_{n\to\infty} R$ sploh obstaja pri izbranem δ, $0 < \delta < (q-1)/q$?



Slika 1.4: Asimptotične ocene med stopnjo R in relativno razdaljo δ pri $q = 2$ (binarne kode)

Problem je leta 2001 ponovno izpostavil Sudan [89] v svojem preglednem članku. Od preteklih rezultatov reševanja problema naj izpostavimo dva. Bilo je pokazano, da se pri $\delta \to 0$ zgornja ocena bolj prilega iskani natančni zgornji meji, pri $\delta \to (q-1)/q$ pa spodnja ocena. Drugi rezultat pa je dokaz obstoja zaporedja kod, ki presežejo Gilbert-Varshamovo oceno. Za ta rezultat so avtorji Tsfasman, Vlăduţ in Zink [91] prejeli nagrado "IEEE Information Theory Group Paper Award for 1983". Če primerjamo Gilbert-Varshamovo spodnjo ter Hammingovo (in tudi Elias-Bassalygovo) zgornjo oceno, vidimo, da se njuni strmini pri $\delta \to 0$ razlikujeta za faktor 2, vendar pa so bile za majhne δ odkrite boljše kode od naključnih. Če pa primerjamo hitrost padanja Gilbert-Varshamovo spodnje ter Elias-Bassalygova zgornje meje pri $\delta \to (q-1)/q$, pa lahko s pomočjo hitrosti padanja McEliece-Rodemich-Rumsey-Welchove [62] (MRRW) zgornje ocene, ki pravi, da je za majhne ε pri $\delta = (q-1)/q - \varepsilon$ stopnja $R = O(\varepsilon^2 \log \frac{1}{\varepsilon})$, kar je skoraj enako padanju Gilbert-Varshamove spodnje ocene. Grafe ocen si lahko ogledamo na sliki 1.4, kjer smo jih predstavili pri $q = 2$. Pri $q = 2$ lahko MRRW oceno povzamemo iz učbenika [56] kot

$$R \le \min_{\substack{0<\delta<(q-1)/q \\ 0<u\le 1-2\delta}} B(u,\delta) \qquad \text{(MRRW ocena za } q=2\text{)},$$

kjer je

$$B(u,\delta) = 1 + h(u^2) - h(u^2 + 2\delta u + 2\delta) \qquad \text{in} \qquad h(x) = H_2\Big(\frac{1}{2} - \frac{1}{2}\sqrt{1-x}\Big).$$

Za $\delta > 0.273$ pa se ocena poenostavi v

$$R \le H_2\Big(\frac{1}{2} - \sqrt{\delta(1-\delta)}\Big) \qquad \text{(MRRW ocena za } q=2 \text{ in } \delta > 0.273\text{)}.$$

Ker za Reed-Solomonove (RS) kode, glej razdelek 2.4, velja $d = n - k + 1$, je torej Singletonova ocena natančna zgornja meja pri iskanju optimalnih kod. Če premečemo sumande in enačbo delimo z n, dobimo oceno $R = 1 - \delta$. Ali je to potem natančna zgornja meja pri asimptotičnem iskanju optimalnih kod? Seveda ne. Čeprav lahko konstruiramo RS kodo pri poljubno velikem n, ne moremo dobiti asimptotično dobrega razreda teh kod. To je v učbeniku [56, Chapter 9.5] pokazano za BCH kode, videli pa bomo, da so RS kode poseben primer BCH kod. RS kode torej dokazujejo le očitno dejstvo, da gre stopnja R asimptotično optimalnih kod proti 1, če gre relativna razdalja δ proti 0.

Najbolj znane vrste kod

Velik razred kod za odpravljanje napak predstavljajo linearne kode. Glede na namen oz. področje uporabe upoštevamo faktorje, kot so zanesljivost kanala, pojavljanje posebnih vrst napak, zahtevana hitrost dekodiranja ipd izberemo določeno podvrsto linearnih kod. To so lahko Reed-Mullerjeve kode (uporaba Hadamardovih matrik), sebi dualne kode (npr. Golayeva koda) ali pa ciklične kode, pri katerih so najbolj znane BCH kode, katerim bomo posvetili največ pozornosti, saj je zanje leta 1967 Berlekamp predstavil hiter algoritem za dekodiranje, katerega je leta 1968 tudi zapisal v [7].

Hammingove kode reda r nad GF(q) so (n, k)-kode, kjer je $n = (q^r - 1)/(q - 1)$ in $k = n - r$, s kontrolno matriko H_r velikosti $r \times n$, tako da noben stolpec ni večkratnik kakega drugega stolpca. Primer Hammingove kode reda 3 nad GF(3) je koda s kontrolno matriko

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 2 & 0 & 1 & 2 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 2 & 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 2 & 1 & 1 & 2 \end{bmatrix}.$$

Kontrolno matriko H Hammingove kode reda r konstruiramo tako, da si v prvem koraku iz $A = \Sigma^r \backslash \{\mathbf{0}\}$ izberemo neničeln vektor $\boldsymbol{h}_1$ in ga vstavimo v prvi stolpec matrike H. Nato iz množice A odstranimo vseh $(q - 1)$ neničelnih linearnih kombinacij vektorja $\boldsymbol{h}_1$. Na i-tem koraku iz preostanka množice A izberemo vektor $\boldsymbol{h}_i$ in ga vstavimo v i-ti stolpec matrike H. Nato iz preostanka množice A odstranimo vseh $q - 1$ neničelnih linearnih kombinacij $\alpha \boldsymbol{h}_i$, kjer je $\alpha \in \mathbb{F} \backslash \{0\}$. Postopek smo lahko izvedli $n = (q^r - 1)/(q - 1)$-krat, saj je $\sum_{i=1}^{n} n(q - 1) = q^r - 1$, kar je ravno toliko, kot smo imeli na začetku vseh vektorjev v množici A.

Po trditvi 6 imajo Hammingove kode razdaljo vsaj $d = 3$ in zato popravijo eno napako. Če bi imele večjo razdaljo, potem bi pri konstrukciji kontrolne matrike morali iz množice A odstranjevati več vektorjev, posledica tega pa bi bila krajša dolžina kode. Razdalja Hammingovih kod je torej natanko 3.

Če preštejemo vse vektorje, ki so na razdalji največ 1 stran od kodnih besed, jih je $q^k V_q(n, 1) = q^k(1 + n(q - 1))$, saj če spremenimo kodno besedo, ji lahko na n mestih spremenimo po en simbol na eno od $q - 1$ drugačnih vrednosti od prvotne. V enakosti upoštevamo definiciji za n in k in dobimo enakost za Hammingovo oceno, torej so Hammingove kode popolne.

BCH kode so poseben primer cikličnih kod, ki jih bomo spoznali v razdelku 1.5. BCH kode bomo spoznali v 2. poglavju.

Reed-Solomonove kode. Kodne besede so predstavljene s koeficienti polinomov, tako kot pri vseh cikličnih kodah. Prvotno sta Reed in Solomon [78] definirala dekodiranje kot iskanje optimalne interpolacije določenega reda. Na njej temelji tudi hitro dekodiranje teh kod v sezname iz leta 1999, ki si ga bomo ogledali na strani 65. Kmalu po odkritju RS kod so ugotovili, da je to poseben razred BCH kod, torej je za njihovo dekodiranje primeren tudi Petersonov algoritem, ki je bil prvi netrivialni dekodirni algoritem polinomske časovne zahtevnosti in ga bomo podrobno spoznali v razdelku 2.2. Spoznali bomo tudi še dva druga dekodirna algoritma za BCH kode, ki pospešita Petersonovega, to je Berlekampovega in Evklidovega, ki je mnogo preprostejši, a časovno nekaj zahtevnejši od Berlekampovega. Reed-Solomonove kode so edine poznane maksimalno razdaljno separabilne kode, to so tiste, pri katerih dosežemo enakost v Singletonovi oceni.

Poleg naštetih so zelo poznane tudi Reed-Mullerjeve kode, področje aktivnega raziskovanja pa še vedno predstavljajo Hadamardove kode ter algebraično-geometrijske kode. Seveda pa tudi mnoge druge, ki jih nismo našteli. Osnovno znanje o vseh teh kodah najdemo v učbeniku MacWilliams et al. [56].

1.4 Dekodiranje linearnih kod

Poglejmo si najprej proceduro dekodiranja pri kodah, ki odpravijo eno napako v kodni besedi. Če pošljemo kodno besedo $\boldsymbol{c}$ in na drugi strani kanala prejmemo zaporedje $\boldsymbol{r}$, potem lahko zapišemo

$$\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e} .$$

Vektorju $\boldsymbol{e}$ papravimo **vektor napake**. Ker je $H\boldsymbol{c}^T = 0$, je

$$H\boldsymbol{r}^T = H\boldsymbol{e}^T .$$

Vektorju $H\boldsymbol{r}^T$ pravimo **sindrom**. Zanj velja

1. $H\boldsymbol{r}^T = \boldsymbol{0}$, če je $\boldsymbol{e} = \boldsymbol{0}$
2. $H\boldsymbol{r}^T = \alpha \boldsymbol{h}_i$, če je $\boldsymbol{e} = (0 \ldots 0\alpha 0 \ldots 0)$ vektor z neničelno i-to koordinato in $\boldsymbol{h}_i$ i-ti stolpec matrike H

Dekodiranje lahko definiramo z algoritmom, predstavljenim na sliki 1.5.

Dekodiranje za linearne kode, ki odpravijo eno napako	
	Naj bo H kontrolna matrika in $\boldsymbol{r}$ prejeti vektor
1.	Izračunaj $\boldsymbol{s}^T = H\boldsymbol{r}^T$
2.	Če je $\boldsymbol{s}^T = \boldsymbol{0}$, sprejmi $\boldsymbol{r}$ kot poslano kodno besedo
3.	Če je $\boldsymbol{s}^T = \alpha \boldsymbol{h}_i$ za nek $\alpha \in \mathbb{F}$ in nek i, $1 \leq i \leq n$, definiraj $\boldsymbol{e} = (0 \ldots 0\alpha 0 \ldots 0)$ z neničelno i-to koordinato in popravi $\boldsymbol{r}$ v kodno besedo $\boldsymbol{c} = \boldsymbol{r} - \boldsymbol{e}$.
4.	Sicer javi, da je prišlo pri prenosu do več kot ene napake

Slika 1.5: Algoritem za dekodiranje linearnih kod, ki odpravijo eno napako

Vrstico 3 v algoritmu izvedemo tako, da pri vsakem i določimo edinega možnega kandidata za α kot $\alpha = s_j h_{ij}^{-1}$, kjer je j tisti indeks, kjer je $h_{ij} \neq 0$.

Naj opomnimo, da če algoritem uspešno izvede dekodiranje, to je se končna v 2. ali 3. vrstici, še ne pomeni, da je dekodiral pravilno, torej v kodno besedo, ki je bila dejansko poslana. Lahko bi prišlo do takih napak, da bi bil prejeti $\boldsymbol{r}$ zopet kodna beseda, vendar drugačna od poslanega $\boldsymbol{c}$.

Zgornji algoritem je torej učinkovit za linearne kode, ki popravijo eno napako, na primer Hammingove kode. Če pa želimo dekodirati kode, ki popravljajo več kot eno napako, potem pa je potrebno algoritem izpopolniti, kar bomo tudi storili.

Kot vsak vektorski prostor je tudi $V_n(\mathbb{F})$ Abelova grupa, to je komutativna grupa za seštevanje. Podprostor $C \subset V_n(\mathbb{F})$ je podgrupa moči q^k, torej lahko razdelimo grupo $V_n(\mathbb{F})$ na odseke po podgrupi C. Odsek, ki vsebuje element $\boldsymbol{x}$, je definiran kot $\boldsymbol{x} + C = \{\boldsymbol{x} + \boldsymbol{c} \,|\, \boldsymbol{c} \in C\}$. Vemo, da so si odseki disjunktni in ekvipolentni. Označimo jih z $C_0, C_1, \ldots, C_{t-1}$, kjer je $C_0 = C$ in $t = q^{n-k}$. Za vsak i naj bo $\boldsymbol{l}_i$ vektor z minimalno težo v C_i. Poimenujemo ga **vodja odseka** in ni v vsakem odseku enolično določen. Seveda je $\boldsymbol{l}_0 = \boldsymbol{0}$. Sedaj lahko zapišemo vse elemente iz $V_n(\mathbb{F})$ v takoimenovano **standardno tabelo**. V prvi vrstici naj bodo vsi elementi iz C, to je $\boldsymbol{l}_0, \boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_{q^k-1}$. V prvem stolpcu naj bodo vsi vodje odsekov, to je $\boldsymbol{l}_0, \boldsymbol{l}_1, \ldots, \boldsymbol{l}_{t-1}$, v $(i+1)$-vi vrstici in $(j+1)$-vem stolpcu znotraj tabele pa naj bo element $\boldsymbol{l}_i + \boldsymbol{c}_j$. Torej je v i-ti vrstici tabele našteti vsi predstavniki odseka C_i.

Primer. Standardna tabela za binarno $(5,2)$-kodo z generatorsko matriko

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

je torej

	vodje odsekov ↓			
koda C →	00000	10101	01110	11011
	00001	10100	01111	11010
	00010	10111	01100	11001
	00100	10001	01010	11111
	01000	11101	00110	10011
	10000	00101	11110	01011
	11000	01101	10110	00011
	10010	00111	11100	01001

.

▲

Na naslednjem rezultatu bo slonela zasnova dekodirnega algoritma.

Trditev 7. [93, Lemma 3.8] *Naj bo C (n,k)-koda nad $\mathrm{GF}(q)$ s kodnimi besedami $\boldsymbol{c}_0, \boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_{q^k-1}$ in naj bo S standardna tabela z vodji odsekov $\boldsymbol{l}_0, \boldsymbol{l}_1, \ldots, \boldsymbol{l}_{q^{n-k}-1}$ in ostalimi elementi $S_{i+1,j+1} = \boldsymbol{l}_i + \boldsymbol{c}_j$. Potem za vsak i, j in h, $0 \leq i \leq q^{n-k} - 1$, $0 \leq (j,h) \leq q^k - 1$, velja*

$$d(\underbrace{\boldsymbol{l}_i + \boldsymbol{c}_j}_{S_{i+1,j+1}}, \boldsymbol{c}_j) \leq d(\underbrace{\boldsymbol{l}_i + \boldsymbol{c}_j}_{S_{i+1,j+1}}, \boldsymbol{c}_h)\,.$$

Trditev pravi, da je kodna beseda $\boldsymbol{c}_j$ ena od tistih, pri katerih je dosežen $\min_{\boldsymbol{c}\in C} d(S_{i+1,j+1}, \boldsymbol{c})$. Kodna beseda $\boldsymbol{c}_j$ leži na vrhu stolpca, v katerem je $S_{i+1,j+1}$.

Dokaz. Po definiciji teže je

$$\begin{aligned} d(\boldsymbol{l}_i + \boldsymbol{c}_j, \boldsymbol{c}_j) &= w(\boldsymbol{l}_i + \boldsymbol{c}_j - \boldsymbol{c}_j) = w(\boldsymbol{l}_i) \qquad \text{in} \\ d(\boldsymbol{l}_i + \boldsymbol{c}_j, \boldsymbol{c}_h) &= w(\boldsymbol{l}_i + \boldsymbol{c}_j - \boldsymbol{c}_h)\,. \end{aligned}$$

Ker je $C_i = \boldsymbol{l}_i + C$ in ker je $\boldsymbol{c}_j - \boldsymbol{c}_h \in C$, sta $\boldsymbol{l}_i$ in $\boldsymbol{l}_i + \boldsymbol{c}_j - \boldsymbol{c}_h$ oba v odseku C_i in ker je $\boldsymbol{l}_i$ vodja tega odseka, je $w(\boldsymbol{l}_i) \leq w(\boldsymbol{l}_i + \boldsymbol{c}_j - \boldsymbol{c}_h)$. ■

Kodna beseda $\boldsymbol{c}$, pri kateri je dosežen $\min_{\boldsymbol{c}\in C} d(\boldsymbol{r}, \boldsymbol{c})$ pri prejetem zaporedju $\boldsymbol{r}$ je enolično določena natanko tedaj, ko je vodja odseka $\boldsymbol{r} + C$ enolično določen. V katerem primeru je, nam pove naslednja trditev. Ni pa to tudi potrebni pogoj za enoličnost.

Trditev 8. [93, Theorem 3.9] *Naj bo C linearna koda z razdaljo d. Če je $\boldsymbol{x}$ vektor, pri katerem je*

$$w(\boldsymbol{x}) \leq \left\lfloor \frac{d-1}{2} \right\rfloor ,$$

potem je $\boldsymbol{x}$ enoličen element z minimalno težo v svojem odseku in zato edini možni vodja odseka.

Dokaz. Recimo, da v odseku $\boldsymbol{x} + C$ obstaja tudi $\boldsymbol{y}$ z lastnostjo

$$w(\boldsymbol{y}) \leq \left\lfloor \frac{d-1}{2} \right\rfloor .$$

Po trikotniški neenakosti je

$$w(\boldsymbol{x} - \boldsymbol{y}) \leq w(\boldsymbol{x}) + w(\boldsymbol{y}) \leq \left\lfloor \frac{d-1}{2} \right\rfloor + \left\lfloor \frac{d-1}{2} \right\rfloor \leq d - 1\,.$$

Ker je razdalja kode enaka d, je $\boldsymbol{x} - \boldsymbol{y} = \boldsymbol{0}$ in zato $\boldsymbol{y} = \boldsymbol{x}$ ■

Preprost algoritem za dekodiranje linearnih kod s pomočjo standardne tabele S je opisan na sliki 1.6.

Dekodiranje za linearne kode s pomočjo standardne tabele	
	Naj bo S standardna tabela in $\boldsymbol{r}$ prejeti vektor
1.	Poišči $\boldsymbol{r}$ v standardni tabeli S
2.	Popravi $\boldsymbol{r}$ v kodno besedo $\boldsymbol{c}$, ki je na vrhu stolpca, v katerem je $\boldsymbol{r}$

Slika 1.6: Algoritem za dekodiranje linearnih kod s pomočjo standardne tabele

Zadnja shema dekodiranja je popolna dekodirna shema, saj nam algoritem vedno popravi prejeto zaporedje v neko kodno besedo. Seveda lahko algoritem izboljšamo, tako da nam ni potrebno hraniti celotne standardne tabele, ampak le vodje odsekov in njihove sindrome. Kljub temu pa je algoritem eksponentne prostorske zahtevnosti glede na dolžino kode. Da utemeljimo algoritem, moramo dokazati naslednjo trditev.

Trditev 9. [93, Theorem 3.10] *Naj bo H kontrolna matrika linearne kode C. Potem sta vektorja $\boldsymbol{x}$ in $\boldsymbol{y}$ v istem odseku po kodi C natanko tedaj, ko imata isti sindrom, to je $H\boldsymbol{x}^T = H\boldsymbol{y}^T$.*

Dokaz.
($\Rightarrow$) Naj bo $\boldsymbol{x}$ kodna beseda odseka z vodjo $\boldsymbol{l}$. Potem je $\boldsymbol{x} = \boldsymbol{l} + \boldsymbol{c}$ za neko kodno besedo $\boldsymbol{c} \in C$ in

$$H\boldsymbol{x}^T = H(\boldsymbol{l} + \boldsymbol{c})^T = H\boldsymbol{l}^T + H\boldsymbol{c}^T = H\boldsymbol{l}^T\,,$$

saj je $H\boldsymbol{c}^T = \boldsymbol{0}$ za vsako kodno besedo. Torej je sindrom elementov istega odseka konstanten.
($\Leftarrow$) Če je $H\boldsymbol{x}^T = H\boldsymbol{y}^T$, potem je $H\boldsymbol{x}^T - H\boldsymbol{y}^T = \boldsymbol{0}$, oziroma $H(\boldsymbol{x} - \boldsymbol{y})^T = \boldsymbol{0}$, iz česar lahko iz trditve 5 zaključimo, da je $\boldsymbol{x} - \boldsymbol{y}$ kodna beseda. Torej je $\boldsymbol{x} - \boldsymbol{y} \in C$, oziroma $\boldsymbol{x} \in \boldsymbol{y} + C$, kar pomeni, da sta $\boldsymbol{x}$ in $\boldsymbol{y}$ v istem odseku. ■

Dekodiranje za linearne kode s pomočjo sindromov in vodij odsekov standardne tabele	
	Izračunajmo standarno tabelo S, shranimo pa si le vse pare $(\boldsymbol{l}_i, \boldsymbol{s}_i)$, kjer so $\boldsymbol{l}_i$ vodje odsekov, $\boldsymbol{s}_i$ pa njihovi sindromi.
	Naj bo $\boldsymbol{r}$ prejeti vektor.
1.	Vektorju $\boldsymbol{r}$ določi sindrom $\boldsymbol{s}^T = H\boldsymbol{r}^T$
2.	Poišči vodjo odseka $\boldsymbol{l}$ s sindromom s
3.	Popravi $\boldsymbol{r}$ v kodno besedo $\boldsymbol{c} = \boldsymbol{r} - \boldsymbol{l}$

Slika 1.7: Algoritem za dekodiranje linearnih kod s pomočjo sindromov in vodij odsekov standardne tabele

1.5 Ciklične kode

Ciklične kode so poseben primer linearnih kod. Ker je njihova implementacija mnogo lažja in učinkovitejša, imajo zelo velik praktičen pomen. V nadaljevanju bomo spoznali tudi BCH kode, ki so poseben primer cikličnih kod.

Vektorski podprostor S prostora $V_n(\mathbb{F})$ imenujemo **ciklični podprostor**, če za vsak vektor $(a_1a_2\dots a_{n-1}a_n) \in S$ velja, da je tudi njegov ciklični pomik $(a_na_1a_2\dots a_{n-1})$ v S. V tem smislu pravimo tudi, da je linearna koda C **ciklična koda**, če je C ciklični podprostor prostora $V_n(\mathbb{F})$.

V prvem podrazdelku bomo s trditvami od 10 do 12 povezali ciklične podprostore z ideali v kolobarjih polinomov nad $\mathbb{F}$, nato pa bomo s trditvama 13 in 14 podali odgovore na naslednja vprašanja:

1. Kako konstruiramo ciklične podprostore?

2. Ali lahko za dan k vedno konstruiramo k-dimenzionalni ciklični podprostor prostora $V_n(\mathbb{F})$?

3. Koliko je cikličnih podprostorov prostora $V_n(\mathbb{F})$?

4. Kateri vektorji iz cikličnega podprostora S imajo lastnost, da vektor in njegovi ciklični pomiki generirajo celoten S?

V drugem podrazdelku pa bomo predstavili algoritem za kodiranje in dekodiranje ter povedali, zakaj so te kode uporabne tudi za odpravljanje ciklično grozdnih napak.

Konstrukcija cikličnih kod

Naj po Vidav [92] povzamemo nekaj splošnih dejstev iz algebre. Pravimo, da je I **ideal** v komutativnem kolobarju K, če je podkolobar in je zaprt za množenje z vsemi elementi iz K. **Glavni kolobar** je tisti kolobar, v katerem je vsak njegov ideal generiran z enim elementom. Če je $\mathbb{F}$ obseg, potem je $\mathbb{F}[x]$, to je kolobar polinomov nad $\mathbb{F}$, glavni kolobar. Prav tako je tudi faktorski kolobar kolobarja $\mathbb{F}[x]$ po idealu generiranim s polinomom $f(x)$, to je $\mathbb{F}[x]/(f(x))$, glavni ideal.

Pravimo, da je zaporedje $\boldsymbol{v} = (a_0 a_1 \ldots a_{n-1})$ **asociirano** polinomu

$$v(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$$

in obratno, polinom $v(x)$ je asociiran zaporedju $\boldsymbol{v}$. Vzemimo

$$f(x) = x^n - 1$$

in z R označimo množico $R = \{p(x) \in \mathbb{F}[x] \mid \deg p < n\}$. V R definiramo seštevanje in množenje kot običajni operaciji po modulu $f(x)$. Tako definiran R je kolobar in je izomorfen kolobarju $\mathbb{F}[x]/(f(x))$. Asociiranja nam sedaj predstavljajo izomorfizem Abelovih grup $V_n(\mathbb{F})$ in R. Torej je seštevanje vektorjev iz $V_n(\mathbb{F})$ asociirano seštevanju polinomov iz R. Ali je kakšna operacija na vektorjih asociirana množenju polinomov? Da. Ker smo vzeli $f(x) = x^n - 1$, je množenje poljubnega polinoma $v(x)$ s polinomom x asociirano cikličnemu pomiku. Pravkar povedamo lahko posplošimo v naslednji trditvi, ki je osnova za obravnavanje cikličnih kod.

Trditev 10. [93, Theorem 5.3] *Neprazna podmnožica S prostora $V_n(\mathbb{F})$ je ciklični podprostor natanko tedaj, ko je množica polinomov I, asociirana S, ideal v kolobarju R.*

Dokaz.
($\Rightarrow$) Naj bo S ciklični podprostor. Potem je I asociiran S Abelova grupa za seštevanje. Da pokažemo, da je I ideal, moramo pokazati zaprtost za množenje z elementi iz R. Ker je S vektorski prostor, je $\lambda \boldsymbol{v} \in S$ za vsak $\lambda \in \mathbb{F}$ in $\boldsymbol{v} \in S$, torej je tudi $\lambda v(x) \in I$. In ker je S ciklični podprostor, je tudi $xv(x) \in I$, kar lahko induktivno posplošimo v $x^i v(x) \in I$ za vsak i, $0 \leq i \leq n-1$. Če rezultata združimo, dobimo, da je za vsak $\boldsymbol{a} = (a_0 a_1 \ldots a_{n-1})$ tudi $a(x)v(x) = \left(\sum_{i=0}^{n-1} a_i x^i\right) v(x) = \sum_{i=0}^{n-1} \left(a_i x^i v(x)\right) \in I$.
($\Leftarrow$) Predpostavimo sedaj, da je I ideal v R. Ker je I Abelova grupa za seštevanje in ker je množenje s skalarjem v S asociirano množenju s konstantnimi polinomi v I, je S vektorski podprostor v $V_n(\mathbb{F})$. In ker je I zaprt za množenje z vsemi polinomi iz R, torej tudi z x, je S ciklični podprostor, saj je množenje z x asociirano cikličnemu pomiku v S.
■

Poiščimo sedaj kodno besedo, ki s svojimi cikličnimi pomiki generira celoten ciklični podprostor S. Naslednja trditev pove, da je ta kodna beseda enolično določena in asociirana polinomu $g(x)$, generatorju ideala I, ki je asociiran podprostoru S.

Trditev 11. [93, Theorem 5.4, 5.5] *Naj bo I netrivialni ideal v R in naj bo $g(x)$ moničen polinom z najmanjšo stopnjo v I. Zanj velja, da je generator ideala I, da deli $f(x) = x^n - 1$ ter da je enolično določen.*

Dokaz. Naj bo $h(x) \in I$. Z algoritmom za deljenje polinomov dobimo

$$h(x) = q(x)g(x) + r(x)\,,$$

kjer je $\deg r(x) < \deg g(x)$. Ker je I ideal in ker sta $h(x), g(x) \in I$, je tudi $r(x) \in I$. Ker ima $g(x)$ po predpostavki najmanjšo stopnjo izmed neničelnih polinomov iz I, je $r(x) = 0$. Torej je vsak $h(x) \in I$ oblike $h(x) = q(x)g(x)$ za nek $q(x) \in R$, kar pomeni, da je $g(x)$ generator ideala I.

Za $f(x)$ in $g(x)$ uporabimo algoritem za deljenje polinomov

$$f(x) = q(x)g(x) + r(x)\,,$$

kjer je $\deg r(x) < \deg g(x)$. Torej je $r(x) = -q(x)g(x) \bmod f(x)$ in ker je $g(x)$ generator za I, je tudi $r(x) \in I$. Po predpostavkah o stopnjah polinomov $g(x)$ in $r(x)$ je $r(x) = 0$, torej je $f(x)$ deljiv z $g(x)$.

Recimo, da imata v idealu I monična polinoma $g(x)$ in $h(x)$ minimalno stopnjo. Po prej pokazanem je $g(x)$ generator I, kjer se operacije izvajajo po modulu $f(x)$, torej je $h(x) = a(x)g(x) - q(x)f(x)$ za neka polinoma $a(x)$ in $q(x)$. Ker $g(x)$ deli $f(x)$, je desna stran enačbe deljiva z $g(x)$, torej je $h(x) = b(x)g(x)$ za nek polinom $b(x)$. Ker sta $h(x)$ in $g(x)$ enakih stopenj, je $b(x)$ stopnje 1 in ker sta oba monična, je tudi $b(x)$ moničen, torej je $b(x) = 1$ in zato $h(x) = g(x)$. ■

V naslednjih dveh trditvah bomo odgovorili na vprašanje, koliko je cikličnih podprostorov.

Trditev 12. [93, Theorem 5.6] *Naj bo $h(x)$ monični delitelj polinoma $f(x) = x^n - 1$. Potem je $h(x)$ edini polinom z najmanjšo stopnjo izmed vseh moničnih polinomov v idealu $I = \{a(x)h(x) \mid a(x) \in R\}$ kolobarja R.*

Dokaz. Naj bo $g(x)$ monični polinom z najmanjšo stopnjo v idealu I. Po prejšnji trditvi je $g(x)$ generator za I in delitelj polinoma $f(x)$. Torej je $h(x) = a(x)g(x) - q(x)f(x)$ za neka polinoma $a(x)$ in $q(x)$. Ker $g(x)$ deli $f(x)$, je $h(x)$ deljiv z $g(x)$ in ker $h(x)$ deli $f(x)$, je $g(x)$ deljiv s $h(x)$. Ker sta oba polinoma monična, sta enaka. Po prejšnji trditvi je $h(x)$ edini polinom z najmanjšo stopnjo v idealu I. ■

Trditev 13. [93, Theorem 5.7] *Obstaja bijektivna korespondenca med cikličnimi podprostori prostora $V_n(\mathbb{F})$ in moničnimi polinomi $g(x) \in \mathbb{F}[x]$, ki delijo $f(x) = x^n - 1$ nad obsegom $\mathbb{F}$.*

Dokaz. Naj bo $\mathcal{S}$ množica vseh cikličnih podprostorov prostora $V_n(\mathbb{F})$, $\mathcal{R}$ množica vseh idealov v kolobarju R in G množica vseh moničnih deliteljev polinoma $f(x) = x^n - 1$. Ker nam trditev 12 zagotavlja obstoj injektivne preslikave iz G v $\mathcal{R}$ in trditev 11 obstoj injektivne preslikave iz $\mathcal{R}$ v G, obstaja bijektivna korespondenca med G in $\mathcal{R}$. Po trditvi 10 imamo bijektivno korespondenco tudi med $\mathcal{R}$ in $\mathcal{S}$ in dokaz je končan. ■

Tehnike faktorizacije polinoma $f(x) = x^n - 1$ nad obsegom $\mathbb{F} = \mathrm{GF}(q)$ so opisane v Vanstone in Oorschot [93, Chapter 5.8]. Recimo, da ima $f(x)$ faktorizacijo $f(x) =$

$(p_1(x))^{n_1}(p_2(x))^{n_2}\cdots(p_t(x))^{n_t}$, kjer so $p_i(x)$ nad $\mathbb{F}$ nerazcepni monični polinomi. Potem $V_n(\mathbb{F})$ vsebuje $(n_1+1)(n_2+1)\cdots(n_t+1)$ cikličnih podprostorov.

Poglejmo, kako izbor moničnega delitelja polinoma x^n-1 vpliva na obliko in lastnosti kode. Z notacijo vektorjev ne bomo striktni, pisali jih bomo kot polinome. Rekli bomo, da polinom $g(x)$ **generira** ciklični podprostor S, če je generator ideala asociiranega podprostoru S.

Trditev 14. [93, Theorem 5.9] *Naj bo $g(x)$ stopnje $n-k$ monični delitelj polinoma x^n-1 nad $\mathbb{F}$. Potem je $g(x)$ generator cikličnega podprostora S prostora $V_n(\mathbb{F})$ in*

$$B=\{g(x),xg(x),\ldots x^{k-1}g(x)\}$$

je baza podprostora S, kar pomeni, da je dimenzija podprostora S enaka k. Velja tudi

$$S=\{a(x)g(x)\mid \deg a(x)<k\}\,.$$

Dokaz. Pokazati moramo, da so vektorji iz B linearno neodvisni in da napenjajo cel prostor S. Naj bo

$$\sum_{i=0}^{k-1}\lambda_i x^i g(x)=0\,,\qquad \lambda_i\in\mathbb{F}\,,$$

poljubna linearna kombinacija vektorjev iz B. Polinom $\sum_{i=0}^{k-1}\lambda_i x^i g(x)$ ima koeficient pred x^{n-1} enak λ_{k-1}, torej je $\lambda_{k-1}=0$. Podobno dobimo $\lambda_i=0$ za vse i, $0\le i\le k-1$, torej je B linearno neodvisna.

Vzemimo poljuben $h(x)\in S$. Ker je $g(x)$ generator ideala asociiranega S, je $h(x)=a(x)g(x)$ za nek $a(x)\in V_n(\mathbb{F})$. Recimo, da je $a(x)$ stopnje manjše od k, torej $a(x)=\sum_{i=0}^{k-1}\lambda_i x^i$. Potem je $h(x)=\left(\sum_{i=0}^{k-1}\lambda_i x^i\right)g(x)=\sum_{i=0}^{k-1}\lambda_i\left(x^i g(x)\right)$ linearna kombinacija elementov iz B. Kaj pa če je $a(x)$ stopnje večje od k? Potem je $\deg a(x)g(x)\ge n$ in obstaja $q(x)\in\mathbb{F}[x]$ ter $r(x)$ stopnje $\deg r(x)\le n-1$, tako da je $a(x)g(x)=q(x)(x^n-1)+r(x)$. Ker $g(x)$ deli x^n-1, deli tudi $r(x)$, torej je $r(x)=a'(x)g(x)$ za nek polinom $a'(x)$ in velja $\deg a'(x)\le k-1$. Ker je sedaj $a(x)g(x)=q(x)(x^n-1)+a'(x)g(x)$, nam polinoma $a(x)g(x)$ in $a'(x)g(x)$ predstavljata isti vektor v S, za $a'(x)g(x)$ pa smo že pokazali, da je linearna kombinacija elementov iz B. ■

Primer. Recimo, da želimo imeti ciklično $(15,9)$-kodo nad obsegom $\mathbb{Z}_3$. Definirali jo bomo s polinomom $g(x)$ stopnje $15-9=6$, ki je delitelj $x^{15}-1$. Ker je

$$x^{15}-1=(2+x)^3(1+x+x^2+x^3+x^4)^3\,,$$

vzamemo $g(x)=(2+x)^2(1+x+x^2+x^3+x^4)$. ▲

Polinomu $h(x)=h_0+h_1x+\cdots+h_kx^k$ definirajmo njegov **recipročni polinom** kot $h_R(x)=h_k+h_{k-1}x+\cdots+h_0x^k$, torej je $h_R(x)=x^kh(1/x)$. V nadaljevanju bomo pokazali, da je ciklični kodi dualna koda tudi ciklična koda. To je lep rezultat, saj bomo s pomočjo polinoma, ki generira dualno kodo, določili kontrolno matriko.

Trditev 15. [93, Theorem 5.10] *Naj bo C ciklična (n,k)-koda, generirana s polinomom $g(x)$. Potem je njej dualna koda $C^\perp$ generirana s polinomom $h_R(x)$, kjer je $h(x)=(x^n-1)/g(x)$.*

Dokaz. Naj bo C' ciklična koda, generirana s polinomom $h(x)$. Potem je za vsak $\boldsymbol{c}_1 \in C$ in $\boldsymbol{c}_2 \in C'$

$$c_1(x)c_2(x) = a_1(x)g(x)a_2(x)h(x) \equiv 0 \pmod{x^n - 1}$$

za neka polinoma $a_1(x)$ in $a_2(x)$. Če vzamemo $\boldsymbol{a} = (a_0a_1 \ldots a_{n-1}) \in C$ in $\boldsymbol{b} = (b_0b_1 \ldots b_{n-1}) \in C'$, potem obstajajo koeficienti $c_i \in \mathbb{F}$, taki da je

$$a(x)b(x) = \left(\sum_{i=0}^{n-1} a_i x^i\right)\left(\sum_{i=0}^{n-1} b_i x^i\right) \equiv \sum_{i=0}^{n-1} c_i x^i \pmod{x^n - 1},$$

in po prej pokazanem so vsi $c_i = 0$, torej tudi

$$c_0 = a_0b_0 + a_1b_{n-1} + a_2b_{n-2} + \cdots + a_{n-1}b_1 = 0\,.$$

Če enačbo zapišemo v smislu notranjega produkta, potem je

$$\boldsymbol{a} \cdot \bar{\boldsymbol{b}} = 0$$

za vsak $\boldsymbol{a} \in C$ in vsak $\bar{\boldsymbol{b}} \in C''$, kjer je koda C'' dobljena kot $C'' = C'P$, kjer je P linearna transformacija, definirana s permutacijsko matriko P, za katero velja $\boldsymbol{b}P = \bar{\boldsymbol{b}}$ pri poljubno izbranem $\boldsymbol{b} \in C'$ in na prejšnji način definiranem $\bar{\boldsymbol{b}} = (b_0b_{n-1}b_{n-2} \ldots b_1)$. Torej je koda C'' linearna in tudi ciklična (mešanje stolpcev ohrani cikličnost podprostora). Generatorska matrika kode C'' je potem $H'' = H'P$, saj nam izomorfizem vektorskih prostorov slika bazo v bazo. To pomeni, da je baza prostora C'' določena z množico polinomov $\{x^{n-k}h_R(x), \ldots, x^2h_R(x), xh_R(x)\}$. Tudi ciklični pomik za eno mesto levo je izomorfizem vektorskih prostorov, torej je baza C'' lahko določena s polinomi $\{x^{n-k-1}h_R(x), \ldots, xh_R(x), h_R(x)\}$. Ker velja tudi, da je $h_R(x)$ delitelj polinoma $f(x) = x^n - 1$, saj je

$$f(x) = x^n f(1/x) = x^{n-k}x^k g(1/x)h(1/x) = g_R(x)h_R(x)\,,$$

je $h_R(x)$ generator cikličnega prostora C'', to je $C^{\perp}$. ■

Naj opomnimo pri poteku dokaza, da kljub temu, da za vsak $\boldsymbol{c}_1 \in C$ in $\boldsymbol{c}_2 \in C'$, velja $c_1(x)c_2(x) \equiv 0 \pmod{x^n - 1}$, to še ne pomeni, da sta si C in C' dualni kodi, saj množenje v R, to je faktorskem kolobarju polinomov reduciranih po modulu $x^n - 1$, ni asociirano notranjemu produktu v $V_n(\mathbb{F})$.

Kodiranje in dekodiranje cikličnih kod

Naj bo $g(x)$ generator ciklične (n,k)-kode C nad $\mathbb{F}$. Iz prejšnjega podrazdelka vemo, da je generator $g(x)$ stopnje $n-k$, ter da je celotna koda enaka $C = \{a(x)g(x) \mid \deg a(x) < k\}$. Zaradi te lastnosti je za prostor sporočil povsem naravno vzeti $\{a(x) \mid \deg a(x) < k\}$ ter definirati kodiranje kot

$$a(x) \mapsto a(x)g(x)\,.$$

Če želimo imeti kodne besede sestavljene na način, da so sporočilu dodani kontrolni simboli, potem moramo generatorsko matriko imeti v standardni obliki. Tu bomo naredili izjemo in bomo konstruirali standardno matriko oblike $G = [R\, I_k]$, torej bomo kontrolne simbole zapisali pred sporočilo. Z algoritmom deljenja izračunamo

$$x^{n-k+i} = q_i(x)g(x) + r_i(x)\,, \qquad i = 0, 1, \ldots, k-1\,,$$

kar pomeni, da tudi polinom $-r_i(x) + x^{n-k+i} = q_i(x)g(x)$, $0 \le i \le k-1$, predstavlja kodno besedo. Če te besede zložimo v matriko G, dobimo $G = [R\ I_k]$, kjer je

$$R = \begin{bmatrix} -r_0(x) \\ -r_1(x) \\ \vdots \\ -r_{k-1}(x) \end{bmatrix}.$$

Ker je matrika G ranga k, je tudi generatorska matrika za C. Kontrolna matrika je potem $H = [I_{n-k}\ \ -R^T]$. Kodiranje sporočila $\boldsymbol{a}$ na način $a(x) \mapsto a(x)g(x)$ je gotovo hitrejše od reševanja sistema $H[\boldsymbol{a}'\boldsymbol{a}]^T = 0$, kjer so $\boldsymbol{a}'$ kontrolni biti. Kaj pa dekodiranje? Poglejmo si najprej, katerim vektorjem so asociirani sindromi.

Trditev 16. [93, Theorem 5.11] *Naj bo ciklična (n,k)-koda C nad obsegom $\mathbb{F}$ generirana s polinomom $g(x)$ in naj bo kontrolna matrika H definirana kot prej. Naj bo vektor $\boldsymbol{s}$ sindrom vektorja $\boldsymbol{r}$. Potem je $s(x)$ ostanek pri deljenju polinoma $r(x)$ s polinomom $g(x)$.*

Dokaz. Naj bo $\boldsymbol{r} = (a_0 a_1 \dots a_{n-1})$. Ker je $H = [I_{n-k}\ \ -R^T]$, je

$$H^T = \begin{bmatrix} I_{n-k} \\ -R \end{bmatrix} = [1, x, \dots, x^{n-k-1}, r_0(x), r_1(x), \dots, r_{k-1}(x)]^T$$

in ker je sindrom definiran kot $\boldsymbol{s} = H\boldsymbol{r}^T$, je

$$\begin{aligned} s(x) &= a_0 + a_1x + \cdots + a_{n-k-1}x^{n-k-1} + \\ &\quad + a_{n-k}r_0(x) + \cdots + a_{n-1}r_{k-1}(x) = \\ &= a_0 + a_1x + \cdots + a_{n-k-1}x^{n-k-1} + \\ &\quad + a_{n-k}(x^{n-k} - q_0(x)g(x)) + \cdots + a_{n-1}(x^{n-1} - q_{k-1}(x)g(x)) = \\ &= r(x) - (a_{n-k}q_0(x) + \cdots + a_{n-1}q_{k-1}(x))g(x) = \\ &= r(x) - b(x)g(x)\,, \end{aligned}$$

kjer smo definirali $b(x) = a_{n-k}q_0(x) + \cdots + a_{n-1}q_{k-1}(x)$. Torej je $r(x) = b(x)g(x) + s(x)$. Ker je

$$s(x) = a_0 + a_1x + \cdots + a_{n-k-1}x^{n-k-1} + a_{n-k}r_0(x) + \cdots + a_{n-1}r_{k-1}(x)$$

in ker so vsi $r_i(x)$ stopnje manjše od $n-k$, je tudi $\deg s(x) \le n-k-1$. Zato sta $b(x)$ in $s(x)$ res polinoma, ki jih dobimo z algoritmom deljenja $r(x)$ z $g(x)$. ■

Trditev 17. [93, Theorem 5.12] *Naj bo ciklična (n,k)-koda C nad obsegom $\mathbb{F}$ generirana s polinomom $g(x)$ in naj ima $r(x)$ sindrom $s(x) = \sum_{i=0}^{n-k-1} s_i x^i$. Potem je sindrom polinoma $x\,r(x)$ enak*

$$\begin{array}{cl} x\,s(x) & ,\ \textit{če je}\ \deg s(x) < n-k-1\ ,\ \textit{in} \\ x\,s(x) - s_{n-k-1}g(x) & ,\ \textit{če je}\ \deg s(x) = n-k-1\,. \end{array}$$

Dokaz. Po prejšnji trditvi sta $r(x)$ in $s(x)$ povezana z enačbo

$$r(x) = b(x)g(x) + s(x)\,.$$

Če je $\deg s(x) < n-k-1$ in prejšnjo enačbo množimo z x, dobimo

$$x\,r(x) = x\,b(x)g(x) + x\,s(x)$$

in dobljena enačba predstavlja enačbo, ki bi jo dobili z algoritmom deljenja $x\,r(x)$ z $g(x)$, saj je ostanek $x\,s(x)$ stopnje manjše od $\deg g(x) = n-k$. Po prejšnji trditvi je $x\,s(x)$ sindrom od $x\,r(x)$. Če pa je $\deg s(x) = n-k-1$, potem nam

$$x\,r(x) = x\,b(x)g(x) + x\,s(x) = (x\,b(x) + s_{n-k-1})g(x) - (s_{n-k-1}g(x) - x\,s(x))$$

zopet predstavlja enačbo, ki bi jo dobili z algoritmom deljenja $x\,r(x)$ z $g(x)$, saj je ostanek $s_{n-k-1}g(x) - x\,s(x)$ zopet stopnje manjše od $\deg g(x) = n-k$. Po prejšnji trditvi je $s_{n-k-1}g(x) - x\,s(x)$ sindrom od $x\,r(x)$. ■

Pokažimo sedaj, kako poteka dekodiranje.

Lema 18. *Naj ima (n,k)-koda C nad obsegom $\mathbb{F}$ kontrolno matriko H oblike $H = [I_{n-k}\,A]$. Naj bo razdalja kode enaka $d = 2t+1$, $\boldsymbol{c}$ poslana kodna beseda, $\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e}$ prejeti vektor in $\boldsymbol{s} = H\boldsymbol{r}^T$ njegov sindrom. Če je $w(\boldsymbol{e}) \le t$ in $w(\boldsymbol{s}) \le t$, potem je $\boldsymbol{e} = (\boldsymbol{s}^T, \boldsymbol{0})$.*

Dokaz. Vemo, da je $\boldsymbol{s} = H\boldsymbol{r}^T = H(\boldsymbol{c}+\boldsymbol{e})^T = H\boldsymbol{e}^T$. Če vzamemo $\bar{\boldsymbol{e}} = (\boldsymbol{s}^T, \boldsymbol{0}_{1\times k})$, potem zaradi standardne oblike matrike H velja $H\bar{\boldsymbol{e}}^T = \boldsymbol{s}$. Torej imata $\bar{\boldsymbol{e}}$ in $\boldsymbol{e}$ isti sindrom in po trditvi 9 sta v istem odseku. Ker je $w(\boldsymbol{s}) \le t$, je tudi $w(\bar{\boldsymbol{e}}) \le t$ in po trditvi 8 je $\bar{\boldsymbol{e}} = \boldsymbol{e}$ enolični vodja svojega odseka. Torej je $\boldsymbol{e} = (\boldsymbol{s}^T, \boldsymbol{0}_{1\times k})$. ■

V zaporedju bomo k ciklično zaporednih simbolov poimenovali **ciklični stik** dolžine k. Tako bomo rekli, da zaporedje (11763111) vsebuje ciklični stik enk dolžine 5.

Naj bo bo C ciklična koda z generatorjem $g(x)$ in naj veljajo predpostavke iz leme. Privzemimo, da je $\boldsymbol{r}$ prejeti vektor in $\boldsymbol{e}$ vektor napake s težo $w(\boldsymbol{e}) \le t$ in da vsebuje ciklični stik najmanj k ničel. S pomočjo trditev 16 in 17 ter zadnje leme poskušajmo določiti $\boldsymbol{e}$ in postopek zapisati v obliki algoritma.

Označimo z R desni ciklični pomik. Naj bo $\boldsymbol{r}_i = R^i\boldsymbol{r}$, $0 \le i \le n-1$. Potem je tudi pripadajoč vektor napake enak $\boldsymbol{e}_i = R^i\boldsymbol{e}$, pripadajoč sindrom $\boldsymbol{s}_i$ pa lahko s pomočjo trditve 17 izračunamo iz že izračunanega $\boldsymbol{s}_{i-1}$, kjer smo $\boldsymbol{s}_0$ izračunali z algoritmom deljenja $r(x)$ z $g(x)$, glej trditev 16. Če pri kakšnem i ugotovimo, da je $w(\boldsymbol{s}_i) \le t$, potem je po lemi $\boldsymbol{e}_i = (\boldsymbol{s}_i^T, \boldsymbol{0})$, saj smo predpostavili $w(\boldsymbol{e}) \le t$. Našli smo napako $\boldsymbol{e}_i$, torej lahko določimo $\boldsymbol{e} = \boldsymbol{e}_0 = R^{n-i}\boldsymbol{e}_i$. Postopku pravimo **sledenje napaki** (ang. error trapping), saj ni nujno, da jo odkrijemo. Opisan je na sliki 1.8. Njegova prednost je, da nam ni potrebno hraniti vseh sindromov.

Ciklični grozd je vektor, ki ima vse neničelne komponente v cikličnem stiku dolžine t. Ciklične kode so uporabne tudi za odpravljanje **cikličnih grozdnih napak**. Algoritem sledenja napaki za popravljanje le teh se od prvotnega razlikuje le v vrstici 3, ki je potem enaka

3.	**if** $s_i(x)$ je ciklični grozd dolžine $\le t$ **then begin**

Več o tem in o dokazu pravilnosti algoritma si bralec lahko prebere v Vanstone et al. [93, Chapter 5.7].

Dekodiranje za ciklične kode (sledenje napaki)

Predpostavke:
    C je ciklična (n,k)-koda, ki popravi t napak
    $g(x)$ je generator kode C
    $s_i(x)$ je oznaka za sindrom od $x^i r(x)$
    $e(x)$ vektor napake
    $w(\boldsymbol{e}) \leq t$ in $\boldsymbol{e}$ vsebuje ciklični stik vsaj k ničel
izračunaj sindrom $s_0(x)$ z algoritmom deljenja $r(x) = q(x)g(x) + s_0(x)$
for $i = 0$ **to** $n-1$ **do begin**
    if $w(s_i(x)) \leq t$ **then begin**
        $\boldsymbol{e} = R^{n-i}(\boldsymbol{s}_i, \mathbf{0})$
        $\boldsymbol{c} = \boldsymbol{r} - \boldsymbol{e}$
        return $\boldsymbol{c}$
    end
    if $\deg s_{i-1}(x) < n-k-1$ **then** $s_i(x) = x\, s_{i-1}(x)$
    else $s_i(x) = x\, s_{i-1}(x) - s_{i-1,n-k-1} g(x)$
end
return 'napake ni možno izslediti'

Slika 1.8: Algoritem za dekodiranje cikličnih kod (sledenje napaki)

Poglavje 2

BCH kode

Pomemben parameter pri izbiri kod je njihova razdalja, o kateri nismo pri cikličnih kodah še nič povedali. S pomočjo trditve 6 iz prvega poglavja lahko določimo razdaljo poljubne linearne kode, v Vanstone et al. [93, str. 210] imamo opisan postopek za določitev spodnje meje za razdaljo cikličnih kod, vendar pa bi si že pri konstrukciji ciklične kode, to je izboru generatorskega polinoma, želeli okvirno vedeti razdaljo kode. To lahko storimo v primeru BCH kod, katere bomo definirali v prvem razdelku, kjer bomo podali tudi dokaz veljavnosti konstrukcijske razdalje. Ta razred cikličnih kod je leta 1959 odkril A. Hocquenghem, leta 1960 pa neodvisno od njega tudi Bose in Ray-Chaudhuri [14]. Te kode se sedaj imenujejo po začetnicah njihovih priimkov, to so BCH kode. Prvotno so bile te kode predstavljene nad obsegi s karakteristiko 2, leta 1961 pa sta jih Gorenstein in Zierler [33] posplošila, to je konstruirala nad poljubnim končnim obsegom. Pri BCH kodah konstruiramo generatorski polinom $g(x)$ na način, da iz oblike ničel v njegovem razpadnem obsegu lahko določimo spodnjo mejo za razdaljo. Poseben primer teh kod so t.i. Reed-Solomonove (RS) kode [78], pri katerih je spodnja meja za razdaljo tudi dejanska razdalja. Razred BCH kod je zelo pomemben še iz dveh razlogov. Odlikuje jih relativno preprosta dekodirna shema ter širok izbor parov ustreznih parametrov n in k za konstruiranje BCH (n, k)-kode.

V drugem razdelku bomo opisali učinkovit postopek dekodiranja BCH kod. Izpeljali bomo takoimenovano ključno enačbo, ki jo lahko rešimo na več načinov. Najučinkovitejši je Berlekampov [7] algoritem, ki ga bomo opisali v tretjem razdelku. V zadnjem razdelku pa bomo bolj podrobno predstavili RS kode s poudarkom na Guruswami-Sudanovem algoritmu za dekodiranje v sezname iz leta 1999.

2.1 BCH kode in določitev razdalje

Pravimo, da monični polinom $g(x) \in \mathrm{GF}(q)[x]$ **razpade** v obsegu $\mathrm{GF}(q^m)$, to je razširitvi obsega $\mathrm{GF}(q)$, če ga lahko razstavimo do linearnih faktorjev

$$g(x) = (x - \alpha_1)(x - \alpha_2) \cdots (x - \alpha_n)\,,$$

kjer so $\alpha_i \in \mathrm{GF}(q^m)$. Po moči najmanjši obseg $\mathrm{GF}(q^m)$, v katerem razpade polinom $g(x)$, imenujemo **razpadni obseg** polinoma $g(x)$. Tu bomo definicijo vzeli bolj ohlapno, zahtevo *najmanjši* bomo izpustili. Vemo, da so si končni obsegi iste moči izomorfni, kakor tudi da razpadni obseg polinoma vedno obstaja, glej Vidav [92, poglavje 9.4].

Lema 19. *Naj bo $g(x)$ generator ciklične (n,k)-kode C nad obsegom* GF(q) *in naj ima ničle $\alpha_1, \alpha_2, \ldots, \alpha_{n-k}$ v nekem razpadnem obsegu* GF(q^m)*. Potem je polinom $t(x) \in C$ natanko tedaj, ko je $t(\alpha_i) = 0$ za vsak i, $1 \le i \le n-k$.*

Dokaz. Naj bo $m_i(x) \in \mathrm{GF}(q)[x]$ minimalni polinom za α_i, $1 \le i \le n-k$. Ker je $g(\alpha_i) = 0$, je $g(x)$ deljiv z $m_i(x)$, to je ena od lastnosti minimalnih polinomov. Po trditvi 14 vemo, da $t(x)$ predstavlja kodno besedo natanko tedaj, ko je $t(x)$ deljiv z $g(x)$. Torej je deljiv tudi z vsemi $m_i(x)$ in zato je $t(\alpha_i) = 0$ za vsak i, $1 \le i \le n-k$. In obratno, če je $t(\alpha_i) = 0$ za vsak i, $1 \le i \le n-k$, potem je $t(x)$ deljiv z vsemi $m_i(x)$, kar pomeni, da je $t(x)$ deljiv z $g(x)$. ■

V naslednji trditvi si bomo ogledali matriko H, ki ima podobno lastnost kot kontrolna matrika, vendar pa ta matrika ne vsebuje elementov iz GF(q), ampak iz GF(q^m).

Trditev 20. [93, Theorem 6.1] *Naj bo $g(x)$ generator ciklične (n,k)-kode C nad obsegom* GF(q) *in naj ima ničle $\alpha_1, \alpha_2, \ldots, \alpha_{n-k}$ v nekem razpadnem obsegu* GF(q^m)*. Definirajmo $H \in \mathrm{GF}(q^m)^{(n-k)\times n}$ kot*

$$H = \begin{bmatrix} \alpha_1^0 & \alpha_1^1 & \cdots & \alpha_1^{n-1} \\ \alpha_2^0 & \alpha_2^1 & \cdots & \alpha_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n-k}^0 & \alpha_{n-k}^1 & \cdots & \alpha_{n-k}^{n-1} \end{bmatrix}.$$

Potem je $\boldsymbol{c} \in C$ natanko tedaj, ko je $H\boldsymbol{c}^T = 0$.

Trditev velja tudi, če matriko H definiramo s podmnožico $\{\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_t}\}$, $t \le n-k$, množice ničel $\{\alpha_1, \alpha_2, \ldots, \alpha_{n-k}\}$ polinoma $g(x)$, če za njo velja $\{m_{\alpha_{i_1}}(x), \ldots, m_{\alpha_{i_t}}(x)\} = \{m_{\alpha_1}(x), \ldots, m_{\alpha_{n-k}}(x)\}$, kjer je $m_\alpha(x)$ oznaka za minimalni polinom elementa α.

Dokaz. Naj bo $\boldsymbol{c} = (c_0 c_1 \ldots c_{n-1})$. Potem je

$$\begin{aligned} H\boldsymbol{c}^T = \boldsymbol{0} \quad &\Leftrightarrow \quad \sum_{j=0}^{n-1} \alpha_i^j c_j = 0\,, \quad 1 \le i \le n-k \quad \Leftrightarrow \\ &\Leftrightarrow \quad c(\alpha_i) = 0\,, \quad 1 \le i \le n-k\,, \end{aligned}$$

to pa je po prejšnji lemi natanko tedaj, ko je $\boldsymbol{c} \in C$. ■

Naj bo β primitivni n-ti koren enote ($\beta^n = 1$ in $\beta^i \neq 1$ za vsak $i = 1, \ldots, n-1$) iz obsega GF(q^m), ki je razširitev obsega $\mathbb{F} = \mathrm{GF}(q)$. Z $m_i(x) \in \mathbb{F}[x]$ označimo minimalni polinom elementa β^i, z lcm pa najmanjši skupni večkratnik (ang. least common multiple). Ciklična (n,k)-koda C nad obsegom $\mathbb{F} = \mathrm{GF}(q)$ z generatorskim polinomom $g(x) \in \mathbb{F}[x]$ se imenuje **BCH koda s konstrukcijsko razdaljo**[1] δ, če je

$$g(x) = \mathrm{lcm}\{m_i(x) \,|\, a \le i \le a+\delta-2\} \in \mathbb{F}[x]$$

in množica ničel polinoma $g(x)$ vsebuje $\delta - 1$ različnih elementov $\beta^a, \beta^{a+1}, \ldots, \beta^{a+\delta-2}$, kjer je a neko celo število. Ker je β primitivni koren enote, velja tudi $(\beta^i)^n = 1$ za vsak i, kar pomeni, da $m_i(x)$ deli $x^n - 1$ za vsak i. Ker so si različni minimalni polinomi tuji, $g(x)$ deli $x^n - 1$ in zato so BCH kode res ciklične kode. Če je $n = q^m - 1$, potem

[1] Parameter δ za konstrukcijsko razdaljo ima povsem drug pomen kot parameter δ za relativno razdaljo iz prvega poglavja. Lahko bi uvedli drugačne oznake, a bomo raje ostali konsistentni z literaturo.

je obstoj primitivnega n-tega korena enote zagotovljen, saj je $\mathrm{GF}(q^m)^*$ vedno ciklična grupa. Tako dobljeni BCH kodi pravimo **primitivna** BCH koda, v posebnem primeru, ko je $n = q - 1$, je to Reed-Solomonova koda. Če je $a = 1$ pa rečemo kodi **preprosta** BCH koda. S pomočjo prejšnje trditve in in formule

$$D = \prod_{0 \leq j < i \leq t} (x_i - x_j) \tag{2.1}$$

za izračun **Vandermondove determinante**

$$D = \begin{vmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_t \\ x_0^2 & x_1^2 & \cdots & x_t^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_0^t & x_1^t & \cdots & x_t^t \end{vmatrix}$$

bomo določili spodnjo mejo za razdaljo BCH kod.

Trditev 21. [93, Theorem 6.2] *Za BCH (n, k)-kodo C nad $\mathrm{GF}(q)$ s konstrukcijsko razdaljo δ velja, da ima razdaljo $d \geq \delta$. Za dimenzijo k pa velja $k \geq n - (\delta - 1)m$.*

Dokaz. Naj bo dolžina kode C enaka n in naj bo $g(x)$ njen generator. Naj bo α primitivni n-ti koren enote v $\mathrm{GF}(q^m)$ in naj množica ničel polinoma $g(x)$ vsebuje $\delta - 1$ elementov oblike $\alpha^a, \alpha^{a+1}, \ldots, \alpha^{a+\delta-2}$ za nek $a \in \mathbb{Z}$. Naj bo

$$H = \begin{bmatrix} (\alpha^a)^0 & (\alpha^a)^1 & \cdots & (\alpha^a)^{n-1} \\ (\alpha^{a+1})^0 & (\alpha^{a+1})^1 & \cdots & (\alpha^{a+1})^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ (\alpha^{a+\delta-2})^0 & (\alpha^{a+\delta-2})^1 & \cdots & (\alpha^{a+\delta-2})^{n-1} \end{bmatrix}.$$

Če bi pisali $x_i = \alpha^{a+i}$ in bi iz matrike H vzeli $\delta - 1$ stolpcev, recimo stolpce z indeksi $t_1, t_2 \ldots, t_{\delta-1}$, ter tvorili determinanto, iz katere bi iz i-tega stolpca izpostavili α^{at_i}, bi dobili Vandermondovo determinanto. Ker so vsi x_i paroma različni, je ta determinanta različna od 0, kar pomeni, da je vsak izbor $\delta - 1$ stolpcev iz H linearno neodvisen nad $\mathrm{GF}(q^m)$.

Po trditvi 20 je $H\boldsymbol{c}^T = 0$ natanko tedaj, ko $\boldsymbol{c} \in C$. Ker je $\mathrm{GF}(q^m)$ vektorski prostor dimenzije m nad obsegom $\mathrm{GF}(q)$, lahko vsak element v matriki $H \in \mathrm{GF}(q^m)^{(\delta-1)\times n}$ zamenjamo s stolpcem dolžine m, ki predstavlja njegovo vektorsko reprezentacijo. Označimo tako dobljeno matriko s $\bar{H} \in \mathrm{GF}(q)^{((\delta-1)m)\times n}$. Za vsak $\boldsymbol{c} \in \mathrm{GF}(q)^n$ pokažimo, da je $\bar{H}\boldsymbol{c}^T = 0$ natanko tedaj, ko je $H\boldsymbol{c}^T = 0$, to pa vemo, da je natanko tedaj, ko $\boldsymbol{c} \in C$.

Naj bo $\{\beta_1, \ldots, \beta_m\}$ baza vektorskega $\mathrm{GF}(q^m)$ nad obsegom $\mathrm{GF}(q)$ in vzemimo poljubno vrstico $[h_1, \ldots, h_n]$ iz matrike H. Vsak h_i lahko potem zapišemo kot linearno kombinacijo nad $\mathrm{GF}(q)$ baznih elementov, torej $h_i = \sum_{j=1}^m h_{ij}\beta_j$. Za vsak $\boldsymbol{c} \in \mathrm{GF}(q)^n$ je potem

$$\sum_{i=1}^n h_i c_i = 0 \Leftrightarrow \sum_{i=1}^n \sum_{j=1}^m h_{ij}\beta_j c_i = 0 \Leftrightarrow \sum_{j=1}^m \Big(\sum_{i=1}^n h_{ij} c_i\Big)\beta_j = 0 \Leftrightarrow \sum_{i=1}^n h_{ij} c_i = 0 \text{ za } \forall j\,.$$

Iz matrike $\bar{H}$ vzamemo maksimalno linearno neodvisno množico vrstic ter jih zapišemo v matriko $\bar{\bar{H}}$, ki tako postane kontrolna matrika kode C. Ker je poljubnih $\delta - 1$ stolpcev matrike H linearno neodvisnih nad GF(q^m) in zato tudi nad GF(q), je poljubnih $\delta - 1$ stolpcev matrike $\bar{\bar{H}}$ linearno neodvisnih nad GF(q), sicer bi slednja odvisnost implicirala prvo. Po trditvi 6 je razdalja kode C enaka vsaj δ.

Kontrolna matrika $\bar{\bar{H}}$ ima največ $(\delta - 1)m$ vrstic, torej je $n - k \leq (\delta - 1)m$. ■

Primer. Konstruirajmo binarno BCH kodo z dolžino bloka 15 in s konstrukcijsko razdaljo 7. Potrebujemo primitivni 15-ti koren enote α nad $\mathbb{Z}_2 = \mathrm{GF}(2)$. Če vzamemo obseg GF($2^4$) in določimo vse ciklotomične odseke števila 2 po modulu 15, dobimo

$$\begin{aligned}
C_0 &= \{0\}\,,\\
C_1 &= \{1, 2, 4, 8\}\,,\\
C_3 &= \{3, 6, 9, 12\}\,,\\
C_5 &= \{5, 10\}\,,\\
C_7 &= \{7, 14, 13, 11\}\,.
\end{aligned}$$

Iz teorije o končnih odsekih vemo, da imata α^i in α^j enak minimalni polinom natanko tedaj, ko sta i in j v istem ciklotomičnem odseku, ter da je ta polinom enak

$$m_i = \prod_{j \in C_i} (x - \alpha^j)\,,$$

torej stopnje enake moči odseka. Če vzamemo

$$g(x) = m_1(x) m_3(x) m_5(x)\,,$$

potem $g(x)$ generira BCH $(15, 5)$-kodo, saj je $\deg g(x) = 4 + 4 + 2 = 15 - 5$. Množica ničel polinoma $g(x)$ je enaka

$$\begin{aligned}
R &= \{\alpha^i \mid i \in C_1 \cup C_3 \cup C_5\} =\\
&= \{\alpha^i \mid i = 1, 2, 3, 4, 5, 6, 8, 9, 10, 12\}
\end{aligned}$$

in vsebuje 6 zaporednih potenc elementa α, to so $\alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5$ in α^6. Torej je po zadnji trditvi razdalja kode enaka vsaj 7.

Če vzamemo obseg GF(2^4) generiran z α, ki je ničla $f(x) = 1 + x + x^4$, potem so

$$\begin{aligned}
m_1(x) &= 1 + x + x^4\,,\\
m_3(x) &= 1 + x + x^2 + x^3 + x^4\,,\\
m_5(x) &= 1 + x + x^2\,,\\
g(x) &= 1 + x + x^2 + x^4 + x^5 + x^8 + x^{10}\,.
\end{aligned}$$

Ker nam $g(x)$ predstavlja kodno besedo $\boldsymbol{g} = (11101100101\,0000)$ s težo 7, je po trditvi 4 razdalja kode enaka 7. Ni pa težko najti primerov BCH kod, kjer je dejanska razdalja strogo večja od konstrukcijske. ▲

V Vanstone in Oorschot [93, str. 210] lahko najdemo algoritem za določanje spodnje meje za razdaljo cikličnih kod. Algoritem je smiselno uporabiti skupaj z BCH kodami, pri katerih s konstrukcijo dosežemo določeno razdaljo kode, nato pa s to metodo poiščemo boljšo, če obstaja, spodnjo mejo za razdaljo kode.

V učbeniku [56, Chapter 9.5] je pokazano, da so BCH kode asimptotično slabe. Kljub temu pa so pri n-jih do nekaj tisoč med boljšimi kodami. Dimenzijo k binarne kode se da oceniti za konstanto bolje kot v trditvi 21 za poljuben obseg. Če je C binarna BCH (n,k)-koda, definirana s pomočjo primitivnega elementa iz obsega GF(q^m) ter s konstrukcijsko razdaljo $\delta = 2t+1$, potem je $k \geq n - mt$, glej [56, Chapter 9.1, Theorem 1]. V oceni dosežemo enakost, če velja $2t-1 < 2^{\lceil m/2 \rceil}+1$, torej za δ manjše od približno $\sqrt{n}$, glej [56, Chapter 9.3, Corollary 8].

2.2 Dekodiranje BCH kod

Za dekodiranje cikličnih kod smo v razdelku 1.5 predstavili algoritem sledenja napaki, ki je temeljil na predpostavki, da v napaki $\boldsymbol{e}$ obstaja ciklični stik vsaj k ničel. Za BCH kode ta predpostavka ni potrebna. Konstruirali bomo preprost algoritem za dekodiraje. Naj beseda preprost, ki jo uporabljajo mnogi avtorji v literaturi, bralca ne zavede. Implementacija je preprosta s stališča hardverske oziroma softverske konstrukcije, saj nam v spominu ni potrebno hraniti standardne tabele ali tabele sindromov, izvaja se le preračunavanje elementov v GF(q^m). Opis in dokaz algoritma pa zahtevata nekaj več matematičnega znanja in precej zbranosti.

Prvotno so bile BCH kode binarne in Peterson [74] je leta 1960 podal prvi algoritem polinomske časovne zahtevnosti za njihovo dekodiranje. Gorenstein in Zierler [33] sta poleg BCH kod posplošila tudi dekodirni algoritem za BCH kode nad poljubnim končnim obsegom. Ta algoritem bomo na kratko opisali v tem razdelku. Njegove ideje je kasneje uporabil Berlekamp in leta 1967 v San Remu[2] predstavil t.i. ključno enačbo in pokazal, da lahko dekodiranje BCH kod pospešimo s hitrim iterativnim algoritmom za iskanje rešitve te enačbe. Massey pa je opazil, da bi se dalo ta algoritem uporabiti tudi za sintezo LFSR, o čemer bomo pisali v 5. poglavju. Berlekamp je svoj algoritem leta 1968 objavil v [7], kjer je predstavil tudi Masseyevo idejo. Seveda pa jo je podrobno opisal tudi Massey v [60]. Nekateri avtorji sedaj tudi prvotni Petersonov algoritem za dekodiranje BCH kod imenujejo Berlekampov ali pa celo Berlekamp-Masseyev (BM) algoritem.

Sugiyama et al. [90] so leta 1975 pokazali, da je Berlekampov algoritem za reševanje ključne enačbe ekvivalenten predčasno prekinjenemu razširjenemu Evklidovemu algoritmu. To je asimptotično za velike δ seveda res, vendar pa bomo pokazali, da se vrednost Berlekampovega algoritma pokaže v primerih, ko se pri prenosu pojavi ℓ napak, $\ell \ll \delta$, oziroma pri LFSR sintezi zaporedja z majhno linearno zahtevnostjo. V teh primerih se BM algoritem konča za red hitreje od Evklidovega, to je v času $O(\ell\,\delta)$ namesto $O(\delta^2)$.

Petersonov algoritem za dekodiranje BCH kod

Predstavili bomo Petersonov [74] algoritem za dekodiranje BCH kod, katerega največja vrednost je izpeljava t. i. ključne enačbe, glej tudi Peterson [75, Chapter 9.4]. Le-to je Berlekamp [7] rešil na hiter in zelo eleganten način z algoritmom, katerega se še danes citira v mnogih znanstvenih člankih. Peterson je ključno enačbo reševal kot sistem linearnih enačb. V njegovem algoritmu moramo najprej ugotoviti rang sistema enačb velikosti $t \times t$, kjer je $t = \lfloor (\delta-1)/2 \rfloor$. Za ta korak porabimo $O(t^2)$ operacij in prav tu, oziroma samo tu, je Berlekampov algoritem hitrejši. Njegova časovna zahtevnost je $O(\ell\, t)$, kjer je ℓ rang sistema. Tu nam ℓ predstavlja število napak povzročenih pri prenosu, oziroma linearno zahtevnost zaporedja koeficientov polinoma $S(z)$ iz ključne enačbe.

[2]International Symposium on Information Theory, San Remo, Italija

Naj bo $g(x)$ generatorski polinom BCH kode C nad obsegom $\mathbb{F} = \mathrm{GF}(q)$ in naj bo δ konstrukcijska razdalja ter n dolžina kode. Naj bo α primitivni n-ti koren enote v $\mathbb{F}$ ter $\alpha^a, \alpha^{a+1}, \ldots, \alpha^{a+\delta-2}$ v eksponentih zaporedne ničle polinoma $g(x)$. Z $m_i(x)$ označimo minimalni polinom elementa α^i, torej je

$$g(x) = \operatorname{lcm}\{m_i(x) \mid a \leq i \leq a+\delta-2\}\,.$$

Vektorje iz $V_n(\mathbb{F})$ bomo zapisovali kar v obliki njim asociiranim polinomom. Privzemimo, da smo poslali kodno besedo $c(x)$ ter da je pri prenosu prišlo do napake $e(x)$ s težo ℓ, $\ell \leq t = \lfloor(\delta-1)/2\rfloor$. Z dekodiranjem želimo iz prejete kodne besede $r(x) = c(x) + e(x)$ ugotoviti napako $e(x)$.

Najprej izračunajmo sindrom $s(x)$ prejetega elementa $r(x)$, ki je po trditvi 16 enak[3] ostanku od $r(x)$ pri deljenju z $g(x)$,

$$r(x) = h(x)g(x) + s(x)\,.$$

Če je $s(x) = 0$, potem je $r(x) \in C$, sicer pa je $e(x) \neq 0$. Naj bo $S_{i+1} = s(\alpha^{a+i})$ za $0 \leq i \leq \delta-2$. Ker je $r(x) = c(x) + e(x)$ in ker so α^{a+i}, $0 \leq i \leq \delta-2$, ničle generatorskega polinoma $g(x)$, iz leme 19 sledi

$$S_{i+1} = s(\alpha^{a+i}) = r(\alpha^{a+i}) = e(\alpha^{a+i})\,, \qquad 0 \leq i \leq \delta-2\,.$$

Označimo sedaj položaje neničelnih koeficientov polinoma napake z a_j, $0 \leq j \leq \ell-1$, in jih poimenujmo **položaji napake**. Z algoritmom jih bomo izračunali iz **lokatorjev napak** $u_j = \alpha^{a_j}$, $0 \leq j \leq \ell-1$, šele nato pa bomo določili tudi **velikosti napak**, ki jih označimo z $y_j \neq 0$. Polinom napake je torej enak

$$e(x) = \sum_{j=0}^{\ell-1} y_j x^{a_j}\,.$$

Če pogledamo njegove vrednosti pri $x = \alpha^{a+i}$, $0 \leq i \leq \delta-2$, in upoštevamo definiciji za S_{i+1} in u_j, dobimo sistem $\delta-1$ nelinearnih enačb

$$S_{i+1} = \sum_{j=0}^{\ell-1} y_j u_j^{a+i}\,, \qquad 0 \leq i \leq \delta-2\,, \tag{2.2}$$

z neničelnimi neznankami y_j in u_j, $0 \leq j \leq \ell-1$. Peterson je potem definiral

$$\sigma(z) = \prod_{j=0}^{\ell-1}(1-u_j z)\,, \tag{2.3}$$

ki ga je Berlekamp poimenoval **polinom lokatorja napake**. Peterson je obravnaval (2.3) kot enačbo s parametri $\sigma_1, \ldots, \sigma_\ell$, kjer so σ_i koeficienti polinoma $\sigma(z) = 1 + \sum_{j=1}^{\ell} \sigma_\ell z^\ell$. Vrednost enačbe (2.3) je pogledal pri $z = u_i^{-1}$, $0 \leq i \leq \ell-1$, in jo na obeh straneh pomnožil z $y_i u_i^{a+j}$, $\ell \leq j \leq \delta-2$. Dobljeno enačbo $\sigma(u_i^{-1}) y_i u_i^{a+j} = 0$ je seštel po vseh i, $0 \leq i \leq \ell-1$, ter z upoštevanjem definicije za S_{j-k+1} dobil sistem linearnih enačb $\sum_{k=0}^{\ell} S_{j-k+1}\sigma_k = 0$, $\ell \leq j \leq \delta-2$, z neznankami $\sigma_1, \ldots, \sigma_\ell$. Ta sistem bomo predstavili

[3] Naj opomnimo, da je sklep: "Ker je $c(x)$ oblike $a(x)g(x)$, je $e(x) = s(x)$," napačen, saj to, da je $c(x)$ oblike $a(x)g(x)$ še ne pomeni, da je $c(x) = h(x)g(x)$.

v naslednji trditvi kot relacijo med polinomom $\sigma(z)$, **polinomom sindroma** $S(z)$, ki ga definiramo kot

$$S(z) = \sum_{j=0}^{\delta-2} S_{j+1} z^j \,, \tag{2.4}$$

ter **polinomom izračuna napake** $\omega(z)$, ki pa ga definiramo kot

$$\omega(z) = \sum_{i=0}^{\ell-1} y_i u_i^a \prod_{\substack{j=0 \\ j\neq i}}^{\ell-1} (1 - u_j z) \,. \tag{2.5}$$

Polinome $\sigma(z)$, $\omega(z)$ in $S(z)$ smo vzeli kot polinome spremenljivke z namesto x, s čemer smo poudarili, da polinomi ne predstavljajo vektorjev iz $V_n(\mathbb{F})$. Polinom izračuna napake je dobil takšno ime zato, ker lahko z njim na 2. način v 4. koraku Petersonovega algoritma s slike 2.1 izračunamo vrednosti napak y_k, $0 \leq k \leq \ell - 1$.

Trditev 22. [8, Chapter 10.1] *Polinome $S(z)$, $\sigma(z)$ in $\omega(z)$, definirane v (2.3), (2.4) in (2.5), povezuje enačba*

$$S(z)\sigma(z) \equiv \omega(z) \pmod{z^{\delta-1}} \,. \tag{2.6}$$

Berlekamp jo je poimenoval **ključna enačba** *za dekodiranje BCH kod.*

Zgornja trditev nam pove, da so koeficienti produkta $S(z)\sigma(z)$ pri potencah $z^\ell, z^{\ell+1}, \ldots, z^{\delta-2}$ enaki 0.

Dokaz. Iz definicij omenjenih polinomov in (2.4) dobimo

$$\begin{aligned} S(z)\sigma(z) &= \sum_{k=0}^{\delta-2} S_{k+1} z^k \sigma(z) = \sum_{k=0}^{\delta-2} \sum_{i=0}^{\ell-1} y_i u_i^{a+k} z^k \prod_{j=0}^{\ell-1} (1 - u_j z) = \\ &= \sum_{i=0}^{\ell-1} y_i u_i^a \sum_{k=0}^{\delta-2} (u_i z)^k \prod_{j=0}^{\ell-1} (1 - u_j z) \,. \end{aligned}$$

Upoštevamo še vsoto geometrijske vrste $(1-u_i z) \sum_{k=0}^{\delta-2} (u_i z)^k = 1-(u_i z)^{\delta-1} \equiv 1 \pmod{z^{\delta-1}}$ in dobimo

$$S(z)\sigma(z) \equiv \sum_{i=0}^{\ell-1} y_i u_i^a \prod_{\substack{j=0 \\ j\neq i}}^{\ell-1} (1 - u_j z) \equiv \omega(z) \pmod{z^{\delta-1}} \,,$$

kar smo želeli pokazati. ■

Lema 23. *Za dan polinom sindroma $S(z)$ je poljuben par $(\omega'(z), \sigma'(z))$ rešitve ključne enačbe (2.6), za katerega velja $\deg \omega'(z) \leq \deg \omega(z) = \ell - 1$ in $\deg \sigma'(z) \leq \deg \sigma(z) = \ell$, oblike*

$$\omega'(z) = \lambda\, \omega(z) \qquad \text{in} \qquad \sigma'(z) = \lambda\, \sigma(z)$$

za neko konstanto $\lambda \in \mathbb{F}$.

Dokaz. Ker sta $\omega'(z)$ in $\sigma'(z)$ rešitvi ključne enačbe, velja

$$S(z)\sigma'(z) \equiv \omega'(z) \pmod{z^{\delta-1}} \,. \tag{2.7}$$

Če enačbo (2.6) pomnožimo z $\sigma'(z)$, enačbo (2.7) pa z $\sigma(z)$, dobimo

$$\omega(z)\sigma'(z) \equiv \omega'(z)\sigma(z) \pmod{z^{\delta-1}}.$$

Ker je $\deg(\omega(z)\sigma'(z)) \leq 2\ell - 1 \leq \delta - 2$ in prav tako tudi $\deg(\omega'(z)\sigma(z)) \leq \delta - 2$, je

$$\omega(z)\sigma'(z) = \omega'(z)\sigma(z). \tag{2.8}$$

Iz definicij polinomov $\omega(z)$ in $\sigma(z)$ je preprosto videti, da je $\gcd(\omega(z), \sigma(z)) = 1$, saj polinom $\sigma(z)$ razpade na linearne faktorje, ki delijo vse, razen enega sumanda polinoma $\omega(z)$. Iz enačbe (2.8) potem sledi, da $\omega(z)$ deli $\omega'(z)$. Ker velja $\deg \omega'(z) \leq \deg \omega(z)$, je $\omega'(z) = \lambda\, \omega(z)$ za neko konstanto $\lambda \in \mathbb{F}$. Če to relacijo vstavimo v enačbo (2.8), dobimo še $\sigma'(z) = \lambda\, \sigma(z)$. ■

Celoten opis postopka lahko predstavimo z algoritmom na sliki 2.1, primer dekodiranja pri konkretni BCH kodi pa je dan na strani 49.

Petersonov algoritem za dekodiranje BCH kod
Predpostavke: C je BCH koda nad $\mathbb{F} = \mathrm{GF}(q)$ z dolžino n in konstrukcijsko razdaljo δ, $\alpha \in \mathbb{F}$ je primitivni n-ti koren enote, $g(x)$ je generator kode C in $\{\alpha^a, \alpha^{a+1}, \ldots, \alpha^{a+\delta-2}\}$ je podmnožica njegovih ničel, $m_i(x)$ je minimalni polinom elementa α^i, $g(x) = \mathrm{lcm}\{m_i(x) \mid a \le i \le a+\delta-2\}$, $r(x)$ je prejeta kodna beseda, $e(x)$ pa vektor napake s težo ℓ, $e(x) = \sum_{j=0}^{\ell-1} y_j x^{a_j}$, $y_j \neq 0$. Ostale oznake: S_{j+1} sindrom elementa α^{a+j}, $0 \le j \le \delta-2$, $u_j = \alpha^{a_j}$, $0 \le j \le \ell-1$, $S(z) = \sum_{j=0}^{\delta-2} S_{j+1} z^j$, $\sigma(z) = \prod_{j=0}^{\ell-1}(1-u_j z)$, $\omega(z) = \sum_{i=0}^{\ell-1} y_i u_i^a \prod_{\substack{j=0\\ j\neq i}}^{\ell-1}(1-u_j z)$.
1. Določi sindrom $s(x)$ od $r(x)$ z algoritmom deljenja $r(x)$ z $g(x)$: $s(x) = r(x) - h(x)g(x)$. Če je $s(x) = 0$, potem $r(x)$ dekodiraj v $c(x) = r(x)$ in `STOP`, sicer določi $S_{i+1} = s(\alpha^{a+i})$, $0 \le i \le \delta-2$.
2. Poišči $\sigma'(z)$, ki reši ključno enačbo $S(z)\sigma(z) \equiv \omega(z) \pmod{z^{\delta-1}}$. 1. način: reši kot sistem linearnih enačb (Peterson), 2. način: uporabi razširjeni Evklidov algoritem, ki ga predčasno prekineš (Sugiyama et al., opis na strani 47), 3. način: uporabi Berlekampov algoritem (najhitrejši, opisan v razdelku 2.3).
3. Določi $\ell = \deg(\sigma'(z))$. Poišči ničle polinoma $\sigma'(z)$ in jih zapiši v seznam $\bar{U}$: $U = \{(\bar{u})^{-1} \mid \bar{u} \in \bar{U}\}$, $A = \{\log_\alpha u \mid u \in U\}$, v A imamo shranjene lokacije napak a_j, $0 \le j \le \ell-1$. Uredi elemente v seznamu A po velikosti in skladno z njim tudi U, $A = (a_0, \ldots, a_{\ell-1})$ in $U = (u_0, \ldots, u_{\ell-1})$.
4. Če je koda binarna, so vse velikosti napak enake 1, sicer določi velikosti napak kot: 1. način: rešitev linearnega sistema enačb (2.10), 2. način: $y_k = \omega_R(u_k) / \big(u_k^a \prod_{\substack{j=0\\ j\neq k}}^{\ell-1}(u_k - u_j)\big)$. Dekodiraj $r(x)$ v $c(x) = r(x) - e(x)$, kjer je $e(x) = \sum_{j=0}^{\ell-1} y_j x^{a_j}$.

Slika 2.1: Petersonov algoritem za dekodiranje BCH kod

Opombe in predlogi za hitro izvajanje posameznih korakov algoritma s slike 2.1:

1. korak. Ker izračunamo S_{i+1} le za i, $0 \le i \le \delta-2$, bomo potrošili manj operacij, če bomo izračunali S_{i+1} kot vrednost ostanka pri deljenju $r(x)$ z $m_{a+i}(x)$, saj je $m_{a+i}(x)$ minimalni polinom za α^{a+i}, $0 \le i \le \delta-2$. Torej z algoritmom deljenja določimo $r(x) = s_i(x) = q_i(x)m_{a+i}(x)$ in od tod še S_{i+1}.

2. korak – reševanje ključne enačbe :

1. način. Peterson je imel izpeljan ključni enačbi (2.6) ekvivalenten sistem enačb

$$\begin{bmatrix} S_{t+1} & S_t & \dots & S_1 \\ S_{t+2} & S_{t+1} & \dots & S_2 \\ \vdots & \vdots & \ddots & \vdots \\ S_{2t} & S_{2t-1} & \dots & S_t \end{bmatrix} \begin{bmatrix} 1 \\ \sigma_1 \\ \vdots \\ \sigma_t \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.9}$$

iz katerega je vzel največje število zaporednih linearno neodvisnih enačb. Teh enačb je ℓ, kjer nam ℓ predstavlja število napak, povzročenih pri prenosu. Potem je postavil $\sigma_{\ell+1}, \dots, \sigma_t$ na vrednost 0 in iz sistema določil $\sigma_1, \dots, \sigma_\ell$. Nato je določil lokatorje u_j, $0 \leq j \leq \ell - 1$, kot ničle polinoma lokatorja napake in potem še velikosti napak $y_0, \dots, y_{\ell-1}$. Če je koda binarna, t. j. $\mathbb{F} = \mathbb{Z}_2$, so velikosti napak enake 1, sicer pa jih poiščemo kot rešitev linearnega sistema enačb $S_{i+1} = \sum_{j=0}^{\ell-1} y_j u_j^{a+i}$, $0 \leq i \leq \ell - 1$, t. j.

$$\begin{bmatrix} u_0^a & u_1^a & \dots & u_{\ell-1}^a \\ u_0^{a+1} & u_1^{a+1} & \dots & u_{\ell-1}^{a+1} \\ \vdots & \vdots & \ddots & \vdots \\ u_0^{a+\ell-1} & u_1^{a+\ell-1} & \dots & u_{\ell-1}^{a+\ell-1} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{\ell-1} \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_\ell \end{bmatrix}. \tag{2.10}$$

2. način. Sugiyama et al. [90] so pokazali, da lahko določimo polinoma $\omega(z)$ in $\sigma(z)$ iz Berlekampove ključne enačbe (2.6) s predčasno prekinjenim razširjenim Evklidovim algoritmom, ki je zapisan na sliki 2.2. Pravzaprav nas zanima le $\sigma(z)$, saj lahko, iz njegovih ničel ugotovimo lokatorje napak u_j, iz njih pa lokacije napak. Postopek razširjenega Evklidovega algoritma je enak postopku običajnega Evklidovega algoritma za izračun največjega skupnega delitelja $G(z)$ polinomov $a(z)$ in $b(z)$ z razliko, da sproti izračunamo še polinoma $s(z)$ in $t(z)$, ki rešita enačbo

$$s(z)a(z) + t(z)b(z) = G(z)\,.$$

Algoritem deluje, saj se z indukcijo ni težko prepričati o veljavnosti enačbe

$$s_i(z)a(z) + t_i(z)b(z) = r_i(z)\,. \tag{2.11}$$

Vidimo tudi, da $\deg r_i(z)$ monotono strogo pada. Če je $\deg a(z) \geq \deg b(z)$, potem za vsak i, $i \geq 1$, in r_i iz algoritma velja tudi (indukcija)

$$\deg t_i(z) \leq \deg a(z) - \deg r_{i-1}(z)\,. \tag{2.12}$$

Razširjeni Evklidov algoritem bomo pri dekodiranju uporabili pri podatkih $a(z) = z^{\delta-1}$ in $b(z) = S(z)$, vendar pa bomo algoritem prekinili pred običajnim koncem, saj ne želimo izračunati največji skupni delitelj polinomov $z^{\delta-1}$ in $S(z)$, ampak poiskati tako rešitev $\sigma'(z)$ ključne enačbe, da je $\deg \sigma'(z) \leq (\delta-1)/2$. Ker velja enačba (2.11), bomo vzeli $\sigma'(z) = t_i$ pri tistem i, kjer je $\deg r_{i-1}(z) \geq (\delta-1)/2$ in $\deg r_i(z) \leq (\delta-1)/2$. Z upoštevanjem (2.12) dobimo

$$\deg \sigma'(z) = \deg t_i(z) \leq \deg(z^{\delta-1}) - \deg r_{i-1}(z) = \frac{\delta-1}{2}\,.$$

Razširjeni Evklidov algoritem za izračun največjega skupnega delitelja $G(z)$ polinomov $a(z)$ in $b(z)$
1. $s_0(z) = 1$, $t_0(z) = 0$, $r_0(z) = a(z)$ $s_1(z) = 0$, $t_1(z) = 1$, $r_1(z) = b(z)$ $i = 2$ 2. **while** $r_{i-1}(z) \neq 0$ **do begin** deli polinom $r_{i-2}(z)$ s polinomom $r_{i-1}(z)$ in določi $q_i(z)$ in $r_i(z)$ $r_{i-2}(z) = q_i(z) r_{i-1}(z) + r_i(z)$ $s_i(z) = s_{i-2}(z) - q_i(z) s_{i-1}(z)$ $t_i(z) = t_{i-2}(z) - q_i(z) t_{i-1}(z)$ $i = i + 1$ **end** 3. $s(z) = s_{i-2}(z)$ $t(z) = t_{i-2}(z)$ $G(z) = r_{i-2}(z)$

Slika 2.2: Razširjeni Evklidov algoritem

3. korak. Obstajajo tehnike iskanja ničel polinomov v končnih obsegih, nekaj jih je opisanih v Vanstone in Oorschot [93, Chapter 6.5]. Chien [17] pa je za iskanje ničel pri hardverskem dekodiranju BCH kod predlagal postopek z vzporednim računanjem. Ker so kandidati za ničle polinoma $\sigma'(z)$ le inverzi možnih lokatorjev napak, to je $\alpha^{-1} = \alpha^{n-1}, \alpha^{-2} = \alpha^{n-2}, \ldots, \alpha^{-n} = 1$, je za preizkus vseh možnosti, kjer računamo s pomočjo registrov, smiseln vzporedni algoritem, predstavljen na sliki 2.3.

Chienovo iskanje ničel polinoma $\sigma'(z)$
Predpostavke: $\sigma'(z) = \sigma'_0 + \sigma'_1 z + \cdots + \sigma'_\ell z^\ell$ r_i vrednost v i-tem registru, $1 \leq i \leq \ell$ 1. inicializacija registrov (vzporedno): $r_i = \sigma'_i$ 2. **for** $k = 1$ **to** n **do begin** $r_i = \alpha^i r_i$, $1 \leq i \leq \ell$ (vzporedno) $\sigma'(\alpha^k) = \sigma'_0 + \sum_{i=1}^{\ell} r_i$ **if** $\sigma'(\alpha^k) = 0$ **then** $n - k$ je lokacija napake pri dekodiranju **end**

Slika 2.3: Chienov algoritem za iskanje ničel polinoma $\sigma'(z)$ (Chien Search)

4. korak. Za rešitev sistema enačb (2.10) lahko uporabimo metodo Gaussove eliminacije. Seveda pa je za večje ℓ hitrejši Berlekampov izračun. Naj bo $\sigma_\ell = (-1)^\ell \prod_{j=0}^{\ell-1} u_j$ vodilni koeficient polinoma $\sigma(z)$ in σ'_l vodilni koeficient polinoma $\sigma'(z)$. Potem je

$$\sigma(z) = \sigma_\ell \cdot (\sigma'_\ell)^{-1} \cdot \sigma'(z)\,.$$

Iz ključne enačbe izračunamo

$$\omega(z) = \sigma(z) S(z) \bmod z^{\delta-1}\,.$$

Sedaj upoštevamo, da je v naslednji vsoti neničelen le sumand pri $i = k$:

$$\omega(u_k^{-1}) = \sum_{i=0}^{\ell-1} y_i u_i^a \prod_{\substack{j=0 \\ j\neq i}}^{\ell-1} (1 - u_j u_k^{-1}) = y_k u_k^a \prod_{\substack{j=0 \\ j\neq k}}^{\ell-1} (1 - u_j u_k^{-1}),$$

kar pomeni

$$y_k = \frac{\omega(u_k^{-1})}{u_k^a \prod_{\substack{j=0 \\ j\neq k}}^{\ell-1} (1 - u_j u_k^{-1})} = \frac{u_k^{-(\ell-1)} \omega_R(u_k)}{u_k^{-(\ell-1)} u_k^a \prod_{\substack{j=0 \\ j\neq k}}^{\ell-1} (u_k - u_j)} = \frac{\omega_R(u_k)}{u_k^a \prod_{\substack{j=0 \\ j\neq k}}^{\ell-1} (u_k - u_j)}.$$

Za preračunavanje z elementi v končnem obsegu lahko uporabljamo tabelo za izračun diskretnega logaritma z osnovo pri izbranem primitivnem elementu $\beta \in \mathbb{F}$. Tako lahko pretvarjamo med vektorskim zapisom v $\mathbb{F}$ ter zapisom v obliki β^i. Nasprotni element izračunamo s pomočjo vektorskega zapisa, množenje preprosto izvedemo kot $\beta^i\beta^j = \beta^{i+j}$, seštevanje pa s pomočjo Zechove tabele logaritmov, kjer hranimo pare $[i, \log_\beta(1 + \beta^i)]$, torej za $i \leq j$ velja

$$\beta^i + \beta^j = \beta^i(1 + \beta^{j-i}) = \beta^i \beta^{\log_\beta(1+\beta^{j-i})} = \beta^{i+\log_\beta(1+\beta^{j-i})}.$$

Najbolj nazorno bomo to videli pri izračunu koeficientov polinoma sindroma v naslednjem primeru.

Primer. Konstruirajmo BCH kodo C dolžine 13 s konstrukcijsko razdaljo $\delta = 5$ nad GF(3). Naj bo obseg $\mathbb{F} = \mathrm{GF}(3^3)$ generiran s primitivnim elementom α, ki je ničla nerazcepnega polinoma $f(x) = 1 + 2x^2 + x^3$. Potem je $\beta = \alpha^2$ primitivni 13-ti koren enote v $\mathbb{F}$. Ne zanimajo nas vsi ciklotomični odseki, ampak le tisti, ki vsebujejo β^i, $0 \leq i \leq 12$, ki jih računamo po modulu $(3^3 - 1)/2 = 13$. Zapišimo jih in poleg njih še ustrezne minimalne polinome potenc iz odseka:

$$\begin{array}{ll}
C_0 = \{0\}, & m_0(x) = x - 1, \\
C_1 = \{1, 3, 9\}, & m_1(x) = 2 + 2x + 2x^2 + x^3, \\
C_2 = \{2, 5, 6\}, & m_2(x) = 2 + 2x + x^3, \\
C_4 = \{4, 10, 12\}, & m_4(x) = 2 + x + 2x^2 + x^3, \\
C_7 = \{7, 8, 11\}, & m_7(x) = 2 + x^2 + x^3.
\end{array}$$

Preračunavanja v $\mathbb{F}$ bomo delali s pomočjo tabele, v kateri imamo shranjene vektorske zapise elementov, njihov običajni logaritem ter Zechov algoritem.

α^i	i	$z(i)$
(000)	∞	0
(100)	0	13
(010)	1	18
(001)	2	7
(201)	3	2
(221)	4	12
(220)	5	14
(022)	6	21
(101)	7	3

α^i	i	$z(i)$
(211)	8	19
(222)	9	6
(121)	10	4
(210)	11	1
(021)	12	10
(200)	13	∞
(020)	14	24
(002)	15	16
(102)	16	20

α^i	i	$z(i)$
(112)	17	23
(110)	18	11
(011)	19	22
(202)	20	15
(122)	21	9
(111)	22	8
(212)	23	25
(120)	24	5
(012)	25	17

Kot generator kode C lahko vzamemo

$$g(x) = m_0(x)m_1(x)m_2(x) = 2 + 2x + x^4 + 2x^5 + x^6 + x^7,$$

kar pomeni, da je $\{\beta^0, \beta^1, \beta^2, \beta^3\}$ podmnožica ničel, torej je $a = 0$. Predpostavimo, da je dekoder prejel $\boldsymbol{r} = (220\,021\,110\,211\,0)$. Opravimo dekodiranje po korakih
1.korak

$$\begin{aligned}
r(x) &= q_0(x)m_0(x) + 1\,, & s_0(x) &= 1\,,\\
r(x) &= q_1(x)m_1(x) + (1 + 2x + x^2)\,, & s_1(x) &= 1 + 2x + x^2\,,\\
r(x) &= q_2(x)m_2(x) + (1 + 2x)\,, & s_2(x) &= 1 + 2x\,,
\end{aligned}$$

$$\begin{aligned}
S_1 &= r(\beta^0) = s_0(\beta^0) = 1\,,\\
S_2 &= r(\beta^1) = s_1(\beta^1) = 1 + 2\beta + \beta^2 = 1 + 2\alpha^2 + \alpha^4 = 1 + \alpha^{13}\alpha^2 + \alpha^4 =\\
&= 1 + \alpha^{15} + \alpha^4 = \alpha^{16} + \alpha^4 = \alpha^4(1 + \alpha^{12}) = \alpha^4\alpha^{10} = \alpha^{14}\,,\\
S_3 &= r(\beta^2) = s_2(\beta^2) = 1 + 2\beta^2 = 1 + 2\alpha^4 = \alpha^{23}\,,\\
S_4 &= r(\beta^3) = s_1(\beta^3) = s_1(\beta)^3 = (\alpha^{14})^3 = \alpha^{42} = \alpha^{16}\,,
\end{aligned}$$

$$S(z) = 1 + \alpha^{14}z + \alpha^{23}z^2 + \alpha^{16}z^3\,.$$

2.korak
Z razširjenim Evklidovim algoritmom rešimo ključno enačbo $S(z)\sigma(z) \equiv \omega(z) \pmod{z^4}$.

i	$s_i(z)$	$t_i(z)$	$r_i(z)$	$\deg(r_i(z))$
0	1	0	$(0,0,0,0,1)$	4
1	0	1	$(1,\alpha^{14},\alpha^{23},\alpha^{16})$	3
2	1	$(\alpha^{17},\alpha^{23})$	$(\alpha^{17},\alpha^{16},\alpha^{13})$	2
3	(α^3,α^{16})	$(\alpha^{15},\alpha^3,\alpha^{13})$	$(\alpha^{15},\alpha^{16})$	1

Algoritem smo izvajali, dokler $\deg(r_i(z))$ ni postal manjši od $(\delta - 1)/2 = 2$. Sedaj preberemo $\sigma'(z)$ kot zadnji $t_i(z)$,

$$\sigma'(z) = \alpha^{15} + \alpha^3 z + \alpha^{13}z^2\,.$$

3.korak
Ničle $\sigma'(z)$ lahko dobimo po formuli

$$z_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$$

in dobimo ničli β^5 in β^9. Njuna inverza (lokatorja napake) sta $\beta^{13-5} = \beta^8$ in $\beta^{13-9} = \beta^4$. Torej je $a_0 = 4$ in $a_1 = 8$ ter zato $u_0 = \beta^4$ in $u_1 = \beta^8$

4.korak
Za velikosti napak pa rešimo sistem linearnih enačb (2.10),

$$\begin{bmatrix} 1 & 1 \\ \alpha^8 & \alpha^{16} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \alpha^{14} \end{bmatrix}\,,$$

in dobimo $y_0 = \alpha^{13} = (200) = 2$ ter tudi $y_1 = 2$. Torej je $\boldsymbol{e} = (000\,020\,002\,000\,0)$ in $\boldsymbol{r}$ dekodiramo v $\boldsymbol{c} = (220\,0\underline{\mathbf{0}}1\,11\underline{\mathbf{1}}\,211\,0)$. ▲

Ključna enačba za dekodiranje napak in izbrisov

Postopek prenosa je ponavadi tak, da vsako časovno enoto vemo, da smo prejeli nek simbol. Lahko prejmemo drugačnega, kot je bil poslan, lahko pa je prišlo do takih motenj na kanalu, da ne vemo, kateri simbol smo prejeli. Prvi nevšečnosti rečemo napaka, drugi pa **izbris**. Pri podatkovnih medijih so zaradi fizičnih poškodb izbrisi nekaj povsem običajnega.

S trikotniško neenakostjo se prepričamo, da koda z razdaljo $d \geq 2t + u + 1$ lahko popravi t napak in u izbrisov, glej Vanstone et al. [93, Theorem 7.5]. Recimo, da smo v prejetem vektorju prejeli ℓ_1 napak in ℓ_2 izbrisov. **Polinom lokatorjev napak** je potem enak

$$\sigma(z) = \prod_{j=0}^{\ell_1 - 1} (1 - u_j z)$$

in ker vemo na katerih mestih je prišlo do izbrisov, je **polinom lokatorjev izbrisov**

$$\lambda(z) = \prod_{j=0}^{\ell_2 - 1} (1 - u_{\ell_1 + j} z)$$

že poznan. Sedaj lahko definiramo **polinom lokatorja napak in izbrisov** kot $\widehat{\sigma}(z) = \sigma(z)\lambda(z)$, ki je stopnje $\ell = \ell_1 + \ell_2$.

Trditev 24. [93, Chapter 7.4] *Pri dekodiranju BCH kod s konstrukcijsko razdaljo δ, kjer so poleg napak možni tudi izbrisi, velja* **ključna enačba**

$$\widehat{\omega}(z) \equiv \sigma(z) T_2(z) \pmod{z^{\delta - \ell_2 - 1}}, \tag{2.13}$$

kjer je polinom $T_2(z)$ določen s pomočjo polinoma sindroma $S(z) = \sum_{j=0}^{\delta-2} S_{j+1} z^j$ z enačbo

$$\lambda(z) S(z) = T_1(z) + z^{\ell_2} T_2(z),$$

$\deg T_1 \leq \ell_2 - 1$*, polinom $\widehat{\omega}(z)$ pa s pomočjo $\omega(z) = \sum_{i=0}^{\ell-1} y_i u_i^a \prod_{\substack{j=0 \\ j \neq i}}^{\ell-1} (1 - u_j z)$ kot*

$$\widehat{\omega}(z) = \big(\omega(z) - \sigma(z) T_1(z)\big) / z^{\ell_2}.$$

Polinoma $\widehat{\omega}(z)$ in $\sigma(z)$ sta stopenj $\deg \widehat{\omega}(z) = \ell_1 - 1$ in $\deg \sigma(z) = \ell_1$. Velja tudi, da je poljuben par $(\omega'(z), \sigma'(z))$, ki reši enačbo (2.13) in zanj velja $\deg \omega'(z) \leq \ell_1 - 1$ in $\deg \sigma'(z) \leq \ell_1$, enolično določen do multiplikativne konstante $\mu \in \mathbb{F}$ natančno, torej

$$\omega'(z) = \mu\, \widehat{\omega}(z) \qquad \textit{in} \qquad \sigma'(z) = \mu\, \sigma(z).$$

Dokaz. Po trditvi 22 velja enačba

$$S(z) \widehat{\sigma}(z) \equiv \omega(z) \pmod{z^{\delta-1}}.$$

Z upoštevanjem definicij za $\widehat{\sigma}(z)$ in $T_2(z)$ izpeljemo

$$\omega(z) \equiv \sigma(z)\lambda(z)S(z) \pmod{z^{\delta-1}} \equiv \sigma(z)(T_1(z) + z^{\ell_2} T_2(z)) \pmod{z^{\delta-1}},$$

kar pomeni

$$\omega(z) - \sigma(z) T_1(z) \equiv \sigma(z) z^{\ell_2} T_2(z) \pmod{z^{\delta-1}},$$

oziroma
$$\widehat{\omega}(z) \equiv \sigma(z)T_2(z) \pmod{z^{\delta-\ell_2-1}}.$$
Enoličnost pokažemo analogno kot v dokazu leme 23. ■

Rešitev enačbe (2.13) lahko poiščemo analogno kot rešitev ključne enačbe (2.6) pri dekodiranju brez izbrisov. Možni načini so torej reševanje linearnega sistema enačb, izračun s pomočjo uporabe razširjenega Evklidovega algoritma, lahko pa tudi z Berlekampovim algoritmom.

2.3 Berlekampov algoritem za reševanje ključne enačbe

Do leta 1968 so ključno enačbo reševali tako, kot je predlagal Peterson [74], torej da so reševali ekvivalentni matrični sistem (2.9) s strani 47, ki povezuje koeficiente polinomov $S(z)$, $\sigma(z)$ in $\omega(z)$. Reševanje je zahtevalo precej računskih operacij, zato so bile BCH kode mnogo časa zanimive bolj iz teoretičnega vidika. Berlekampov algoritem [7] pa je prvi znani algoritem za hitro reševanje ključne enačbe (2.6) s strani 44 in od leta 1968 naprej so se te kode začele vedno bolj pojavljati tudi v praksi. V tem razdelku bomo po Berlekamp [8] povzeli ta algoritem. Najprej ga bomo konstruirali, nato bomo pokazali njegovo pravilnost, na koncu pa še, kako se poenostavi v primeru binarnih kod.

Konstrukcija algoritma

Ključno enačbo (2.6) bomo zapisali drugače in jo rešili iterativno, tako kot jo je Berlekamp [7]. Enačbo (2.6) pomnožimo z z, kar pomeni, da dobimo kongruenco po modulu z^δ, nato pa na obeh straneh prištejemo σ. Argumenti polinomov so enaki z in jih bomo tudi v nadaljevanju izpuščali. Naj bo $S_B = zS$ ter $\omega_B = \sigma + z\omega$ in dobljena enačba
$$(1+S_B)\sigma \equiv \omega_B \pmod{z^\delta} \tag{2.14}$$
je ekvivalentna ključni enačbi (2.6) iz trditve 22. To smo storili zato, ker bomo v nadaljevanju za vsak k potrebovali relacijo $\gcd(1+S_B, z^k) = 1$.

Ideja reševanja je v iterativnem postopku, pri katerem bomo na k-tem koraku, $k \in \{0, 1, \dots, \delta-1\}$, reševali enačbo
$$(1+S_B)\sigma^{(k)} \equiv \omega_B^{(k)} \pmod{z^{k+1}}. \tag{2.15}$$
po $\sigma^{(k)}$ in $\omega_B^{(k)}$. Za rešitev $\omega_B^{(k)}$ zahtevamo še $\deg \omega_B^{(k)} \le k$. Rešitev ključne enačbe bomo dobili kot $\sigma = \sigma^{(\delta-1)}$ ter $\omega_B = \omega_B^{(\delta-1)}$.

Ker sta stopnji iskanih polinomov σ in ω_B enaki številu napak, si želimo na vsakem koraku minimizirati stopnji polinomov $\sigma^{(k)}$ in $\omega_B^{(k)}$. Naj bo Δ_k koeficient pri z^{k+1} v produktu $(1+S_B)\sigma^{(k)}$. Potem je
$$(1+S_B)\sigma^{(k)} \equiv \omega_B^{(k)} + \Delta_k z^{k+1} \pmod{z^{k+2}}. \tag{2.16}$$
Če je $\Delta_k = 0$, potem lahko vzamemo $\sigma^{(k+1)} = \sigma^{(k)}$ ter $\omega_B^{(k+1)} = \omega_B^{(k)}$, v vsakem primeru pa si lahko pomagamo z reševanjem **pomožne enačbe**
$$(1+S_B)\tau^{(k)} \equiv \gamma^{(k)} + z^k \pmod{z^{k+1}}, \tag{2.17}$$

pri kateri bomo tudi iskali rešitvi $\tau^{(k)}$ in $\gamma^{(k)}$ s čim manjšo stopnjo. Sedaj od enačbe (2.16) odštejemo $(\Delta_k z)$-kratnik enačbe (2.17) in dobimo, da je

$$\begin{aligned} \sigma^{(k+1)} &= \sigma^{(k)} - \Delta_k z \tau^{(k)} \qquad \text{in} \\ \omega_B^{(k+1)} &= \omega_B^{(k)} - \Delta_k z \gamma^{(k)} \end{aligned} \tag{2.18}$$

ustrezna rešitev enačbe (2.15) za korak $k+1$. Rešitev pomožne enačbe (2.17) za korak $k+1$ lahko dobimo kot

$$\begin{aligned} \tau^{(k+1)} &= z\tau^{(k)} \qquad \text{in} \\ \gamma^{(k+1)} &= z\gamma^{(k)}\,. \end{aligned} \tag{2.19}$$

Če je $\Delta_k \neq 0$, pa lahko enačbo (2.16) delimo z Δ_k in vidimo, da je

$$\begin{aligned} \tau^{(k+1)} &= \frac{\sigma^{(k)}}{\Delta_k} \qquad \text{in} \\ \gamma^{(k+1)} &= \frac{\omega_B^{(k)}}{\Delta_k}\,. \end{aligned} \tag{2.20}$$

tudi ustrezna rešitev pomožne enačbe (2.17) za korak $k+1$. Ocenimo sedaj stopnji v (2.18) in (2.19), oziroma (2.20), definiranih polinomov $\sigma^{(k+1)}$ in $\tau^{(k+1)}$,

$$\begin{aligned} \deg \sigma^{(k+1)} &= \begin{cases} \deg \sigma^{(k)} & \text{, če je} \quad \Delta_k = 0 \quad \text{ali} \quad \deg \sigma^{(k)} > \deg \tau^{(k)} + 1 \\ 1 + \deg \tau^{(k)} & \text{, če je} \quad \Delta_k \neq 0 \quad \text{in} \quad \deg \tau^{(k)} > \deg \sigma^{(k)} - 1 \end{cases} \\ \deg \sigma^{(k+1)} &\leq \max(\deg \sigma^{(k)}, 1 + \deg \tau^{(k)})\,, \\ &\qquad\qquad \text{če je} \quad \Delta_k \neq 0 \quad \text{in} \quad \deg \sigma^{(k)} = \deg \tau^{(k)} + 1\,, \\ \deg \tau^{(k+1)} &= \begin{cases} \deg \sigma^{(k)} & \text{, če uporabimo (2.20)} \\ 1 + \deg \tau^{(k)} & \text{, če uporabimo (2.19)} \end{cases}\,. \end{aligned} \tag{2.21}$$

Iz istih enačb dobimo analogno tudi oceni stopenj polinomov $\omega_B^{(k+1)}$ in $\gamma^{(k+1)}$,

$$\begin{aligned} \deg \omega_B^{(k+1)} &= \begin{cases} \deg \omega_B^{(k)} & \text{, če je} \quad \Delta_k = 0 \quad \text{ali} \quad \deg \omega_B^{(k)} > \deg \gamma^{(k)} + 1 \\ 1 + \deg \gamma^{(k)} & \text{, če je} \quad \Delta_k \neq 0 \quad \text{in} \quad \deg \gamma^{(k)} > \deg \omega_B^{(k)} - 1 \end{cases} \\ \deg \omega_B^{(k+1)} &\leq \max(\deg \omega_B^{(k)}, 1 + \deg \gamma^{(k)})\,, \\ &\qquad\qquad \text{če je} \quad \Delta_k \neq 0 \quad \text{in} \quad \deg \omega_B^{(k)} = \deg \gamma^{(k)} + 1\,, \\ \deg \gamma^{(k+1)} &= \begin{cases} \deg \omega_B^{(k)} & \text{, če uporabimo (2.20)} \\ 1 + \deg \gamma^{(k)} & \text{, če uporabimo (2.19)} \end{cases}\,. \end{aligned} \tag{2.22}$$

Naj bo nenegativna funkcija $D(k)$ definirana rekurzivno z

$$D(k+1) = \begin{cases} D(k) & \text{, če je} \quad \Delta_k = 0 \quad \text{ali} \quad D(k) \geq \dfrac{k+1}{2} \\ k+1-D(k) & \text{, če je} \quad \Delta_k \neq 0 \quad \text{in} \quad D(k) \leq \dfrac{k+1}{2} \end{cases}\,. \tag{2.23}$$

Če za rešitvi $\sigma^{(k)}$ in $\tau^{(k)}$ na k-tem koraku velja

$$\begin{aligned} \deg \sigma^{(k)} &\leq D(k) \qquad \text{in} \\ \deg \tau^{(k)} &\leq k - D(k)\,, \end{aligned}$$

potem iz (2.21) dobimo, da velja tudi $\deg \sigma^{(k+1)} \leq D(k+1)$. In analogno, če za rešitvi $\omega^{(k)}$ in $\gamma^{(k)}$ na k-tem koraku velja

$$\begin{aligned} \deg \omega_B^{(k)} &\leq D(k) \qquad \text{in} \\ \deg \gamma^{(k)} &\leq k - D(k)\,, \end{aligned}$$

potem iz (2.22) dobimo, da velja tudi $\deg \omega_B^{(k+1)} \leq D(k+1)$. Pravilo, kdaj uporabiti (2.19), oziroma kdaj (2.20), za definiciji polinomov $\tau^{(k+1)}$ in $\gamma^{(k+1)}$, bi radi določili tako, da bo iz zgornjih predpostavk za k-ti korak sledilo tudi

$$\begin{aligned}\deg \tau^{(k+1)} &\leq k+1-D(k+1) \qquad \text{in}\\ \deg \gamma^{(k+1)} &\leq k+1-D(k+1)\,.\end{aligned}$$

Zato uporabimo

$$\begin{array}{lll} \text{pravilo (2.19),} & \text{če je} \quad \Delta_k = 0 \quad \text{ali} \quad D(k) > \dfrac{k+1}{2}\,, & \text{in}\\ \text{pravilo (2.20),} & \text{če je} \quad \Delta_k \neq 0 \quad \text{in} \quad D(k) < \dfrac{k+1}{2}\,. & \end{array}$$

in glede na definicije pravil (2.19) in (2.20), definicijo funkcije $D(k)$ ter predpostavke o stopnjah polinomov na k-tem koraku dobimo željeno omejitev za stopnji polinomov $\tau^{(k+1)}$ in $\gamma^{(k+1)}$. Če pa je $\Delta_k \neq 0$ in $D(k) = (k+1)/2$, potem nam obe pravili vedno zagotovita ustrezni stopnji polinomov $\tau^{(k+1)}$ in $\gamma^{(k+1)}$. Zato rekurzivno definiramo Boolovo funkcijo

$$B(k+1) = \begin{cases} B(k) & \text{, če smo za izbor } \tau^{(k+1)} \text{ in } \gamma^{(k+1)} \text{ uporabili pravilo (2.19)}\\ 1-B(k) & \text{, če smo za izbor } \tau^{(k+1)} \text{ in } \gamma^{(k+1)} \text{ uporabili pravilo (2.20)} \end{cases}$$

in v še zadnjem nedoločenem primeru določimo izbor med praviloma (2.19) in (2.20) kot

$$\begin{array}{lll} \text{pravilo (2.19),} & \text{če je} \quad \Delta_k \neq 0\,,\ D(k) = \dfrac{k+1}{2} \quad \text{in} \quad B(k) = 0\,, & \text{in}\\ \text{pravilo (2.20),} & \text{če je} \quad \Delta_k \neq 0\,,\ D(k) = \dfrac{k+1}{2} \quad \text{in} \quad B(k) = 1\,. & \end{array}$$

Sedaj se relaciji

$$\begin{aligned}\deg \omega_B^{(k)} &\leq D(k) - B(k) \qquad \text{in}\\ \deg \gamma^{(k)} &\leq k - D(k) - [1-B(k)]\end{aligned}$$

induktivno nadaljujeta in pri vsakem k je vsaj ena od njiju stroga neenakost.

Povzemimo postopek z algoritmom na sliki 2.4, izpeljano pa v naslednji trditvi. Celotno pravilnost algoritma bomo pokazali v nadaljevanju.

Dokaz pravilnosti algoritma

V nadaljevanju bomo pokazali, točneje v trditvi 28, da sta z algoritmom izračunana polinoma σ in ω_B res rešitvi ključne enačbe (2.14) in sta ustreznih stopenj.

Berlekampov algoritem za reševanje ključne enačbe nad poljubnim obsegom

predpostavke:

$S_B = zS$, $\omega_B = \sigma + z\omega$

ključna enačba: $(1+S_B)\sigma \equiv \omega_B \pmod{z^\delta}$

1. inicializacija:

   $\sigma^{(0)} = 1$, $\omega_B^{(0)} = 1$

   $\tau^{(0)} = 1$, $\gamma^{(0)} = 0$

   $D(0) = 0$, $B(0) = 0$

2. **for** $k = 0$ **to** $\delta - 1$ **do begin**

   Δ_k = koeficient pri z^{k+1} v polinomu $(1+S_B)\sigma^{(k)}$

   $\sigma^{(k+1)} = \sigma^{(k)} - \Delta_k z \tau^{(k)}$

   $\omega_B^{(k+1)} = \omega_B^{(k)} - \Delta_k z \gamma^{(k)}$

   if $\Delta_k = 0$ **or** $D(k) > \frac{k+1}{2}$ **or** $\left(\Delta_k \neq 0 \textbf{ and } D(k) = \frac{k+1}{2} \textbf{ and } B(k) = 0\right)$

   then begin

   $D(k+1) = D(k)$

   $B(k+1) = B(k)$

   $\tau^{(k+1)} = z\tau^{(k)}$

   $\gamma^{(k+1)} = z\gamma^{(k)}$

   end

   if $\Delta_k \neq 0$ **and** $\left(D(k) < \frac{k+1}{2} \textbf{ or } \left(D(k) = \frac{k+1}{2} \textbf{ and } B(k) = 1\right)\right)$

   then begin

   $D(k+1) = k + 1 - D(k)$

   $B(k+1) = 1 - B(k)$

   $\tau^{(k+1)} = \frac{\sigma^{(k)}}{\Delta_k}$

   $\gamma^{(k+1)} = \frac{\omega_B^{(k)}}{\Delta_k}$

   end

   end

3. $\sigma = \sigma^{(\delta-1)}$

   $\omega_B = \omega_B^{(\delta-1)}$

Slika 2.4: Berlekampov algoritem za reševanje ključne enačbe nad poljubnim obsegom

Trditev 25. [8, Theorem 7.41] *Za vsak k, $0 \leq k \leq \delta - 2$, za polinome $\sigma^{(k)}$, $\omega_B^{(k)}$, $\tau^{(k)}$ in $\gamma^{(k)}$, izračunane z Berlekampovim algoritmom, veljajo relacije*

$$\sigma^{(k)}(0) = \omega_B^{(k)}(0) = 1\,, \tag{2.24}$$

$$(1+S_B)\sigma^{(k)} \equiv \omega_B^{(k)} \pmod{z^{k+1}}\,, \tag{2.25}$$

$$(1+S_B)\tau^{(k)} \equiv \gamma^{(k)} + z^k \pmod{z^{k+1}}\,, \tag{2.26}$$

$$(1+S_B)\sigma^{(k)} \equiv \omega_B^{(k)} + \Delta_k z^{k+1} \pmod{z^{k+2}}\,, \tag{2.27}$$

$$\deg \sigma^{(k)} \leq D(k)\,, \qquad \textit{z enakostjo pri } B(k) = 1\,, \tag{2.28}$$

$$\deg \tau^{(k)} \leq k - D(k)\,, \qquad \textit{z enakostjo pri } B(k) = 0\,, \tag{2.29}$$

$$\deg \omega_B^{(k)} \leq D(k) - B(k)\,, \qquad \textit{z enakostjo pri } B(k) = 0\,, \tag{2.30}$$

$$\deg \gamma^{(k)} \leq k - D(k) - [1 - B(k)]\,, \qquad \textit{z enakostjo pri } B(k) = 1\,. \tag{2.31}$$

Vse relacije, razen opomb, kdaj so dosežene enakosti, so posledica inicializacije v algoritmu ter konstrukcije na prejšnjih straneh. Seveda pa jih lahko dokažemo tudi neposredno iz algoritma z indukcijo na k. Dokaz opomb, kdaj so dosežene enakosti, bomo navedli po dokazu naslednje leme. Naj opomnimo, da sta pogoja $B(k) = 0$ in $B(k) = 1$ za enakost v relacijah (2.28)–(2.31) zadostna pogoja, ne pa tudi nujno potrebna.

Lema 26. [8, Theorem 7.42] *Za vsak k velja*

$$\omega_{B}^{(k)}\tau^{(k)} - \sigma^{(k)}\gamma^{(k)} = z^k\,.$$

Dokaz. Najprej navzkrižno pomnožimo kongruenci (2.25) in (2.26) iz prejšnje trditve. Dobimo

$$(1+S_B)\tau^{(k)}\omega_B^{(k)} \equiv (1+S_B)\sigma^{(k)}(\gamma^{(k)} + z^k) \pmod{z^{k+1}}\,.$$

Ker je $\gcd(1+S_B, z^{k+1}) = 1$, lahko kongruenco delimo z $(1+S_B)$ in dobimo

$$\tau^{(k)}\omega_B^{(k)} \equiv \sigma^{(k)}\gamma^{(k)} + \sigma^{(k)}z^k \pmod{z^{k+1}}\,.$$

Ker je $\sigma_0^{(k)} = 1$ za vsak k, je $\sigma^{(k)}z^k \equiv \sigma_0^{(k)}z^k \equiv z^k \pmod{z^{k+1}}$ in zato

$$\tau^{(k)}\omega_B^{(k)} - \sigma^{(k)}\gamma^{(k)} \equiv z^k \pmod{z^{k+1}}\,.$$

Če iz prejšnje trditve upoštevamo še oceni stopenj polinomov $\tau^{(k)}$ in $\omega_B^{(k)}$, dobimo $\deg\tau^{(k)} + \deg\omega_B^{(k)} \leq k$. Podobno dobimo tudi $\deg\sigma^{(k)} + \deg\gamma^{(k)} \leq k$, zato je $\deg(\tau^{(k)}\omega_B^{(k)} - \sigma^{(k)}\gamma^{(k)}) \leq k$, kar pomeni, da zadnja kongruenca predstavlja enakost. ■

Dokaz. [dokaz trditve 25] Če je $B(k) = 0$, potem iz že dokazanih neenakosti trditve dobimo

$$\deg\gamma^{(k)} \leq k - D(k) - 1 \qquad \text{in} \qquad \deg\sigma^{(k)} \leq D(k)\,,$$

torej je

$$\deg(\sigma^{(k)}\gamma^{(k)}) \leq k-1\,.$$

Ker je po prejšnji lemi $\deg(\omega_B^{(k)}\tau^{(k)} - \sigma^{(k)}\gamma^{(k)}) = k$, dobimo iz ravnokar pokazanega

$$\deg(\omega_B^{(k)}\tau^{(k)}) = k\,.$$

Z upoštevanjem ocen za $\deg\omega_B^{(k)}$ in $\deg\tau^{(k)}$ iz trditve dobimo

$$\deg\tau^{(k)} = k - D(k) \qquad \text{in} \qquad \deg\omega_B^{(k)} = D(k)\,.$$

Analogno pokažemo, da pri $B(k) = 1$ dobimo

$$\deg\sigma^{(k)} = D(k) \qquad \text{in} \qquad \deg\gamma^{(k)} = k - D(k)$$

in s tem smo pokazali zadostnost pogojev za $B(k)$ za enakost v relacijah (2.28)–(2.31). ■

Sedaj bomo izpeljali trditev, ki nam pove kakšna je splošna rešitev sistema enačb

$$\begin{array}{rcl} \sigma(0) & = & 1 \\ (1+S_B)\sigma & \equiv & \omega \pmod{z^{k+1}} \end{array}\,. \tag{2.32}$$

V trditvi 25, enačbi (2.24) in (2.25), smo pokazali, da je par $\sigma^{(k)}$ in $\omega^{(k)}$ ustrezna rešitev zgornjega sistema. Iz enačbe (2.24) ter enačbe iz prejšnje leme vidimo tudi, da sta si polinoma $\sigma^{(k)}$ in $\tau^{(k)}$ paroma tuja, torej lahko vsak polinom, torej tudi σ, zapišemo kot linearno kombinacijo polinomov $\sigma^{(k)}$ in $\tau^{(k)}$. Prav tako dobimo tudi, da sta si polinoma $\omega^{(k)}$ in $\gamma^{(k)}$ paroma tuja, torej lahko vsak polinom, torej tudi ω, zapišemo kot linearno kombinacijo polinomov $\sigma^{(k)}$ in $\tau^{(k)}$. V naslednji trditvi bomo pokazali, da pri obeh zapisih linearnih kombinacij nastopata ista koeficienta, to je polinoma U in V.

Trditev 27. [8, Theorem 7.43] *Če sta σ in ω katerakoli polinoma, ki rešita sistem enačb (2.32), potem obstajata polinoma U in V z lastnostima*

$$\begin{aligned} U(0) &= 1 \quad \textit{in} \quad \deg U \leq D - D(k) \qquad \textit{ter} \\ V(0) &= 0 \quad \textit{in} \quad \deg V \leq D - [k - D(k)]\,, \end{aligned}$$

kjer je $D = \max(\deg \sigma, \deg \omega)$, tako da velja

$$\begin{aligned} \sigma &= U\sigma^{(k)} + V\tau^{(k)} \qquad \textit{in} \\ \omega &= U\omega^{(k)} + V\gamma^{(k)}\,. \end{aligned}$$

Dokaz. Napišimo relacijo (2.25) iz trditve 25 ter predpostavko te trditve

$$\begin{aligned} (1 + S_{_B})\sigma^{(k)} &\equiv \omega_{_B}^{(k)} \pmod{z^{k+1}}\,, \\ (1 + S_{_B})\sigma &\equiv \omega \pmod{z^{k+1}}\,. \end{aligned}$$

Prvo kongruenco pomnožimo z ω, drugo pa z $\omega_{_B}^{(k)}$. Dobimo

$$(1 + S_{_B})\sigma^{(k)}\omega \equiv (1 + S_{_B})\sigma\omega_{_B}^{(k)} \pmod{z^{k+1}}$$

in ker je $\gcd(1 + S_{_B}, z^{k+1}) = 1$, lahko kongruenco pokrajšamo,

$$\sigma^{(k)}\omega \equiv \sigma\omega_{_B}^{(k)} \pmod{z^{k+1}}\,,$$

torej obstaja polinom $\bar{V}$, da je $\sigma^{(k)}\omega - \sigma\omega_{_B}^{(k)} = -z^{k+1}\bar{V}$. Naj bo $V = z\bar{V}$. Potem je

$$\sigma^{(k)}\omega - \sigma\omega_{_B}^{(k)} = -z^k V \tag{2.33}$$

in $V(0) = 0$ ter $\deg V \leq D(k) + D - k$.

Napišimo sedaj relacijo (2.26) iz trditve 25 ter predpostavko te trditve

$$\begin{aligned} (1 + S_{_B})\tau^{(k)} &\equiv \gamma^{(k)} + z^k \pmod{z^{k+1}}\,, \\ (1 + S_{_B})\sigma &\equiv \omega \pmod{z^{k+1}}\,. \end{aligned}$$

Kongruenci med seboj navzkrižno pomnožimo ter krajšamo z $(1 + S_{_B})$. Dobimo

$$\tau^{(k)}\omega \equiv \sigma\gamma^{(k)} + \sigma z^k \pmod{z^{k+1}}\,.$$

Ker je $\sigma(0) = 1$, je $\sigma z^k \equiv z^k \pmod{z^{k+1}}$ in lahko sklepamo, da obstaja polinom U, za katerega velja $U(0) = 1$ in $\deg U \leq D - D(k)$ ter izpolni enačbo

$$\tau^{(k)}\omega - \sigma\gamma^{(k)} = Uz^k\,. \tag{2.34}$$

Pomnožimo enačbo (2.33) s $\tau^{(k)}$, enačbo (2.34) s $\sigma^{(k)}$ ter ju odštejmo med seboj. Dobimo

$$(\tau^{(k)}\omega^{(k)} - \sigma^{(k)}\gamma^{(k)})\sigma = z^k(U\sigma^{(k)} + V\tau^{(k)})\,.$$

Z uporabo enačbe iz leme 26 dobimo

$$\sigma = U\sigma^{(k)} + V\tau^{(k)}\,.$$

Z množenjem enačbe (2.33) z $\gamma^{(k)}$ in enačbe (2.34) z $\omega^{(k)}$ pa dobimo z analognim postopkom

$$\omega = U\omega^{(k)} + V\gamma^{(k)}\,.$$

■

Trditev 28. [8, Theorem 7.44] *Če sta si polinoma σ in ω paroma tuja ter velja*

$$\begin{aligned} \sigma(0) &= 1 \qquad \textit{in} \\ (1 + S_B)\sigma &\equiv \omega \pmod{z^{k+1}}\,, \end{aligned}$$

potem je

$$\deg\sigma \geq D(k) + 1 - B(k) \geq D(k) \qquad \textit{ali} \qquad \deg\omega \geq D(k) \tag{2.35}$$

in, če velja še $\deg\sigma \leq (k+1)/2$ in $\deg\omega \leq k/2$, potem je $\sigma = \sigma^{(k)}$ ter $\omega = \omega_B^{(k)}$.

Dokaz. Prvi del trditve je direktna posledica enačbe (2.34) iz prejšnje trditve ter ocen stopenj polinomov $\tau^{(k)}$ in $\gamma^{(k)}$ iz trditve 25.

Dokažimo še drugi del trditve. Če je $\deg\sigma \leq (k+1)/2$ in $\deg\omega \leq k/2$, potem s pomočjo prvega dela trditve dobimo $D(k) \leq (k+1)/2$, s strogo neenakostjo pri $B(k) = 0$. Oceno za $D(k)$ sedaj vstavimo v oceni stopenj polinomov $\sigma^{(k)}$ in $\omega^{(k)}$ v trditev 25 in dobimo

$$\deg\sigma^{(k)} \leq \frac{k+1}{2} \qquad \text{in} \qquad \deg\omega^{(k)} \leq \frac{k}{2}\,,$$

kar nam da z upoštevanjem predpostavk o $\deg\sigma$ in $\deg\omega$

$$\deg(\sigma^{(k)}\omega) \leq k + 1/2 < k+1 \qquad \text{in} \qquad \deg(\sigma\omega^{(k)}) \leq k + 1/2 < k+1\,.$$

Oceni vstavimo v enačbo (2.33) iz prejšnje trditve

$$\deg(-z^kV) = \deg(\sigma^{(k)}\omega - \sigma\omega^{(k)}) < k+1\,,$$

kar pomeni, da je $\deg V < 1$, in ker je $V(0) = 0$, je $V = 0$. To sedaj vstavimo v prejšnjo trditev in dobimo $\sigma = U\sigma^{(k)}$ in $\omega = U\omega^{(k)}$. Ker sta si σ in ω po predpostavki tuja in ker je $U(0) = 1$, dobimo $U = 1$, iz česar sledi trditev. ■

Časovna zahtevnost algoritma

V naslednji trditvi bomo pokazali, da je zaporedje $\{D(k)\}_{k\geq 0}$ naraščajoče in da za vsak k, $0 \leq k \leq \delta - 1$, velja $0 \leq D(k) \leq k$, kar bo osnova za oceno časovne zahtevnosti algoritma.

Trditev 29. *Naj bo ℓ število napak, povzročenih pri prenosu kodne besede, in δ konstrukcijska razdalja BCH kode. Če je $\ell \leq (\delta-1)/2$, potem je časovna zahtevnost Berlekampovega algoritma za iskanje rešitve ključne enačbe enaka največ $O(\ell\,\delta)$ operacij v obsegu $\mathbb{F}_q$. V vsakem primeru pa se algoritem konča po največ $O(\delta^2)$ operacijah v obsegu $\mathbb{F}_q$.*

Dokaz. Pokažimo najprej, da je zaporedje $\{D(k)\}_{k\geq 0}$, definirano v (2.23) na strani 53 in z začetnim členom $D(0) = 0$, naraščajoče. Zaporedje se lahko poveča le pri tistih k, kjer določimo $D(k+1) = k+1-D(k)$. Da se v algoritmu izvede ta korak, je potreben pogoj $D(k) \leq (k+1)/2$. V tem primeru dobimo z definicijo za $D(k+1)$ oceno $D(k) \leq (D(k+1)+D(k))/2$, iz katere sledi $D(k) \leq D(k+1)$. V nadaljevanju pa bomo uporabili tudi lastnost, da je $0 \leq D(k) \leq k$, o kateri se hitro prepričamo z indukcijo na k, $0 \leq k \leq \delta - 1$.

Trditev 22 nam pove, da pri pogoju $\ell \leq (\delta-1)/2$ obstaja rešitev σ in ω ključne enačbe (2.6). Ker je $\ell \leq (\delta-1)/2$, pri $k = \delta - 1$ velja $\deg\sigma = \ell \leq (\delta-1)/2 \leq k/2$ in $\deg\omega = \ell - 1 \leq (\delta-1)/2 - 1 \leq k/2 - 1$, kar izpolnjuje pogoje drugega dela trditve 28, in zato je $\sigma = \sigma^{(\delta-1)}$ in $\omega = \omega^{(\delta-1)}$.

Za stopnji polinomov σ in ω velja po oceni (2.35) iz trditve 28 tudi ocena $\deg\sigma \geq D(\delta-1)$ ali pa ocena $\deg\omega \geq D(\delta-1)$. Skupaj z relacijama (2.28) in (2.30) iz trditve 25 pa dobimo $\deg\sigma = D(\delta-1)$ ali $\deg\omega = D(\delta-1)$, kar pomeni, da je $D(\delta-1) = \ell$ ali $D(\delta-1) = \ell-1$.

Zgornja ocena za časovno zahtevnost Berlekampovega algoritma s slike 2.4 je potem asimptotično enaka seštevku operacij za izračun Δ_k, $\sigma^{(k+1)}$ in $\omega_B^{(k+1)}$, $0 \leq k \leq \delta - 1$, po posameznih korakih **for** zanke iz 2. koraka algoritma.

Za izračun Δ_k algoritem potrebuje $O(\deg\sigma^{(k)})$ operacij in ker je

$$\deg\sigma^{(k)} \leq D(k) \leq D(\delta-1) \leq \ell\,,$$

potrebujemo za ta izračun največ $O(\ell)$ operacij. Upoštevali smo relacijo (2.28) iz trditve 25 in monotonost zaporedja $\{D(k)\}_{k\geq 0}$.

Če je $\deg\sigma^{(k)} \neq 1 + \deg\tau^{(k)}$ in $k \leq \delta - 2$, lahko $\sigma^{(k+1)}$ izračunamo z največ $O(\deg\sigma^{(k+1)})$ operacijami, oziroma ker je $\deg\sigma^{(k+1)} \leq D(k+1) \leq D(\delta-1) \leq \ell$, je to največ $O(\ell)$ operacij. Upoštevali smo relacijo (2.28) iz trditve 25 in definicijo za $\sigma^{(k+1)}$. Na koraku $k = \delta - 1$ pa za izračun $\sigma^{(k+1)}$ potrebujemo največ $O(\max\{\deg\sigma^{(k)}, 1+\deg\tau^{(k)}\})$ operacij, oziroma ker je $\max\{\deg\sigma^{(k)}, 1+\deg\tau^{(k)}\} \leq \max\{D(k), 1+k-D(k)\} \leq 1+k = \delta$, je to največ $O(\delta)$ operacij. Če pa je $\deg\sigma^{(k)} = 1 + \deg\tau^{(k)}$, pa za izračun $\sigma^{(k+1)}$ potrebujemo največ $O(\deg\sigma^{(k)})$ operacij, oziroma ker je $\deg\sigma^{(k)} \leq D(k) \leq D(\delta-1) \leq \ell$, je to največ $O(\ell)$ operacij.

Analogno kot za $\sigma^{(k+1)}$ dobimo tudi zgornjo oceno za število operacij, potrebnih za izračun $\omega^{(k+1)}$, $0 \leq k \leq \delta-1$. Za $k \leq \delta-2$ je to največ $O(\ell)$ operacij, za $k = \delta-1$ pa največ $O(\delta)$ operacij. Zgornja meja za časovno zahtevnost algoritma je torej $(\delta-1)\cdot O(\ell) + O(\delta) = O(\ell\,\delta)$ operacij v obsegu $\mathbb{F}_q$.

V primeru, da pri prenosu kodne besede pride do več kot $(\delta-1)/2$ napak, niso izpolnjeni pogoji za trditev 28, kar pomeni, da ne moremo uporabljati ocene $D(\delta-1) \leq \ell$. Vedno pa lahko uporabljamo oceno $D(\delta-1) \leq \delta - 1$, kar pomeni, da je časovna zahtevnost algoritma največ $O(\delta^2)$ operacij iz $\mathbb{F}_q$. ∎

Poenostavitev algoritma za binarne kode

V tem podrazdelku bomo privzeli, da je BCH koda binarna, torej da je obseg simbolov enak GF(2). Pokazali bomo, da lahko Berlekampov algoritem pospešimo in je njegov čas izvajanja za polovico krajši, saj lahko polovico korakov izpustimo.

Za razliko od prejšnjih podrazdelkov bomo privzeli, da je koda preprosta, torej da je $a = 1$, saj bomo tako lažje preračunavali indekse. Torej je $S_k = \sum_{j=0}^{\ell-1} u_j^k$. Posplošimo to definicijo sedaj na vse k, $k \geq 1$. Ker je karakteristika obsega enaka 2, je

$$S_k^2 = \left(\sum_{j=0}^{\ell-1} u_j^k\right)^2 = \sum_{j=0}^{\ell-1} u_j^{2k} = S_{2k}\,.$$

Naj bo "$\widehat{\ }$" oznaka za sodi del funkcije, "$\widetilde{\ }$" oznaka za lihi del funkcije, $S_{_B}(z)$ pa vrsta

$$S_{_B}(z) = \sum_{k=1}^{\infty} S_k z^k\,.$$

Potem s pomočjo prejšnje relacije in upoštevanja karakteristike obsega dobimo relacijo

$$[S_{_B}(z)]^2 = \sum_{k=1}^{\infty} S_{2k} z^{2k} = \widehat{S}_{_B}(z)\,, \tag{2.36}$$

ki nam bo omogočila poenostavitev algoritma.

Lema 30. [8, Lemma 7.61] *Naj bosta $1+S$ in $1+R$ recipročni generatorski funkciji v obsegu s karakteristiko 2, to je*

$$(1+S)(1+R) = 1\,.$$

Potem je $\widehat{R} = 0$ natanko tedaj, ko je $\widehat{S} = S^2$

Dokaz. Zapišimo funkciji S in R kot vsoti sodega in lihega dela, torej $S = \widehat{S} + \widetilde{S}$ in $R = \widehat{R} + \widetilde{R}$, ter to vstavimo v enačbo $(1+S)(1+R) = 1$. Zapišimo enačbo za sodi del ter enačbo za lihi del dobljene enačbe. Dobimo

$$(1+\widehat{S})(1+\widehat{R}) + \widetilde{S}\widetilde{R} = 1 \qquad \text{ter} \qquad \widetilde{S}(1+\widehat{R}) + \widetilde{R}(1+\widehat{S}) = 0\,.$$

Sedaj od $(1+\widehat{S})$-kratnika prve enačbe odštejemo $\widetilde{S}$-kratnik druge enačbe in dobimo $[(1+\widehat{S})^2 - \widetilde{S}^2](1+\widehat{R}) = 1+\widehat{S}$. Ker je karakteristika obsega enaka 2, je $(1+\widehat{S})^2 - \widetilde{S}^2 = 1 + \widehat{S}^2 + \widetilde{S}^2 = 1 + S^2$. To upoštevamo v prej dobljeni relaciji in vidimo, da je $\widehat{R} = 0$ natanko tedaj, ko je $1+\widehat{S} = 1+S^2$, kar smo želeli dokazati. ∎

Trditev 31. [8, Theorem 7.62] *Če je preprosta BCH koda definirana nad obsegom s karakteristiko 2, potem funkcije iz Berlekampovega algoritma zadoščajo relacijam*

$$\begin{aligned} \omega_{_B}^{(i)} &= \widehat{\sigma}^{(i)}\,, \\ \gamma^{(i)} &= \begin{cases} \widetilde{\tau}^{(i)} & ,\ \text{če je } i \text{ sod} \\ \widehat{\tau}^{(i)} & ,\ \text{če je } i \text{ lih} \end{cases} \qquad\qquad \textit{in} \\ \Delta_i &= 0\,,\ \text{če je } i \text{ lih}\,. \end{aligned}$$

Dokaz. Z indukcijo na i. Za $i = 0$ trditev drži. Predpostavimo, da trditev drži za $i = 2k$, torej

$$\omega_{_B}^{(2k)} = \widehat{\sigma}^{(2k)} \qquad \text{in} \qquad \gamma^{(2k)} = \widetilde{\tau}^{(2k)}\,.$$

Z upoštevanjem relacij iz definicije algoritma ter indukcijske predpostavke dobimo

$$\omega_{_B}^{(2k+1)} = \omega_{_B}^{(2k)} - \Delta_{2k} z \gamma^{(2k)} = \widehat{\sigma}^{(2k)} - \Delta_{2k} z \widetilde{\tau}^{(2k)} = \widehat{\ }\left(\sigma^{(2k)} - \Delta_{2k} z \tau^{(2k)}\right) = \widehat{\sigma}^{(2k+1)}\,.$$

Podobno dobimo

$$\gamma^{(2k+1)} = z\gamma^{(2k)} = z\widetilde{\tau}^{(2k)} = \widehat{\ } \left(z\tau^{(2k)}\right) = \widehat{\tau}^{(2k+1)} ,$$

če se je algoritem izvajal v prvi **if** zanki, in

$$\gamma^{(2k+1)} = \frac{\omega_B^{(2k)}}{\Delta_{2k}} = \frac{\widehat{\sigma}^{(2k)}}{\Delta_{2k}} = \widehat{\ } (\frac{\sigma^{(2k)}}{\Delta_{2k}}) = \widehat{\tau}^{(2k+1)} ,$$

če se je algoritem izvajal v drugi **if** zanki. Pokažimo še $\Delta_{2k+1} = 0$. Z upoštevanjem relacije (2.27) s strani 55 dobimo

$$(1 + S_B)\sigma^{(2k+1)} \equiv \widehat{\sigma}^{(2k+1)} + \Delta_{2k+1} z^{2k+2} \pmod{z^{2k+3}} .$$

Če je $1 + R_B$ recipročna funkcija funkcije $1 + S_B$, potem iz enačbe (2.36) in prejšnje leme lahko zaključimo $\widehat{R}_B = 0$. Zadnjo kongruenco sedaj pomnožimo z $1 + R_B$ in dobimo

$$\sigma^{(2k+1)} \equiv (1 + \widetilde{R}_B)\widehat{\sigma}^{(2k+1)} + (1 + \widetilde{R}_B)\Delta_{2k+1} z^{2k+2} \pmod{z^{2k+3}} .$$

Ker R_B nima konstantnega člena, je

$$\widetilde{\sigma}^{(2k+1)} + \widetilde{R}_B \widehat{\sigma}^{(2k+1)} \equiv \Delta_{2k+1} z^{2k+2} \pmod{z^{2k+3}} .$$

Ker je sodi del na levi strani enak 0, mora biti $\Delta_{2k+1} = 0$. Torej smo pokazali induktivni korak $2k \rightsquigarrow 2k + 1$.

Ker je $\Delta_{2k+1} = 0$, dobimo v algoritmu

$$\begin{aligned} \sigma^{(2k+2)} &= \sigma^{(2k+1)}, & \tau^{(2k+2)} &= z\tau^{(2k+1)} , \\ \omega_B^{(2k+2)} &= \omega_B^{(2k+1)}, & \gamma^{(2k+2)} &= z\gamma^{(2k+1)} , \end{aligned}$$

torej je

$$\begin{aligned} &\omega_B^{(2k+2)} = \omega_B^{(2k+1)} = \widehat{\sigma}^{(2k+1)} = \widehat{\sigma}^{(2k+2)} \qquad \text{in} \\ &\gamma^{(2k+2)} = z\gamma^{(2k+1)} = z\widehat{\tau}^{(2k+1)} = \widetilde{\ } \left(z\tau^{(2k+1)}\right) = \widetilde{\tau}^{(2k+2)} , \end{aligned}$$

s čemer smo pokazali še induktivni korak $2k + 1 \rightsquigarrow 2k + 2$. ■

Trditev 32. [8, Theorem 7.63] *Neenakosti (2.28) in (2.29) s strani 55 postanejo pri preprostih binarnih BCH kodah enakosti, torej*

$$\deg \sigma^{(k)} = D(k) \qquad \textit{in} \qquad \deg \tau^{(k)} = k - D(k) .$$

Dokaz. Z indukcijo na k. Za $k = 0$ je $\deg \sigma^{(0)} = \deg \tau^{(0)} = 0 = D(0)$ in trditev očitno velja. Pokažimo še induktivni korak $k \rightsquigarrow k + 1$.

Če sta polinoma $\sigma^{(k)}$ in $\Delta_k z\tau^{(k)}$ enakih stopenj, je po indukcijski predpostavki $D(k) = k - D(k) + 1$, torej $k + 1 = 2D(k)$, kar pomeni, da je k lih. Po prejšnji trditvi je $\Delta_k = 0$, torej dobimo v algoritmu $\sigma^{(k+1)} = \sigma^{(k)}$, $\tau^{(k+1)} = z\tau^{(k)}$ in $D(k + 1) = D(k)$, kar pomeni, da relacije iz trditve v tem primeru držijo.

Privzemimo sedaj, da sta polinoma $\sigma^{(k)}$ in $\Delta_k z\tau^{(k)}$ različnih stopenj. Z upoštevanjem relacije $\sigma^{(k+1)} = \sigma^{(k)} - \Delta_k z\tau^{(k)}$ iz definicije algoritma ter induktivne predpostavke dobimo

$$\deg \sigma^{(k+1)} = \max(D(k), k - D(k) + 1) .$$

Če se algoritem na tem koraku izvede v prvi **if** zanki, je $D(k) \geq (k+1)/2$ in zato $k - D(k) + 1 \leq D(k)$. V tem primeru je $\deg \sigma^{(k+1)} = D(k)$ in ker je v tej **if** zanki $D(k+1) = D(k)$, je $\deg \sigma^{(k+1)} = D(k+1)$. Če pa se algoritem na tem koraku izvede v drugi **if** zanki, je $D(k) \leq (k+1)/2$ in zato $k-D(k)+1 \geq D(k)$. Torej je $\deg \sigma^{(k+1)} = k-D(k)+1$ in ker je v tej **if** zanki $D(k+1) = k+1-D(k)$, je $\deg \sigma^{(k+1)} = D(k+1)$.

Izračunajmo še $\deg \tau^{(k+1)}$. Če se algoritem na tem koraku izvede v prvi **if** zanki, je $\deg \tau^{(k+1)} = \deg \tau^{(k)} + 1$. Z upoštevanjem, da je v tej **if** zanki $D(k+1) = D(k)$ in indukcijske predpostavke dobimo $\deg \sigma^{(k+1)} = k - D(k) + 1 = k + 1 - D(k+1)$. Če pa se algoritem na tem koraku izvede v drugi **if** zanki, je $\deg \tau^{(k+1)} = \deg \sigma^{(k)}$. Z upoštevanjem, da je v tej **if** zanki $D(k+1) = k+1-D(k)$ in indukcijske predpostavke dobimo $\deg \tau^{(k+1)} = D(k) = k+1-D(k+1)$. ■

Podatek $\Delta_{2k+1} = 0$ iz predzadnje trditve nam točno določi izvajanje v algoritmu pri lihih korakih, kar je razvidno iz enačb (2.37). Zadnja trditev pa nam zelo poenostavi pogoje **if** stavkov. Poenostavljen algoritem je predstavljen na sliki 2.5.

Berlekampov algoritem za reševanje ključne enačbe za preproste binarne kode
predpostavke:
$\quad S_B = zS,\ \omega_B = \sigma + z\omega$
$\quad$ ključna enačba: $(1+S_B)\sigma \equiv \omega_B \pmod{z^\delta}$
1. inicializacija:
$\quad \sigma^{(0)} = 1$
$\quad \tau^{(0)} = 1$
2. **for** $k = 0$ **to** $\lfloor(\delta-1)/2\rfloor$ **do begin**
$\quad \Delta_{2k}$ = koeficient pri z^{2k+1} v polinomu $(1+S_B)\sigma^{(2k)}$
$\quad \sigma^{(2k+2)} = \sigma^{(2k)} + \Delta_{2k} z \tau^{(2k)}$
$\quad$ **if** $\Delta_{2k} = 0$ **or** $\deg \sigma^{(2k)} > k$ **then** $\tau^{(2k+2)} = z^2 \tau^{(2k)}$
$\quad$ **if** $\Delta_{2k} \neq 0$ **and** $\deg \sigma^{(2k)} \leq k$ **then** $\tau^{(2k+2)} = \dfrac{z\sigma^{(2k)}}{\Delta_{2k}}$
end
3. $\sigma = \sigma^{(2\lfloor(\delta-1)/2\rfloor)}$

Slika 2.5: Berlekampov algoritem za reševanje ključne enačbe za preproste binarne kode

2.4 Reed-Solomonove kode in dekodiranje v sezname

V tem razdelku bomo predstavili poseben razred BCH kod, to so Reed-Solomonove kode, pri katerem nam ni potrebno računati v razširitvi obsega simbolov. Kot bomo videli, s tem izgubimo možnost, da je razdalja kode večja od konstrukcijske, pridobimo pa na hitrosti preračunavanja, saj se vse operacije izvajajo v manjšem obsegu. Dolžina kode ostane nespremenjena. Te kode so ene izmed najpogosteje uporabljenih v praksi. Uporabljajo se tudi pri zapisovanju podatkov na zgoščenkah (CD-jih), kjer se uporabljata še dodatni tehniki **prepletanja kod** in **rekonstrukcija izbrisa** pri BCH kodah, o kateri smo že pisali v zadnjem delu razdelka 2.2. Prva tehnika omogoči, da je kodna beseda zapisana bolj razpršeno na podatkovnem mediju, obe skupaj pa, da posamezne fizične poškodbe zgoščenke (praske, luknjice) v majhnih razsežnostih ne povzročijo izgube podatkov. Več o tehnikah in sami tehnologiji zgoščenk si lahko preberemo v Vanstone in Oorschot [93, Chapter 7], lahko pa tudi v Jurišić [45].

Naj bo α element reda n iz obsega $\mathbb{F} = \mathrm{GF}(q)$ ter

$$g(x) = (x - \alpha^{a+1})(x - \alpha^{a+2}) \cdots (x - \alpha^{a+\delta-1})$$

za izbrani celi števili $a \geq 0$ in $\delta \geq 2$. Ker $g(x)$ deli $x^n - 1$, je po trditvi 14 generator ciklične (n, k)-kode $C \subset V_n(\mathbb{F})$, kjer je $n - k = \delta - 1$. Na tak način konstruirano BCH kodo imenujemo **Reed-Solomonova koda** ali na kratko (n, k)-RS koda. Glede na splošne BCH kode je njena posebnost v tem, da je α element obsega $\mathbb{F}$ in ne kake njegove razširitve. Ker je α^i element obsega $\mathbb{F}$ za vsak i, je njegov minimalni polinom enak $x - \alpha^i$. Po trditvi 21 je razdalja BCH kod C vsaj δ. Pokažimo, da je pri RS kodah to tudi dejanska razdalja.

Trditev 33. *Vsaka (n, k)-RS koda C s konstrukcijsko razdaljo δ ima razdaljo*

$$d = n - k + 1 = \delta\,.$$

Dokaz. Naj bo $g(x)$ generatorski polinom kode C. Potem po definiciji RS kod obstaja element $\alpha \in \mathbb{F}$ reda n tak, da je

$$g(x) = \prod_{i=1}^{\delta-1}(x - \alpha^i)\,,$$

kjer je $\delta = n - k + 1$. Po trditvi 21 je razdalja kode C enaka vsaj δ. Označimo z $\boldsymbol{g}$ kodno besedo, katero predstavljajo koeficienti polinoma $g(x)$. Ker je stopnja $g(x)$ manjša od δ, je $w(\boldsymbol{g}) \leq \delta$, to pa po trditvi 4 pomeni, da je $d \leq \delta$. ■

Za linearne kode velja Singletonova ocena, glej str. 23, to pomeni, da za razdaljo poljubne linearne (n, k)-kode velja ocena $d \leq n - k + 1$. RS kode so glede na doseženo razdaljo optimalne linearne kode. Glede na zadnjo trditev in trditev 3 lahko zaključimo, da (n, k)-RS koda lahko odkrije $n - k$ napak ali pa popravi $\lfloor (n-k)/2 \rfloor$ napak. Reed-Solomonove kode nad $\mathrm{GF}(2^m)$ so poleg preproste implementacije (obseg ima karakteristiko 2) zanimive tudi zaradi svoje sposobnosti odkrivanja in popravljanja grozdnih napak, ki smo jih na strani 36 že omenili.

Prvotno sta Reed in Solomon [78] RS kode definirala s pomočjo diskretne Fourierove transformacije, glej MacWilliams et al. [56, Chapter 8.6]. Naj bo $q \geq n$ in naj bo α primitivni n-ti koren enote v obsegu $\mathbb{F} = \mathrm{GF}(q^m)$ pri nekem $m \in \mathbb{N}$. **Diskretna Fourierova transformacija** $\mathcal{F} : \mathbb{F}^n \to \mathbb{F}^n$ je definirana kot linearna preslikava, podana z matriko

$$F = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-1} & \alpha^{2(n-1)} & \cdots & \alpha^{(n-1)^2} \end{bmatrix}$$

Naj bo $\boldsymbol{a} = (a_0 \ldots a_{n-1})^T \in \mathbb{F}^n$ in $a(x) = \sum_{i=0}^{n-1} a_i x^i$ vektorju $\boldsymbol{a}$ asociiran polinom. Naj bo $\boldsymbol{A} = (A_0 \ldots A_{n-1})^T$ diskretna Fourierova transformacija vektorja $\boldsymbol{a}$, torej $\boldsymbol{A} = F\boldsymbol{a}$. Potem je $A_j = a(\alpha^j) = \sum_{i=0}^{n-1} a_i \alpha^{ij}$, $j \in \mathbb{Z}$. V naslednji trditvi pa bomo izpeljali formulo za inverz diskretne Fourierove transformacije.

Trditev 34 (Inverzna formula). *Diskretna Fourierova transformacija $\mathcal{F} : \mathbb{F}^n \to \mathbb{F}^n$ je bijektivna preslikava, katere inverz je določen s predpisom*

$$\mathcal{F}^{-1} : \boldsymbol{A} \mapsto \boldsymbol{a}\,,$$
$$a(x) = \frac{1}{n}\sum_{i=0}^{n-1} A(\alpha^{-i})x^i\,, \tag{2.37}$$

kjer je polinom $A(x)$ asociiran vektorju $\boldsymbol{A} = (A_0 \dots A_{n-1})^T \in \mathbb{F}^n$ in polinom $a(x)$ asociiran vektorju $\boldsymbol{a} = (a_0 \dots a_{n-1})^T \in \mathbb{F}^n$.

Dokaz. Determinanta matrike F je Vandermondova determinanta, glej stran 40, ki je v tem primeru različna od 0. Torej je preslikava $\mathcal{F}$ injektivna in ker $F \in \mathbb{F}^{n\times n}$, je tudi bijektivna. Dokažimo še, da formula (2.37) res določa inverzno preslikavo:

$$\begin{aligned}
\frac{1}{n}\sum_{i=0}^{n-1} A(\alpha^{-i})x^i &= \frac{1}{n}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} A_j\alpha^{-ij}x^i = \frac{1}{n}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{\ell=0}^{n-1} a_\ell\alpha^{j\ell}\alpha^{-ij}x^i = \\
&= \frac{1}{n}\sum_{i=0}^{n-1}\sum_{\ell=0}^{n-1} a_\ell x^i \sum_{j=0}^{n-1}\alpha^{(\ell-i)j}
\end{aligned}$$

Če je $\ell \neq i$, je $\sum_{j=0}^{n-1}\alpha^{(\ell-i)j} = (1-\alpha^{(\ell-i)n})/(1-\alpha^{\ell-i}) = 0$, saj je $\alpha^n = 1$. Če pa je $\ell = i$, je ta vsota enaka n. Torej je

$$\frac{1}{n}\sum_{i=0}^{n-1} A(\alpha^{-i})x^i = \frac{1}{n}\sum_{i=0}^{n-1} a_i x^i n = a(x)\,,$$

kar potrjuje trditev. ■

Reed Solomonova koda z dolžino n in dolžino sporočila k je definirana kot

$$C = \left\{ \left(m(1), m(\alpha), m(\alpha^2), \dots, m(\alpha^{n-1})\right) \,\middle|\, m(x) = \sum_{j=0}^{k-1} m_j x^j,\ \boldsymbol{m} = (m_0 \dots m_{k-1}) \in \mathcal{M} \right\},$$

kjer je $\mathcal{M} = \Sigma^k$ prostor sporočil. Posamezna kodna beseda je torej sestavljena iz n vrednosti polinoma, ki ustreza bloku sporočila. Iz te definicije lahko hitro zaključimo, da je razdalja (n,k)-RS kode enaka $n-k+1$, saj se dva različna polinoma stopenj največ $k-1$ ujemata v največ $k-1$ točkah. Pravzaprav je posamezna kodna beseda RS-kode enaka diskretni Fourierovi transformaciji vektorja $(m_0 \dots m_{k-1}\underbrace{0\dots0}_{n-k})$, kjer je $\boldsymbol{m} = (m_0 \dots m_{k-1})$ izbran blok sporočila. Sedaj bomo pokazali, da je ta definicija RS kod ekvivalentna prejšnji, kar pomeni, da RS kode lahko dekodiramo tudi s pomočjo Berlekampovega algoritma. Nato pa bomo izpeljali danes aktualno dekodiranje v sezname, ki lahko odpravi mnogo več napak kot nedvoumno dekodiranje.

Trditev 35. *Naj bo k dimenzija prostora sporočil $\mathcal{M}$. Naj bo $n \in \mathbb{N}$, $n > k$, in še $g(x) = (x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{n-k})$, kjer je α element reda n v obsegu $\mathbb{F}_q$. Potem sta (n,k)-RS kodi*

$$C_1 = \left\{ a(x)g(x) \,\middle|\, a(x) = \sum_{j=0}^{k-1} a_j x^j,\ \boldsymbol{a} = (a_0 \dots a_{k-1}) \in \mathcal{M} \right\}$$

in

$$C_2 = \Big\{ \sum_{i=0}^{n-1} m(\alpha^i)x^i \,\Big|\, m(x) = \sum_{j=0}^{k-1} m_j x^j,\ \boldsymbol{m} = (m_0 \dots m_{k-1}) \in \mathcal{M} \Big\}$$

enaki.

Dokaz. ($C_1 \subseteq C_2$) Naj bo $c(x) \in C_1$, tj. $c(x) = a(x)g(x)$ za nek polinom $a(x) = \sum_{j=0}^{k-1} a_j x^j$ stopnje manjše od k, kjer je $\boldsymbol{a} = (a_0 \dots a_{k-1}) \in \mathcal{M}$. S pomočjo inverzne formule (2.37) za diskretno Fourierovo transformacijo poiščimo vektor $\boldsymbol{m}$, ki se z diskretno Fourierovo transformacijo preslika v vektor $\boldsymbol{c}$, asociiran polinomu $c(x)$:

$$m(x) = \frac{1}{n}\sum_{i=0}^{n-1} c(\alpha^{-i})x^i = \frac{1}{n}\sum_{i=0}^{n-1} a(\alpha^{-i})g(\alpha^{-i})x^i = \frac{1}{n}\sum_{i=0}^{k-1} a(\alpha^{-i})g(\alpha^{-i})x^i\,.$$

Pri zadnjem enačaju pa smo upoštevali, da so $\alpha^1, \dots, \alpha^{n-k}$ ničle polinoma $g(x)$. Ker je $\deg m(x) \le k-1$, je $c(x) \in C_2$.

($C_2 \subseteq C_1$) Naj bo sedaj $c(x) \in C_2$, kar pomeni $c(x) = \sum_{i=0}^{n-1} m(\alpha^i)x^i$ za nek $m(x) = \sum_{j=0}^{k-1} m_j x^j$, $\boldsymbol{m} = (m_0 \dots m_{k-1}) \in \mathcal{M}$. Izračunajmo vrednosti $c(\alpha^t)$, $k \le t \le n-1$,

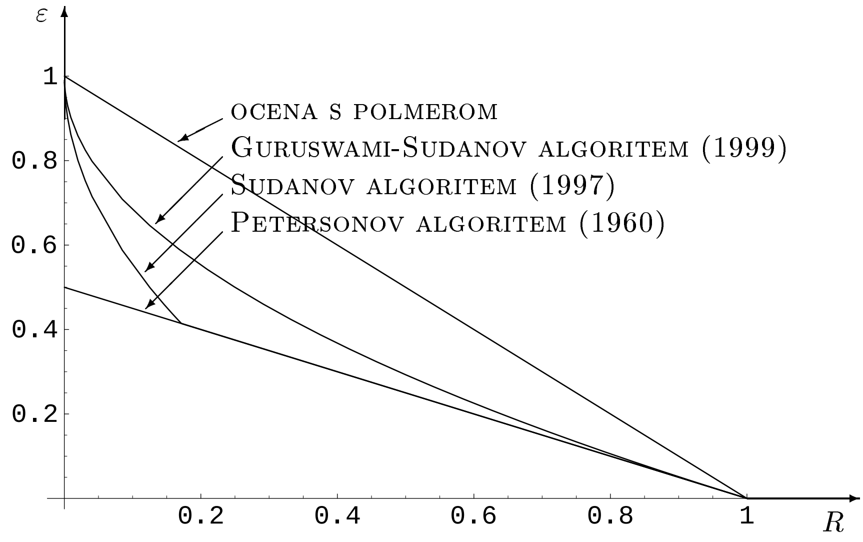
$$c(\alpha^t) = \sum_{i=0}^{n-1} m(\alpha^i)\alpha^{ti} = \sum_{i=0}^{n-1}\sum_{j=0}^{k-1} m_j \alpha^{ij}\alpha^{ti} = \sum_{j=0}^{k-1} m_j \sum_{i=0}^{n-1} \alpha^{(j+t)i} = \sum_{j=0}^{k-1} m_j \frac{\alpha^{(j+t)n}-1}{\alpha^{j+t}-1}\,.$$

Ker je $\alpha^n = 1$, velja $\alpha^{(j+t)n} = 1$ in iz zgornje zveze sledi $c(\alpha^t) = 0$. Ker to velja za vsak $t \in \{k, \dots, n-1\}$, je $c(x)$ deljiv z $g(x)$ in zato $c(x) \in C_1$. ■

Guruswami-Sudanov algoritem za dekodiranje RS kod v sezname

Do leta 1997 ni bila poznana nobena koda, za katero bi obstajal polinomski algoritem za dekodiranje v sezname, ki bi bil učinkovitejši od nedvoumnega dekodiranja (definiciji sta na str. 14). Če želimo popraviti vsaj $e < d$ napak (**ocena s polmerom**), potem po Singletonovi oceni $d-1 \le n-k$ velja $e \le n-k$, oziroma $\varepsilon \le 1-R$, kjer je $R = k/n$ stopnja kode in $\varepsilon = e/n$ **relativno število napak**, ki jih popravi algoritem. Sidelnikov je leta 1994 predstavil algoritem, ki popravi $(n-k)/2 + O(\log n)$ napak. V tem primeru je $\varepsilon = (1-R)/2 + o(1)$, kar je asimptotično enako kot pri Petersonovemu algoritmu. Leta 1997 pa je Sudan [88] predstavil prvi algoritem za dekodiranje RS kod, ki za $R > \frac{1}{3}$ popravi asimptotično več napak kot Petersonov (Berlekampov) algoritem. Pri dani stopnji R za Sudanov algoritem velja $\varepsilon > \max\{1-\sqrt{2R}, \frac{1}{2}(1-R)\}$, za Petersonov algoritem pa $\varepsilon = \frac{1}{2}(1-R)$. Leta 1999 pa sta Guruswami in Sudan [35, 36] objavila algoritem, ki je pri vsaki stopnji R boljši od Petersonovega, saj za njun algoritem velja $\varepsilon = 1-\sqrt{R}$. Primerjave med kapacitetami algoritmov si lahko ogledamo na grafu slika 2.6.

V tem razdelku si bomo ogledali Guruswami-Sudanov algoritem. Avtorja sta s pomočjo prevedbe na faktorizacijo, oz. iskanje ničel polinoma dveh spremenljivk, rešila **problem prilagajanja krivulje** (ang. curve-fitting), ki ga v članku imenujeta tudi **problem rekonstrukcije polinoma** (ang. polynomial reconstruction problem). Za dane točke $(x_1, y_1), \dots, (x_n, y_n) \in \mathbb{F}^2$, dano stopnjo k in parameter za število napak e je potrebno poiskati vse polinome $p(x) \in \mathbb{F}[x]$ stopnje največ k, za katere je $p(x_i) = y_i$ za vsaj $t = n-e$ različnih indeksov $i \in \{1, \dots, n\}$. Očitno lahko problem dekodiranja RS kod v



Slika 2.6: Relativno število napak $\varepsilon = e/n$, ki jih popravijo posamezni algoritmi pri stopnji $R = k/n$ Reed-Solomonove (n, k)-kode

sezname preprosto prevedemo na problem rekonstrukcije polinoma. Guruswami-Sudanov algoritem reši ta problem za $e < n - \sqrt{nk}$. Tako kot v Sudan [88] sta avtorja tudi tu problem prilagajanja krivulje prevedla na problem faktorizacije polinomov dveh spremenljivk nad obsegom $\mathbb{F}$. Ta problem lahko v primeru končnih obsegov rešimo deterministično v polinomskem času glede na velikost obsega $\mathbb{F}$ ali pa z verjetnostnim algoritmom v polinomskem času glede na logaritem velikosti obsega $\mathbb{F}$, glej Grigoriev [34], Kaltofen [48] in Kaltofen [49]. Pri končnih obsegih je najučinkovitejši Shokrollahijev [83] algoritem.

Omenimo še dva rezultata, ki kažeta na potencialne omejitve za velikost parametra e pri iskanju algoritmov polinomske časovne zahtevnosti za dekodiranje v sezname. Goldreich et al. [28] so pokazali, da za $e < n - \sqrt{n(n-d)}$ algoritem za dekodiranje v sezname vrne glede na n polinomsko mnogo kodnih besed, ki so od prejetega vektorja oddaljene za manj kot e. Justesen [47] pa je iskal največje možno število napak $e_{c,d,n}$ pri katerem bi algoritem za dekodiranje v sezname pri dani dolžini n in razdalji kode d vrnil največ c rešitev. Pokazal je, da če fiksiramo c in razmerje d/n, potem $e_{c,d,n}/n$ konvergira proti $1 - \sqrt{1 - d/n}$.

V algoritmu bomo iskali polinom $Q(x, y)$, za katerega velja, da gre krivulja $Q(x, y) = 0$ skozi vse vhodne točke (x_i, y_i). Zahtevali bomo še več - da je vsaka od vhodnih točk **singularnost** stopnje r, kar pomeni, da gre krivulja skozi vsako točko vsaj r-krat. Če izhodišče koordinatnega sistema postavimo v točko (α, β), se enačba krivulje $Q(x, y) = 0$ transformira v $Q_{\alpha,\beta}(x, y) = 0$, kjer je $Q_{\alpha,\beta}(x, y) = Q(x + \alpha, y + \beta)$. Da bomo dosegli primerno singularnost, bomo definirali uteženo stopnjo monomov in za polinom $Q_{\alpha,\beta}(x, y)$ zahtevali, da ima vsak od monomov primerno veliko uteženo stopnjo.

Lema 36. *Koeficiente polinoma $Q_{\alpha,\beta}(x, y) = \sum_{i\geq 0} \sum_{j\geq 0} (q_{\alpha,\beta})_{i,j} x^i y^j$ določimo iz koeficientov polinoma $Q(x, y) = \sum_{i\geq 0} \sum_{j\geq 0} q_{i,j} x^i y^j$ s pomočjo formule*

$$(q_{\alpha,\beta})_{i,j} = \sum_{i'\geq i} \sum_{j'\geq j} \binom{i'}{i} \binom{j'}{j} q_{i',j'} \alpha^{i'-i} \beta^{j'-j} .$$

Vidimo, da nove koeficiente dobimo s pomočjo linearne transformacije starih.

Dokaz. Z upoštevanjem definicije $Q(x,y) = \sum_i \sum_j q_{i,j} x^i y^j$ dobimo

$$\begin{aligned} Q_{\alpha,\beta}(x,y) &= Q(x+\alpha, y+\beta) = \sum_{i'} \sum_{j'} q_{i',j'} (x+\alpha)^{i'} (y+\beta)^{j'} = \\ &= \sum_{i'} \sum_{j'} q_{i',j'} \sum_{i \le i'} \binom{i'}{i} x^i \alpha^{i'-i} \sum_{j \le j'} \binom{j'}{j} y^j \beta^{j'-j} . \end{aligned}$$

Sedaj upoštevamo $\sum_{i'=0}^{n} \sum_{i=0}^{i'} = \sum_{i=0}^{n} \sum_{i'=i}^{n}$ in podobno še za j. Ker so vsote končne, lahko menjamo vrstni red sumiranja po i in j. Dobimo

$$Q_{\alpha,\beta}(x,y) = \sum_i \sum_j \Big(\sum_{i' \ge i} \sum_{j' \ge j} \binom{i'}{i} \binom{j'}{j} q_{i',j'} \alpha^{i'-i} \beta^{j'-j} \Big) x^i y^j .$$

Faktor $x^i y^j$ smo postavili pred vsoti po i' in j', saj je neodvisen od i' in j'. ■

Za nenegativni uteži w_1 in w_2 definiramo (w_1, w_2)**-uteženo stopnjo monoma** $x^i y^j$ kot $iw_1 + jw_2$. Za polinom $Q(x,y)$ pa jo definiramo kot največjo (w_1, w_2)-uteženo stopnjo njegovih neničelnih monomov. Sedaj si na sliki 2.7 lahko ogledamo algoritem za problem rekonstrukcije polinoma.

Algoritem za problem rekonstrukcije polinoma (dekodiranje v sezname za RS kode)
Vhodni podatki: $n, k, t, \{(x_i, y_i)\}_{i=1}^n$, kjer so $(x_i, y_i) \in \mathbb{F}^2$ paroma različne točke. Inicializacija: $r = 1 + \left\lfloor \dfrac{kn + \sqrt{k^2n^2 + 4(t^2 - kn)}}{2(t^2 - kn)} \right\rfloor,$ $\ell = rt - 1.$ 1. Poišči polinom $Q(x,y)$, tako da njegova $(1,k)$-utežena stopnja ni večja od ℓ, za koeficiente q_{j_1,j_2}, kjer je $j_1, j_2 \ge 0$ in $j_1 + kj_2 \le \ell$, pa velja: i) vsaj en koeficient je neničeln ($Q(x,y) \neq 0$), ii) za vsak i, $1 \le i \le n$, velja: če je $Q^{(i)}$ zamik Q-ja v točko (x_i, y_i), torej $Q^{(i)}(x,y) = Q(x + x_i, y + y_i)$, potem so koeficienti polinoma $Q^{(i)}$ pri monomih stopnje manjše od r enaki 0 (vsaka točka (x_i, y_i), $1 \le i \le n$, ima dovolj veliko singularnost). To pomeni: za vsak i, $1 \le i \le n$, in za vsak $j_1, j_2 \ge 0$, $j_1 + j_2 < r$, velja $$\sum_{j_1' \ge j_1} \sum_{j_2' \ge j_2} \binom{j_1'}{j_1} \binom{j_2'}{j_2} q_{j_1',j_2'} x_i^{j_1'-j_1} y_i^{j_2'-j_2} = 0 \qquad (2.38)$$ (koeficient polinoma $Q(x + x_i, y + y_i)$ pred $x^{j_1} y^{j_2}$ je za $j_1 + j_2 < r$ enak 0). 2. Poišči vse polinome $p(x) \in \mathbb{F}_q[x]$ stopnje največ k, ki so ničle polinoma $Q(x,y)$. To pomeni, da je $y - p(x)$ faktor od $Q(x,y)$. Za vsak najden polinom $p(x)$ preveri, ali je $p(x_i) = y_i$ za vsaj t vrednosti i, $1 \le i \le n$. Če je, ga dodaj v izhodni seznam.

Slika 2.7: Algoritem za rekonstrukcijo polinoma (dekodiranje v sezname za RS kode)

Trditev 37. [35, Theorem 7] *Naj bo n dolžina RS kode dimenzije k in naj bo $t = n - e$, kjer je e parameter za število napak v problemu rekonstrukcije polinoma. Če je $t > \sqrt{kn}$, potem pri vsakem obsegu $\mathbb{F}$, za katerega obstaja algoritem polinomske časovne zahtevnosti za iskanje ničel polinomov ene spremenljivke nad obsegom racionalnih funkcij $\mathbb{F}(x)$, Guruswami-Sudanov algoritem reši problem rekonstrukcije polinoma v polinomskem času glede na n.*

Dokaz. Ker je $t^2 > kn$, pokažimo, da za parametra r in ℓ, kot sta definirana v algoritmu, velja

$$n\binom{r+1}{2} < \frac{\ell(\ell+2)}{2k}. \tag{2.39}$$

Z upoštevanjem definicije parametra $\ell = rt - 1$ preoblikujemo neenakost v

$$\begin{aligned} n\binom{r+1}{2} &< \frac{(rt-1)(rt+1)}{2k} \iff \\ k\,n\,r(r+1) < r^2t^2 - 1 &\iff r^2(t^2 - kn) - k\,n\,r - 1 > 0\,. \end{aligned}$$

Večji od korenov enačbe $r^2(t^2 - kn) - knr - 1 = 0$ je $r_1 = \frac{kn+\sqrt{k^2n^2+4(t^2-kn)}}{2(t^2-kn)}$ in ker je parameter r v algoritmu večji od r_1, zadnja neenakost gotovo velja, torej tudi prva.

S pomočjo neenakosti (2.39) lahko sedaj pokažemo obstoj polinoma $Q(x,y)$, ki ga iščemo v prvem koraku algoritma. Določimo ga lahko v polinomskem času z reševanjem linearnega sistema (2.38). Homogen linearni sistem (2.38) ima neničelno rešitev natanko takrat, ko je rang sistema strogo manjši od števila neznank. Rang sistema je manjši ali enak številu enačb. Na izbiro imamo n različnih i-jev in $\binom{r+1}{2}$ različnih parov (j_1, j_2), torej je enačb največ $n\binom{r+1}{2}$. Število neznank pa je največ toliko, kot je lahko monomov v polinomu $(1,k)$-utežene stopnje največ ℓ in to je

$$\begin{aligned} \sum_{j_2=0}^{\lfloor \frac{\ell}{k} \rfloor} \sum_{j_1=0}^{\ell - kj_2} 1 = \sum_{j_2=0}^{\lfloor \frac{\ell}{k} \rfloor} \ell + 1 - kj_2 &= \\ &= (\ell+1)\left(\left\lfloor \frac{\ell}{k} \right\rfloor + 1\right) - \frac{k}{2}\left\lfloor \frac{\ell}{k} \right\rfloor \left(\left\lfloor \frac{\ell}{k} \right\rfloor + 1\right) \geq \\ &\geq \left(\left\lfloor \frac{\ell}{k} \right\rfloor + 1\right)\left(\ell + 1 - \frac{\ell}{2}\right) \geq \frac{\ell}{k}\frac{\ell+2}{2}\,. \end{aligned}$$

Ker velja (2.39), je torej rang sistema ustrezno majhen, kar nam zagotavlja obstoj netrivialne rešitve sistema (2.38).

S pomočjo neenakosti $rt > \ell$, ki velja po definiciji parametra $\ell = rt - 1$, pa bomo pokazali, da $y - p(x)$ deli polinom $Q(x,y)$ za vsak polinom $p(x)$, ki je stopnje največ k ter zanj velja $y_i = p(x_i)$ za vsaj t indeksov i, $1 \leq i \leq n$. Ker je $(1,k)$-utežena stopnja polinoma $Q(x,y)$ enaka največ ℓ, je stopnja polinoma $g(x)$, definiranega kot $g(x) = Q(x, p(x))$, največ ℓ. Označimo s S množico tistih indeksov i, pri katerih je $y_i = p(x_i)$. Za $i \in S$ pokažimo najprej, da $(x - x_i)^r$ deli polinom $g(x)$. Definirajmo $p'(x) = p(x + x_i) - y_i$. Pokazati moramo, da x^r deli $g(x + x_i)$. Ker je $p'(0) = 0$, lahko definiramo polinom $p''(x)$ tako, da je $p'(x) = xp''(x)$. Definirajmo še $g'(x) = Q^{(i)}(x, p'(x))$ in pokažimo, da je $g'(x - x_i) = g(x)$:

$$g(x) = Q(x, p(x)) = Q^{(i)}(x - x_i, p(x) - y_i) = Q^{(i)}(x - x_i, p'(x - x_i)) = g'(x - x_i)\,.$$

Glede na konstrukcijo iz algoritma polinom $Q^{(i)}$ nima neničelnih monomov stopnje manjše od r. To pomeni, da je polinom

$$g(x+x_i) = g'(x) = Q^{(i)}(x, p'(x)) = Q^{(i)}(x, xp''(x))$$

deljiv z x^r. Torej $(x - x_i)^r$ deli polinom $g(x)$ za vsak $t \in S$ in ker je pri teh indeksih $x_i \neq x_j$ za $1 \leq i < j \leq n$, sicer bi bilo $(x_i, y_i) = (x_i, p(x_i)) = (x_j, p(x_j)) = (x_j, y_j)$, je polinom $g(x)$ deljiv s $\prod_{i \in S}(x - x_i)^r$. Po predpostavki je $|S| \geq t$, torej smo našli polinom $\prod_{i \in S}(x - x_i)^r$ stopnje vsaj rt, ki deli polinom $g(x)$ stopnje $\ell < rt$. To je možno le, če je $g(x) = 0$, kar pomeni, da je $Q(x, p(x)) = 0$. Z algoritmom deljenja dobimo, da $y - p(x)$ deli $Q(x, y)$. ∎

Posledica 38. *Reed-Solomonovo kodo stopnje R z zahtevano stopnjo odpravljanja napak $\varepsilon = 1 - \sqrt{R}$ lahko dekodiramo v sezname v polinomskem času glede na dolžino kode in velikost abecede $\Sigma = \mathbb{F}$.*

Poglavje 3

Varnost kriptosistemov

Potreba po učinkovitem skrivanju informacij je gotovo stara kot človeštvo. Najboljši pregled nad razvojem kriptografije nam gotovo nudi Kahnova monografija *The Codebreakers*, ki opisuje kriptosisteme od uporabe pri starih Egipčanih pa vse do leta 1963, ko je knjiga izšla. Zelo pomembno vlogo je kriptografija odigrala med obema svetovnima vojnama, v povojnem obdobju, to je obdobju hladne vojne, pa na diplomatskem področju. Sedaj pa je seveda nepogrešljiva pri e-storitvah na internetu, kot so e-bančništvo, e-trgovina, ponudniki e-poštnih storitev ipd.

Zelo uporaben priročnik na področju kriptografije so napisali Menezes, Oorschot in Vanstone [64], v tem trenutku najnovejša referenca za pregled stanja na področju kriptografije pa je gotovo zadnje poročilo [69] kriptografskega natečaja NESSIE evropske unije. Primeren učbenik za učenje osnov predstavlja Stinson [87], lahko pa tudi Hankerson et al. [39].

V prvem razdelku bomo predstavili pregled področij kriptografije in opisali najpomembnejše dogodke v njenem razvoju v prejšnjem stoletju. Nato se bomo omejili na šifrirne sisteme ter v naslednjih dveh razdelkih opredelili stopnje varnosti ter opisali, kakšna je brezpogojna varnost in dokazali, kako jo lahko dosežemo. V četrtem in petem razdelku bomo opisali potrebne pogoje za varnost tokovnih šifer ter psevdonaključnih zaporedij bitov. V zadnjem razdelku je opisana ena od možnih klasifikacij povratnih pomičnih registrov, ki so ponavadi predstavniki hitrih tokovnih šifer.

3.1 Kriptografija in informacijska varnost

Med najosnovnejše in najvarnejše kriptografske algoritme gotovo še vedno sodi algoritem DES, oziroma izboljšava trojni DES. DES[1] je začel razvijati Feistel pri IBM v začetku 70-ih, leta 1977 pa je postal oden od FIPS[2] standardov, sprejetih pri NIST[3]. Delovanje DES-a si lahko preberemo v Stinson [87], kjer so opisani tudi najbolj znani napadi nanj, kot sta diferenčna in linearna kriptoanaliza. O izboru S škatel v algoritmu DES je bilo mnogo dvomov, zakaj take, ali njihov izbor omogoča NSA[4] vpogled v šifrirana besedila skozi kakšna stranska vrata. Nekaj dvomov o tem sta odpravila tudi Golomb in Gong [32], ko sta pokazala da ima vseh osem uporabljenih S-škatel, ki so permutacije, v polinomskem

[1]DES – Data Encryption Standard
[2]FIPS – Federal Information Processing Standards
[3]NIST – National Institute of Standards and Technology
[4]NSA – National Security Agency

zapisu skoraj vse koeficiente neničelne. Kljub dvomom se je DES obdržal kot standard.

Kljub temu, da je NIST sprejel že nov standard AES za bločne šifre, le ta še ni zaživel v praksi. AES je bil izbran na ameriškem natečaju za izbor bločne šifre. Na natečaj so bili vabljeni kriptografi s celega sveta, ki so in predlagali nove šifre, kakor jih tudi analizirali in ocenjevali. Leta 2000 je bila izbrana šifra Rijndael, ki se sedaj imenuje AES (Advanced Encryption Standard). Šifra ima več matematične strukture kot DES, je odporna proti diferenčnem in linearnem napadu, vendar pa obstaja možnost drugih napadov. Podoben projekt, projekt NESSIE, je od leta 2000 do leta 2003 izvajala Evropska unija. Vendar pa se ni omejila le na bločne šifre, ampak na vse kriptografske algoritme. Izbranih algoritmov ni definirala kot standarde, pač pa jih le priporoča za uporabo.

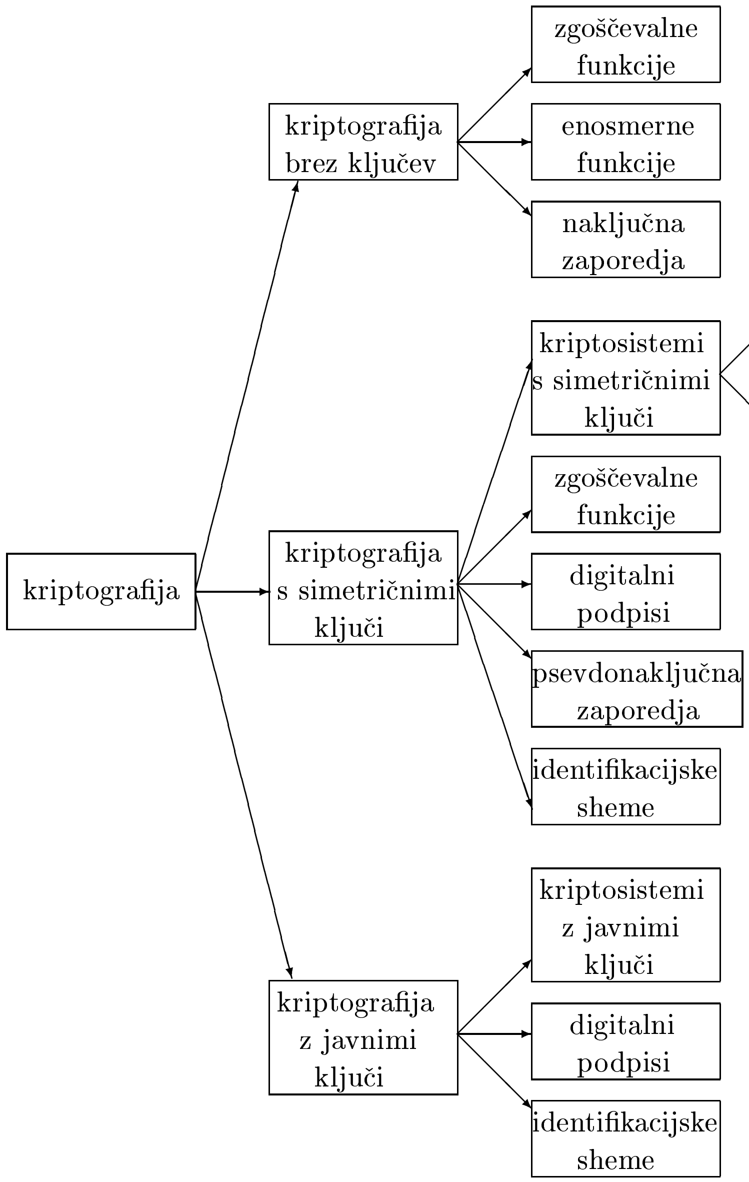
Kot smo že omenili v uvodu, predstavlja gotovo največji dogodek v kriptografiji v prejšnjem stoletju zamisel koncepta javne kriptografije, ki sta ga leta 1976 predstavila Diffie and Hellman [21]. Prvo varno izvedbo pa so leta 1978 predstavili Rivest, Shamir in Adleman z algoritmom RSA, ki temelji na težavnosti problema faktorizacije celih števil, pogosto uporabljen pa je tudi leta 1985 ElGamalov kriptosistem, katerega varnost pa temelji na težavnosti izračuna diskretnega logaritma (DLP problem). Kriptografija z javnimi ključi je v svojem razvoju predstavila še možnost digitalnega podpisovanja digitalnih dokumentov. Pomemben del strukture podpisovanja predstavljajo enosmerne funkcije, katerih izračunljivost inverza ni možna v polinomskem času, glede na velikost argumenta. Osnovo za konstrukcijo nam predstavljajo različni še nerešeni NP problemi[5].

Seveda je področje kriptografije preobširno, da bi ga lahko celotnega opisali, zato je bralec vabljen k predlaganim učbenikom. Pregled lahko strnemo na sliki 3.1, kjer pa ne omenjamo kvantne kriptografije, ki je še vedno v povojih, saj še ni bilo izdelanega uporabnega kvantnega računalnika. Na tem področju se je izvajal tajni projekt v Los Alamosu, a so ga njegovo obsežnost občutno zmanjšali.

Pri posredovanju in delu z informacijami želimo imeti vgrajene mehanizme, kot so zasebnost, zaupnost, pristnost, celovitost ipd. Za zagotavljanje teh mehanizmov uporabljamo kriptografske tehnike, kot so šifriranje, digitalni podpisi, digitalni certifikati in druge ter seveda zakonodaja in tudi politika podjetij, ki morajo skrbeti, da je njihovo delovanje na področju informacijske varnosti vestno. To pomeni, da imajo izdelan načrt neprekinjenega poslovanja, varovanje informacijskega sistema pred virusi, vdori ipd ter imajo seveda kontrolo znotraj sistema, saj v večini primerov uhajajo informacije iz notranjih virov podjetij. V standardu ISO 17799 oz. njegovi britanski različici BS 7799 je opisano, kaj vse je potrebno storiti za zagotovitev varnosti v podjetju.

Za razvoj kriptografije skrbijo znanstveniki, ki delajo na uneverzah ter v raziskovalnih laboratorijih večjih podjetij. Mnogo pa jih deluje na varnostnih agencijah posameznih držav. Tako je na primer britanska obveščevalna služba uporabljala sistem RSA, še predno je bil javno predstavljen. Svoje delo znanstveniki ponavadi objavijo v revijah, kot so na primer *IEEE Transactions of Information Theory*, *IEEE Transactions on Computers* ali pa *Journal of Cryptology* in druge. Mnogi svoje delo že pred objavo predstavijo na kateri od kriptografskih konferenc, kot so Eurocrypt, Crypto (ZDA), Asiacrypt, Indocrypt in druge. Prav tako pa so zelo dobrodošli tudi projekti, kot sta bila AES in NESSIE.

[5] NP – nedeterministično polinomski; ponujeno rešitev problema je možno preveriti v polinomskem času



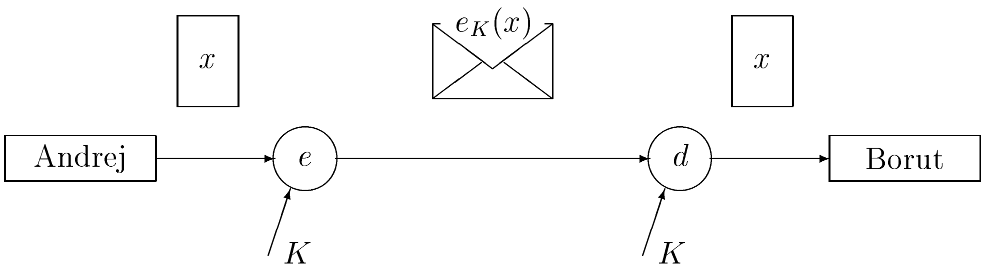
Slika 3.1: Področja uporabe kriptografije

3.2 Stopnje varnosti kriptosistemov

Osnovni cilj kriptografije je omogočiti dvema osebama, recimo Andreju in Borutu, komuniciranje preko neke povezave, npr. e-pošte, na tak način, da opazovalec, npr. Oskar, ne bi mogel razumeti, kakšne informacije si izmenjujeta. Tekst x, ki ga Andrej želi poslati Borutu, bomo imenovali **čistopis**. To je lahko običajen tekst, lahko pa so računalniške datoteke slik, zvoka ipd. S prej dogovorjenim ključem K, ki ga poznata le Andrej in Borut, bo Andrej sporočilo zašifriral s postopkom e_K in ga nato poslal Borutu. Zašifrirano sporočilo imenujemo **tajnopis** in ga Borut lahko dešifrira s pomočjo dešifrirnega postopka d_K prirejenega ključu K. Ker Oskar ne pozna ključa K, pravimo, da je naš postopek **kriptosistem s privatnim ključem** in ga opišemo kot peterico $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ za katero velja:

1. $\mathcal{P}$ je končna množica možnih čistopisov (plaintext),
2. $\mathcal{C}$ je končna množica možnih tajnopisov (ciphertext),
3. $\mathcal{K}$ je končna množica možnih **ključev** in
4. za vsak ključ $K \in \mathcal{K}$ imamo natanko določen **šifrirni postopek** $e_K \in \mathcal{E}$ (encryption rule) in ustrezen **dešifrirni postopek** $d_K \in \mathcal{D}$ (decryption rule), kjer sta $e_K : \mathcal{P} \to \mathcal{C}$ in $d_K : \mathcal{C} \to \mathcal{P}$ taki funkciji, da je $d_K(e_K(x)) = x$ za vsak čistopis $x \in \mathcal{P}$.

Najpomembnejša je četrta točka, ki pravi, da ko Borut dešifrira sporočilo $e_K(x)$, mora dobiti prvotno sporočilo x, ki mu ga je poslal Andrej. Slika 3.2 nam prikazuje celotno shemo delovanja kriptosistema.



Slika 3.2: Shema kriptosistema

V splošnem velja kriptosistem za varnega, če Oskar, kljub temu da pozna delovanje sistema, ne more ugotoviti ključa, s katerim je bilo sporočilo šifrirano. V tem primeru pravimo, da je kriptosistem **varen po Kerckhoffovem principu**. Za kriptosisteme, ki so danes v komercialni uporabi, to pomeni, da ne obstaja metoda, ki bi za cel velikostni razred hitreje, kot je preskus vseh možnih ključev, ugotovila dejanski ključ sporočila. Če z n označimo dolžino zapisa ključa, je velikost prostora ključev navadno enaka 2^n, zato preiskus vseh ključev v povprečju traja $2^n/2$ časovnih enot. Pravimo, da je kriptosistem varen, če ne obstaja metoda za ugotovitev ključa, ki bi bila bistveno hitrejša od preizkusa vseh ključev. Glede na podatke, ki jih dobi napadalec Oskar za ugotavljanje ključa, lahko napade na kriptosistem razdelimo na več težavnostnih stopenj:

1. **samo tajnopis**: Oskar pozna tajnopis x,
2. **poznani čistopis**: Oskar pozna čistopis x in ustrezen tajnopis $y = e_K(x)$,
3. **izbrani čistopis**: Oskar ima začasno na voljo šifrirni stroj (ki uporablja funkcijo e_K, ki je Oskar ne pozna) in za poljubno izbran čistopis x lahko določi ustrezen tajnopis $y = e_K(x)$,
4. **izbrani tajnopis**: Oskar ima začasno na voljo dešifrirni stroj (ki uporablja funkcijo d_K, ki je Oskar ne pozna) in za poljubno izbran tajnopis y lahko določi ustrezen čistopis $x = d_K(y)$.

Za varen prenos velikih količin podatkov potrebujemo kriptosistem, ki je odporen proti vsem zgoraj naštetim stopnjam napada.

V daljnji preteklosti so uporabljali preproste kriptosisteme, kot sta na primer **substitucijski tajnopis** in Vigenèrejev tajnopis. Šifriranje s prvim je uporaba fiksne permutacije na množici črk na vsaki črki čistopisa posebej. Pri **Vigenèrejevem tajnopisu** pa prištejemo čistopisu ustrezno ponavljanje ključa, kakor je razvidno na primeru, ko šifriramo besedilo $information$ s ključem key

$$\begin{array}{rl} & information \\ + & keykeykeyke \end{array} \quad \sim \quad \begin{array}{rrrrrrrrrrrr} & 8 & 13 & 5 & 14 & 17 & 12 & 0 & 19 & 8 & 14 & 13 \\ + & 10 & 4 & 24 & 10 & 4 & 24 & 10 & 4 & 24 & 10 & 4 \\ & - & - & - & - & - & - & - & - & - & - & - \\ & 18 & 17 & 3 & 24 & 21 & 10 & 10 & 23 & 6 & 24 & 17 \end{array}$$

in ustrezni tajnopis je potem $RQCXUJJWFXQ$.

Njuna slabost je bila, da lahko na podlagi pričakovane frekvence pojavljanja črk v jeziku, v katerem je napisan čistopis, določimo skrivni ključ. Oba napada sta podrobno opisana v Stinson [87, sect. 1.2] in tudi v Novak [71, poglavje 1.2].

Brezpogojna in računska varnost

Predstavili bomo dva osnovna pogleda na varnost kriptosistemov, kot jo je predstavil Claude Shannon [82] leta 1949, povzetek najdemo tudi v Stinson [87, Chapter 2] ali pa Novak [71]. Delo je imelo velik vpliv na znanstven pristop k kriptografiji.

Za dan kriptosistem pravimo, da je pri dani stopnji napada **računsko varen**, če najboljši algoritem za razbitje opravi vsaj N operacij, kjer je N neko predpisano veliko število in se nanaša na zmogljivost današnjih računalnikov. V praksi nismo tako formalni in pravimo, da je kriptosistem **računsko varen**, če najboljši znan algoritem za razbitje opravi vsaj N operacij, vendar pa to še zdaleč ni dolgoročno zagotovilo za varnost. Veliko današnjih kriptosistemov pa je takih, da obstaja njihovo razbitje le, če obstaja algoritem za rešitev nekega težkega matematičnega problema. To so navadno t.i. NP-polni problemi, znani 100 in več let, glej Petkovšek in Pisanski [76]. Na drugi strani pa so **brezpogojno varni** kriptosistemi, ki jih pri dani stopnji napada ni mogoče razbiti niti z neomejeno zmogljivimi računalniki. Glede na to, kakšen nivo varnosti zagotavlja avtor kriptosistema, lahko definiramo pojem **razbitja**. V drugem primeru je to lahko tudi preskus vseh ključev, četudi ga je z današnjo tehniko nemogoče opraviti v kratkem času.

Opišimo sedaj, kaj pomeni, da je dan kriptosistem brezpogojno varen pri napadu s poznanim tajnopisom. V tem primeru je razbitje poljuben postopek, ki nam za dan tajnopis y da nek čistopis x, za katerega zagotavlja, da je verjetnost, da je x rezultat dešifriranja tajnopisa y pri iskanem ključu K, večja od verjetnosti, da za poljubno šifriranje izberemo čistopis x, skratka

$$P_{\mathcal{P}}(x|y) > P_{\mathcal{P}}(x)\,.$$

Zato pravimo, da kriptosistem zagotovlja **popolno tajnost**, če je

$$P_{\mathcal{P}}(x|y) = P_{\mathcal{P}}(x)$$

za vsak par $x \in \mathcal{P}$ in $y \in \mathcal{C}$, kjer je $y = e_K(x)$. V Stinson [87, Chapter 2] si lahko preberemo dokaz, da nam pri pogoju $|\mathcal{K}| = |\mathcal{C}| = |\mathcal{P}| = n$ kriptosistem zagotavlja popolno tajnost, natanko tedaj ko je vsak ključ izbran z enako verjetnostjo, to pomeni $P_{\mathcal{K}}(K) = 1/n$ za vsak $K \in \mathcal{K}$, in ko za vsak par $x \in \mathcal{P}$ in $y \in \mathcal{C}$ obstaja natanko en ključ $K \in \mathcal{K}$, za katerega velja $y = e_K(x)$.

Primer kriptosistema, ki zagotavlja popolno tajnost, je **Vernamov enkratni ščit**, pri katerem šifriramo po istem principu kot z Vigenèrejev tajnopisom, le da s ključem, ki je enako dolg kot čistopis. Enkratni ščit je Gilbert Vernam patentiral že leta 1917 in so ga uporabljali pri prenosu telegrafskih sporočil, njegovo brezpogojno varnost pa je dokazala šele Shannonova teorija 30 let kasneje. Ideja Vigenèrejevega kriptosistema je danes najpogostejša pri konstrukciji kriptosistemov. Za uporabo v računalnikih črke zamenjamo z biti, seštevanje in odštevanje po modulu 2 pa je bolj poznano kot XOR seštevanje, ki ga označimo z $\oplus$.

Vendar pa so vsi kriptosistemi, ki zagotavljajo popolno tajnost, zaradi dejstva, da mora biti $|\mathcal{K}| \geq |\mathcal{P}|$, uporabni za komercialne namene le na način, da namesto varne izmenjave velike količine ključa uporabljamo psevdonaključna zaporedja bitov, s čemer se odrečemo brezpogojni varnosti in se zadovoljimo z računsko varnostjo.

3.3 Tokovni tajnopisi

Tokovnim tajnopisom pravimo tudi tokovne šifre. Od bločnih šifer se ločijo po tem, da šifrirajo vsak znak čistopisa posebej ter da se šifriraje spreminja s časom v odvisnosti od predhodnih znakov, ponavadi bitov. Ponavadi so tokovne šifre hitrejše od bločnih, ni pa to nujno, saj so nekatere izpeljane iz bločnih šifer. Zelo malo jih je javno znanih, saj lastniki v mnogih primerih tretirajo njihovo konstrukcijo kot tajen podatek, kar pa je v nasprotju s Kerckhoffovim principom.

Prikažimo aksiomatsko definicijo tokovnih tajnopisov ter njihov zelo pogost primer uporabe, ki spominja na enkratni ščit, to je šifriranje s pomočjo generatorja psevdonaključnih zaporedij bitov. Tak kriptosistem nam ne more zagotavljati brezpogojne varnosti, lahko pa zagotavlja računsko varnost. Le to bomo podali z definicijo kriptografsko varnega generatorja psevdonaključnih zaporedij bitov.

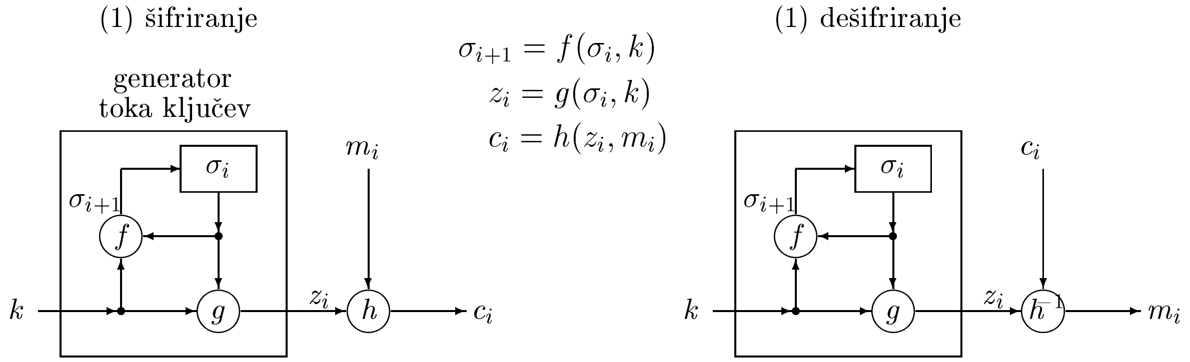
Tokovni tajnopis je sedmerica $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{F}, \mathcal{E}, \mathcal{D})$ za katero velja:

1. $\mathcal{P}$ je končna množica možnih **čistopisov**,
2. $\mathcal{C}$ je končna množica možnih **tajnopisov**,
3. $\mathcal{K}$ je končna množica možnih **ključev**,
4. $\mathcal{L}$ je končna množica **tokovne abecede**,
5. $\mathcal{F} = (f_1, f_2, \ldots)$ je **generator toka ključev**:
$$f_i : \mathcal{K} \times \mathcal{P}^{i-1} \to \mathcal{L} \qquad \text{za } i \geq 1\,,$$
6. za vsak tok ključev $z \in \mathcal{L}$ imamo natanko določen **šifrirni postopek** $e_z \in \mathcal{E}$ in ustrezen **dešifrirni postopek** $d_z \in \mathcal{D}$, kjer sta $e_z : \mathcal{P} \to \mathcal{C}$ in $d_z : \mathcal{C} \to \mathcal{P}$ taki funkciji, da je $d_z(e_z(x)) = x$ za vsak čistopis $x \in \mathcal{P}$.

V ozadju je torej postopek, ki nam na podlagi enega ključa k zgradi tok ključev $\boldsymbol{z} = z_1 \ldots z_n$, s katerimi potem šifriramo 'tok' čistopisa $\boldsymbol{x} = x_1 \ldots x_n$, kjer so $x_i \in \mathcal{P}$ in $\mathcal{P}$ je znakovna abeceda, ponavadi kar $\mathcal{P} = \mathbb{Z}_2$. S tem poskusimo zagotoviti vsaj računsko varnost tokovnega kriptosistema, saj vemo, da ni varno šifrirati več čistopisov z istim ključem.

Tokovne šifre lahko klasificiramo glede na sinhronizacijo, poznamo sinhrone in samosinhrone tokovne šifre. **Sinhrone** šifre so šifre, pri katerih je tok ključev neodvisen od toka čistopisa. Šifriranje in dešifriranje najlažje ponazorimo s shemo, predstavljeno na sliki 3.3.

Funkcija f nam spreminja notranje stanje generatorja, funkcija g pa nam glede na ključ



Slika 3.3: Shema sinhrone tokovne šifre

in notranje stanje na vsakem koraku vrne nov znak toka ključev. Kot bomo videli pri nelinearizacijah LFSR generatorja, je f mnogokrat odvisna le od notranjega stanja σ_i, od ključa k pa je odvisno le začetno notranje stanje σ_0. V primeru, ko je $\mathcal{P} = \mathbb{Z}_2$, nam tok ključev predstavlja zaporedje psevdonaključnih bitov $\boldsymbol{z} = z_1 \dots z_n$, s pomočjo katerega zašifriramo čistopis $\boldsymbol{x} = x_1 \dots x_n$, ponavadi s pravilom

$$y_i = x_i \oplus z_i \tag{3.1}$$

in tako dobimo tok tajnopisa $\boldsymbol{y} = y_1 \dots y_n$.

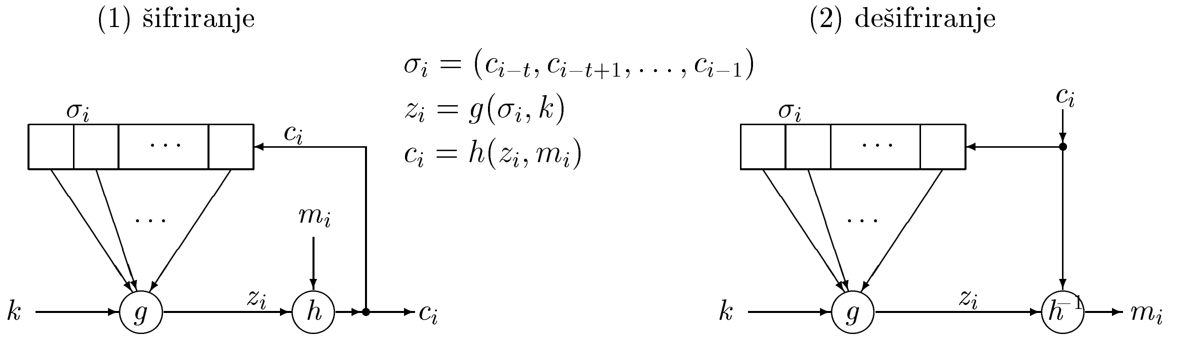
Pri uporabi sinhronih tokovnih šifer morata biti pošiljatelj in prejemnik sinhronizirana. To pomeni, da morata delovati z istim delom toka ključev na istoležnem delu čistopisa oz. tajnopisa in zato ne sme priti pri prenosu do izbrisa ali vrivanja podatkov, to je do izgube sinhronizacije. Ta lastnost omogoča možnost aktivnega napada na tak kriptosistem. Napadalec briše in vstavlja znake pri prenosu podatkov. Ker se pošiljatelj in prejemnik zaradi izgube sinhronizacije **resinhronizirata** in ponovita postopek pošiljanja od tistega dela naprej, kjer sta izgubila sinhronizacijo, to omogoči napadalcu, da vidi, kako spremembe tajnopisa vplivajo na njegovo nadaljevanje.

Asinhrone tokovne šifre pa so tokovne šifre, pri katerih je tok ključev odvisen le od ključa in $t \geq 1$ predhodnih znakov tajnopisa. Ponazorimo jih z enačbami in shemo, predstavljenimi na sliki 3.4.

Prednost asinhronih šifer je, da se lahko pri izgubi sinhronizacije same sinhronizirajo in je le nekaj znakov tajnopisa nedešifriranih. Je pa ta lastnost lahko tudi škodljiva, saj zaradi nje pri dešifriranju lahko ne zaznamo, da so bili določeni znaki izbrisani ali pa vrinjeni v tajnopis. Zato je potrebno v shemo vstaviti dodatne mehanizme, ki zagotavljajo celovitost in avtentičnost poslanih podatkov. Glede na to, da vsak znak tokovnega čistopisa vpliva na celotno nadaljevanje tajnopisa, lahko pričakujemo, da so asinhroni tajnopisi bolj odporni na napade, ki temeljijo na nenaključni porazdeljenosti znakov v čistopisu.

3.4 Generatorji naključnih zaporedij

V tem razdelku si bomo ogledali, kakšne lastnosti naj bi imeli generatorji naključnih zaporedij. Vsekakor mora te lastnosti imeti tok tajnopisa tokovnih šifer. Pri sinhronih šifrah



Slika 3.4: Shema asinhrone tokovne šifre

pa to pomeni, da če hočemo preprečiti napade tipa poznani/izbrani čistopis/tajnopis, moramo zagotoviti, da ima zahtevane lastnosti že generator toka ključev.

Generator naključnih zaporedij bitov (**RBG** - random bit generator) je naprava ali algoritem, ki sestavlja zaporedja naključnih (kar pomeni $P(\texttt{bit} = \texttt{1}) = \frac{1}{2}$), in med seboj neodvisnih bitov. Taki generatorji so lahko narejeni s strojno ali pa s programsko opremo. Bolj varni so prvi (šumni generator, ki npr. izkorišča termično šumenje tranzistorja), saj je pri njih opazovalec onemogočen, da bi opazoval, oziroma vplival na naključnost.

Generator psevdonaključnih zaporedij bitov (**PRBG** - pseudorandom bit generator) je determinističen algoritem, ki kot vhodne podatke dobi zaporedje bitov, generirano s pravim RBG, imenovano **seme**, dolžine k ter vrne zaporedje binarnih števil dolžine $\ell \gg k$, ki izgleda naključno. Kdaj zaporedje izgleda dovolj naključno, je težko opredeliti, saj je opredelitev odvisna tudi od namena uporabe generatorja. Lahko pa bi rekli, da takrat, ko ga statistični testi ne zavrnejo.

Izhodni podatki PRBG niso naključni, saj izmed vseh možnih zaporedij dolžine ℓ (2^ℓ jih je) lahko dobimo le en del teh, toliko kot je različnih semen (2^k). Seveda pa bi radi naredili tak PRBG, da nihče, ki bi opazoval izhodno zaporedje, ne bi mogel učinkovito ločiti to zaporedje od takega, ki bi ga dal pravi RBG. Da dosežemo zaupanje, da je PRBG res dober, mora ustrezati čimveč (vsem, ki jih naredimo) statističnim testom, ki preverjajo, ali ima opazovani PRBG take lastnosti, kot naj bi jih imeli pravi RBG. Takih testov je lahko toliko, kot je lastnosti RBG, torej neskončno. Naredimo pa seveda tiste, ki bi nam lahko pomagali pri razbitju algoritma. Nekaj je preprostih standardnih, t.i. osnovnih testov (npr: ničel naj bi bilo približno toliko kot enk), nekaj pa bolj zapletenih. V Novak [71, poglavje 4] so opisani testi iz programa Crypt-XS (osnovni testi, test linearnosti ter kompresijski test) in iz programa Diehard, ki ga razvija G. Marsaglia. Testi tam so bolj zahtevni, saj iščejo najpogostejše šibke točke generatorjev, ki so mnogokrat povzročene zaradi neizogibne uporabe linearnih komponent. Ti testi so bili uporabljeni tudi pri utemeljevanju kvalitete kandidatov za tokovne šifre pri evropskem projektu NESSIE. Da testi ne pokažejo nobenih napak na PRBG, je potreben, ne pa tudi zadosten pogoj, da je PRBG varen za uporabo v kriptografske namene.

Pravimo, da PRBG **ustreza vsem polinomskim statističnim testom**, če noben algoritem polinomske časovne zahtevnosti ne more pravilno določiti razlike med zaporedjem generiranim s tem PRBG in zaporedjem generiranim z RBG z verjetnostjo večjo od

1/2. Po terminologiji, ki jo bomo uvedli, to pomeni, da ne obstaja ε-razlikovalec med porazdelitvijo zaporedij, generiranih s PRBG, in poreazdelitvijo zaporedij, generiranih s RBG.

Pravimo, da PRBG **ustreza napovednemu testu**, če noben algoritem polinomske časovne zahtevnosti ne more za noben i na podlagi danih prvih $i-1$ bitov izhodnega zaporedja $\boldsymbol{s}$ določiti naslednji bit, to je bit s_{i+1}, zaporedja $\boldsymbol{s}$ z verjetnostjo večjo od 1/2.

Naslednji izrek nam bo povedal, da sta definiciji ekvivalentni. Definiciji nam karakterizirata računsko varnost z enačbo (3.1) opisanega tokovnega tajnopisa, zato PRBG, ki ustreza eni od definicij, pravimo **kriptografsko varen PRBG**. Sinhroni tokovni algoritmi, glej sliko 3.3, uporabljajo ključ za tvorbo začetnega notranjega stanja, skupaj z notranjo preslikavo f pa se tvorijo nova notranja stanja. Pri teh algoritmih je pomembno, da je niz zaporednih notranjih stanj, najpomembneje pa, da je s tem in s pomočjo izhodne preslikave g izhodni niz čimbolj naključen. Zato je vsak tokovni šifrirni algoritmi obenem tudi PRBG. Če naključnost izhodnega niza ni dobra, lahko s statističnimi metodami ugotovimo lastnosti vhodnega niza, kar je prvi korak k razbitju šifrirnega algoritma. Ideje za razbitje najdemo v dokazih trditev 40 in 41, katerih posledica je naslednji izrek.

Izrek 39 (Univerzalnost napovednega testa). *PRBG ustreza napovednemu testu, natanko tedaj ko ustreza vsem polinomskim statističnim testom.*

V nadaljevanju bomo izpeljali dve trditvi, ki dokazujeta pravilnost izreka. Za dokaz ($\Rightarrow$) uporabimo trditev 41, za ($\Leftarrow$) pa trditev 40.

Vzemimo verjetnostni algoritem $\boldsymbol{A}$ polinomske časovne zahtevnosti, ki za vsakemu zaporedju $\boldsymbol{z} \in \mathbb{Z}_2^\ell$ določi vrednost 1 ali 0. Algoritem $\boldsymbol{A}$ je pravzaprav funkcija

$$\boldsymbol{A} : \mathbb{Z}_2^\ell \to \{0,1\}\,,$$

oziroma slučajna spremenljivka na prostoru binarnih zaporedij dolžine ℓ. Za dano porazdelitev p na prostoru binarnih zaporedij dolžine ℓ lahko izračunamo pričakovano vrednost (matematično upanje) spremenljivke $\boldsymbol{A}$

$$E_{\boldsymbol{A}}(p) = \sum_{\boldsymbol{z} \in \mathbb{Z}_2^\ell,\, \boldsymbol{A}(\boldsymbol{z})=1} p(\boldsymbol{z})\,.$$

Za algoritem $\boldsymbol{A}$ pravimo, da je **ε-razlikovalec** med porazdelitvama p_0 in p_1, če je

$$|E_{\boldsymbol{A}}(p_0) - E_{\boldsymbol{A}}(p_1)| \geq \varepsilon$$

za nek $\varepsilon > 0$. Naj bo **(k,ℓ)-PRBG** oznaka za PRBG s semenom dolzine k, ki generira zaporedje dolzine ℓ. V ozadju zgornje definicije imamo željo, da bi nam algoritem določil 1 natanko takrat (ali čim bližje temu), ko je zaporedje lahko generirano s (k,ℓ)-PRBG.

Za verjetnostni algoritem $\boldsymbol{B}_i$, $0 \leq i \leq \ell - 1$, polinomske časovne zahtevnosti pa pravimo, da je **ε-napovedovalec naslednjega bita** za (k,ℓ)-PRBG, če iz prvih $i-1$ bitov izhodnega zaporedja pravilno napove naslednji bit z verjetnostjo vsaj $1/2+\varepsilon$. Vpeljimo oznako $\boldsymbol{z}^i = z_1, \ldots, z_i$ in naj bo p_1 porazdelitev, ki jo ima (k,ℓ)-PRBG na prostoru zaporedij dolžine ℓ, pri čemer je seme izbrano naključno z RBG. Če s $p_1(\boldsymbol{z}^i)$ označimo verjetnost, da nam je naš (k,ℓ)-PRBG vrnil $\boldsymbol{z}^i$ kot prvih i bitov, potem je

$$\sum_{\boldsymbol{z}^i \in \mathbb{Z}_2^i,\, \boldsymbol{B}_i(\boldsymbol{z}^{i-1})=z_i} p_1(\boldsymbol{z}^i) \geq \frac{1}{2} + \varepsilon\,.$$

Vsota v enačbi gre pravzaprav čez vsa zaporedja $\boldsymbol{z}^i$, ki jih lahko zgenerira naš PRBG, saj je za ostala zaporedja $p_1(\boldsymbol{z}^i) = 0$.

Konstrukcija ε-razlikovalca $\boldsymbol{A}$ iz ε-napovedovalca $\boldsymbol{B}_i$ naslednjega bita za (k,ℓ)-PRBG
input: $\boldsymbol{z} = (z_1, \ldots, z_\ell)$
1. **if** $\boldsymbol{B}_i(\boldsymbol{z}^{i-1}) = z_i$ **then**
$\quad\boldsymbol{A}(\boldsymbol{z}) = 1$
else
$\quad\boldsymbol{A}(\boldsymbol{z}) = 0$

Slika 3.5: Algoritem za konstrukcijo ε-razlikovalca iz ε-napovedovalca naslednjega bita

Trditev 40. [87, Theorem 12.2] *Naj bo $\boldsymbol{B}_i$ ε-napovedovalec naslednjega bita za (k,ℓ)-PRBG f in p_1 porazdelitev, ki jo ima f na prostoru $\mathbb{Z}_2^\ell$. Naj bo p_0 enakomerna porazdelitev na prostoru $\mathbb{Z}_2^\ell$. Potem je algoritem $\boldsymbol{A}$, opisan na sliki 3.5, ε-razlikovalec med porazdelitvama p_1 in p_0.*

Dokaz. Ker je $\boldsymbol{A}(\boldsymbol{z}) = 1$ natanko tedaj, ko je $\boldsymbol{B}_i(\boldsymbol{z}^{i-1}) = z_i$, lahko izračunamo

$$E_{\boldsymbol{A}}(p_1) = \sum_{\boldsymbol{z}\in\mathbb{Z}_2^\ell,\, \boldsymbol{A}(\boldsymbol{z})=1} p_1(\boldsymbol{z}) = \sum_{\boldsymbol{z}\in\mathbb{Z}_2^\ell,\, \boldsymbol{B}_i(\boldsymbol{z}^{i-1})=z_i} p_1(\boldsymbol{z}) = \sum_{\boldsymbol{z}^i\in\mathbb{Z}_2^i,\, \boldsymbol{B}_i(\boldsymbol{z}^{i-1})=z_i} p_1(\boldsymbol{z}^i) \geq \frac{1}{2} + \varepsilon\,.$$

Očitno je $E_{\boldsymbol{A}}(p_0) = 1/2$ in zato $|E_{\boldsymbol{A}}(p_0) - E_{\boldsymbol{A}}(p_1)| \geq \varepsilon$, kar pomeni, da je algoritem $\boldsymbol{A}$ ε-razlikovalec med porazdelitvama p_1 in p_0. ■

Konstrukcija (ε/ℓ)-napovedovalca $\boldsymbol{B}_i$ naslednjega bita za (k,ℓ)-PRBG iz ε-razlikovalca $\boldsymbol{A}$
predpostavke: i ustrezen, to je $|E_{\boldsymbol{A}}(q_{i-1}) - E_{\boldsymbol{A}}(q_i)| \geq \varepsilon/\ell$, glej dokaz trditve 41
input: $\boldsymbol{z}^{i-1} = (z_1, \ldots, z_{i-1})$
1. uporabi RBG in določi $(z_i, \ldots, z_\ell) \in \mathbb{Z}_2^{\ell-i+1}$
2. $\boldsymbol{z} = (z_1, \ldots, z_{i-1}, z_i, \ldots, z_\ell)$
3. $\boldsymbol{B}_i(\boldsymbol{z}^{i-1}) = (\boldsymbol{A}(\boldsymbol{z}) + z_i) \bmod 2$

Slika 3.6: Algoritem za konstrukcijo (ε/ℓ)-napovedovalca naslednjega bita iz ε-razlikovalca

Trditev 41. [87, Theorem 12.3] *Naj bo algoritem $\boldsymbol{A}$ ε-razlikovalec med porazdelitvama p_1 in p_0, kjer je p_1 porazdelitev, ki jo ima (k,ℓ)-PRBG f na prostoru $\mathbb{Z}_2^\ell$ in p_0 enakomerna porazdelitev na prostoru $\mathbb{Z}_2^\ell$. Potem obstaja i, pri katerem obstaja (ε/ℓ)-napovedovalec $\boldsymbol{B}_i$ naslednjega bita za f.*

Dokaz. Naj bo q_i, $0 \leq i \leq \ell$, oznaka za porazdelitev na prostoru $\mathbb{Z}_2^\ell$, kjer je prvih i bitov generiranih z f, ostalih $\ell - i$ pa z RBG. Torej je $q_0 = p_0$, $q_\ell = p_1$ in po predpostavki velja

$$|E_{\boldsymbol{A}}(q_0) - E_{\boldsymbol{A}}(q_\ell)| \geq \varepsilon\,.$$

Z uporabo trikotniške neenakosti na $E_{\boldsymbol{A}}(q_0) - E_{\boldsymbol{A}}(q_\ell) = \sum_{i=1}^{\ell}[E_{\boldsymbol{A}}(q_{i-1}) - E_{\boldsymbol{A}}(q_i)]$ dobimo $|E_{\boldsymbol{A}}(q_0) - E_{\boldsymbol{A}}(q_\ell)| \leq \sum_{i=1}^{\ell} |E_{\boldsymbol{A}}(q_{i-1}) - E_{\boldsymbol{A}}(q_i)|$, torej je

$$\sum_{i=1}^{\ell} |E_{\boldsymbol{A}}(q_{i-1}) - E_{\boldsymbol{A}}(q_i)| \geq \varepsilon\,.$$

Potem obstaja i, pri katerem je i-ti sumand zgornje vsote večji od ε/ℓ. Brez škode za splošnost lahko privzamemo

$$E_{\boldsymbol{A}}(q_{i-1}) - E_{\boldsymbol{A}}(q_i) \geq \frac{\varepsilon}{\ell}\,.$$

Naj bo algoritem $\boldsymbol{B}_i$ tak kot je opisan na sliki 3.6. Če bi bilo $E_{\boldsymbol{A}}(q_{i-1}) - E_{\boldsymbol{A}}(q_i) \leq \varepsilon/\ell$, potem bi v algoritmu vzeli $\boldsymbol{B}_i(\boldsymbol{z}^{i-1}) = (\boldsymbol{A}(\boldsymbol{z}) + z_i + 1) \bmod 2$. Naj bo vektor $\boldsymbol{z}$ tisti, ki ga dobimo v algoritmu, kar pomeni, da je bit z_i dobljen z RBG. Izračunajmo verjetnost pravilne napovedi algoritma $\boldsymbol{B}_i$ pri danih podatkih $\boldsymbol{z}^{i-1} = (z_1, \ldots, z_{i-1})$:

$$\begin{aligned}
&P(\boldsymbol{B}_i \text{ pravilno napove naslednji bit pri danih podatkih } \boldsymbol{z}^{i-1}) = \\
&= P(f \text{ nadaljuje zaporedje } \boldsymbol{z}^{i-1} \text{ z bitom } \boldsymbol{B}_i(\boldsymbol{z}^{i-1})) = \\
&= P(f \text{ nadaljuje zaporedje } \boldsymbol{z}^{i-1} \text{ z bitom } \boldsymbol{B}_i(\boldsymbol{z}^{i-1}) | \boldsymbol{A}(\boldsymbol{z}) = 0)\; P(\boldsymbol{A}(\boldsymbol{z}) = 0) + \\
&\quad + P(f \text{ nadaljuje zaporedje } \boldsymbol{z}^{i-1} \text{ z bitom } \boldsymbol{B}_i(\boldsymbol{z}^{i-1}) | \boldsymbol{A}(\boldsymbol{z}) = 1)\; P(\boldsymbol{A}(\boldsymbol{z}) = 1)
\end{aligned}$$

Z upoštevanjem formule $\boldsymbol{B}_i(\boldsymbol{z}^{i-1}) = (\boldsymbol{A}(\boldsymbol{z}) + z_i) \bmod 2$ iz algoritma nadaljujemo

$$\begin{aligned}
&P(\boldsymbol{B}_i \text{ pravilno napove naslednji bit pri danih podatkih } \boldsymbol{z}^{i-1}) = \\
&= P(f \text{ nadaljuje zaporedje } \boldsymbol{z}^{i-1} \text{ z bitom } z_i)\; P(\boldsymbol{A}(\boldsymbol{z}) = 0) + \\
&\quad + P(f \text{ nadaljuje zaporedje } \boldsymbol{z}^{i-1} \text{ z bitom } 1 - z_i)\; P(\boldsymbol{A}(\boldsymbol{z}) = 1) = \\
&= p_1(z_i | \boldsymbol{z}^{i-1})\; P(\boldsymbol{A}(\boldsymbol{z}) = 0) + p_1(1 - z_i | \boldsymbol{z}^{i-1})\; P(\boldsymbol{A}(\boldsymbol{z}) = 1) = \\
&= p_1(z_i | \boldsymbol{z}^{i-1})\; P(\boldsymbol{A}(\boldsymbol{z}) = 0) + (1 - p_1(z_i | \boldsymbol{z}^{i-1}))\; P(\boldsymbol{A}(\boldsymbol{z}) = 1)\,.
\end{aligned}$$

Ker ima zaporedje $\boldsymbol{z}$ porazdelitev q_{i-1}, je $P(\boldsymbol{A}(\boldsymbol{z}) = 0) = \sum_{\boldsymbol{z}\in\mathbb{Z}_2^\ell,\, \boldsymbol{A}(\boldsymbol{z})=0} q_{i-1}(\boldsymbol{z})$, analogno velja tudi za $P(\boldsymbol{A}(\boldsymbol{z}) = 1)$. Izpeljimo še

$$\begin{aligned}
&q_{i-1}(\boldsymbol{z}) p_1(z_i | \boldsymbol{z}^{i-1}) = \\
&= P(\text{PRBG vrne } \boldsymbol{z}^{i-1} \text{ in RBG ostalih } \ell - i + 1 \text{ bitov}) p_1(z_i | \boldsymbol{z}^{i-1}) = \\
&= p_1(\boldsymbol{z}^{i-1})(1/2)^{\ell-i+1} p_1(z_i | \boldsymbol{z}^{i-1}) = \\
&= p_1(\boldsymbol{z}^{i})(1/2)^{\ell-i+1} = \\
&= P(\text{PRBG vrne } \boldsymbol{z}^{i} \text{ in RBG ostalih } \ell - i \text{ bitov})(1/2) = \frac{q_i(\boldsymbol{z})}{2}
\end{aligned}$$

in dokončno izračunajmo iskano verjetnost

$$\begin{aligned}
&P(\boldsymbol{B}_i \text{ pravilno napove naslednji bit pri danih podatkih } \boldsymbol{z}^{i-1}) = \\
&= \sum_{\boldsymbol{z}\in\mathbb{Z}_2^\ell,\, \boldsymbol{A}(\boldsymbol{z})=0} q_{i-1}(\boldsymbol{z}) p_1(z_i | \boldsymbol{z}^{i-1}) + \sum_{\boldsymbol{z}\in\mathbb{Z}_2^\ell,\, \boldsymbol{A}(\boldsymbol{z})=1} q_{i-1}(\boldsymbol{z})(1 - p_1(z_i | \boldsymbol{z}^{i-1})) = \\
&= \sum_{\boldsymbol{z}\in\mathbb{Z}_2^\ell,\, \boldsymbol{A}(\boldsymbol{z})=0} \frac{q_i(\boldsymbol{z})}{2} + \sum_{\boldsymbol{z}\in\mathbb{Z}_2^\ell,\, \boldsymbol{A}(\boldsymbol{z})=1} q_{i-1}(\boldsymbol{z}) - \sum_{\boldsymbol{z}\in\mathbb{Z}_2^\ell,\, \boldsymbol{A}(\boldsymbol{z})=1} \frac{q_i(\boldsymbol{z})}{2} = \\
&= \frac{1}{2}(1 - E_{\boldsymbol{A}}(q_i)) + E_{\boldsymbol{A}}(q_{i-1}) - \frac{1}{2} E_{\boldsymbol{A}}(q_i) = \\
&= \frac{1}{2} + E_{\boldsymbol{A}}(q_{i-1}) - E_{\boldsymbol{A}}(q_i) \geq \frac{1}{2} + \frac{\varepsilon}{\ell}\,,
\end{aligned}$$

kar potrjuje trditev. ■

Oglejmo si sedaj poučen primer, ki pokaže, da prelepe statistične lastnosti PRBG lahko vodijo do algoritma, ki je razlikovalec med porazdelitvijo PRBG in porazdelitvijo RBG, glede na zadnjo trditev pa lahko tudi do napovedovalca naslednjega bita PRBG, kar je daleč od kriptografske varnosti PRBG.

Primer. Predpostavimo, da nam PRBG f vrača zaporedja dolžine ℓ, pri katerih je natanko $\ell/2$ bitov enakih 0. Potem lahko sestavimo determinističen algoritem $\boldsymbol{A}$ polinomske časovne zahtevnosti

$$\boldsymbol{A}(\boldsymbol{z}) = \begin{cases} 1 & ; \quad \boldsymbol{z} = (z_1, \ldots, z_\ell) \text{ ima } \ell/2 \text{ bitov enakih } 0 \\ 0 & ; \quad \text{sicer} \end{cases} .$$

Ker je

$$E_{\boldsymbol{A}}(p_0) = \frac{\binom{\ell}{\ell/2}}{2^\ell} \qquad \text{in} \qquad E_{\boldsymbol{A}}(p_1) = 1 \,,$$

obstaja $\varepsilon > 0$, tako da je

$$|E_{\boldsymbol{A}}(p_0) - E_{\boldsymbol{A}}(p_1)| \geq \varepsilon \,,$$

torej je $\boldsymbol{A}$ ε-razlikovalec med porazdelitvama p_0 in p_1. Ker je $\lim_{\ell\to\infty} E_{\boldsymbol{A}}(p_0) = 0$, lahko pri večjem ℓ izberemo večji ε, kar pomeni, da večji kot je ℓ, boljši razlikovalec je $\boldsymbol{A}$. ▲

Naj n_0 označuje število ničel, n_1 pa število enic v zaporedju. Primer bi deloval pri vsaki porazdelitvi slučajne spremenljivke $X = (n_0 - n_1)^2/\ell$, drugačni od $\chi^2(1)$, le izračun, oziroma ocena ε bi bila težje določljiva. Kakšno porazdelitev ima spremenljivka X in tudi druge, ki jih uporabljamo za preverjanje naključnosti zaporedij, si lahko preberemo v Novak [71, poglavje 4]. Poglejmo si še primer, kako lahko test parov, glej Novak [71, str. 49], vodi do napovedovalca naslednjega bita.

Primer. Naj bo f (5,10)-PRBG, skonstruiran s pomočjo linearnega kongruenčnega generatorja, pri katerem je $g(s) = (3s + 5) \bmod 31$. Generator g ima dva cikla, to je $(0, 5, 20, 3, 14, 16, 22, 9, 1, 8, 29, 30, 2, 11, 7, 26, 21, 6, 23, 12, 10, 4, 17, 25, 18, 28, 27, 24, 15, 19)$ in (13). Seme generatorja f je dolgo 5 bitov in ga pretvorimo v število $s_0 \in \mathbb{Z}_{32}$. Nato tvorimo zaporedji

$$s_i = g(s_{i-1}) \qquad \text{in} \qquad z_i = s_i \bmod 2 \,, \qquad 1 \leq i \leq 10 \,.$$

Zaporedje $\boldsymbol{z}^{10} = (z_1, \ldots, z_{10})$ je izhodno zaporedje PRBG f. Naj opomnimo, da seme ni del zaporedja. Izpišimo izhodna zaporedja PRBG f pri danih semenih:

s_0	$\boldsymbol{z}^{10}$
0	1010001101
1	0100110101
2	1101010001
3	0001101001
4	1100101101
5	0100011010
6	1000110010
7	0101000110
8	1001101010
9	1010011010

s_0	$\boldsymbol{z}^{10}$
10	0110010110
11	1010100011
12	0011001011
13	1111111111
14	0011010011
15	1010100011
16	0110100110
17	1001011010
18	0101101010
19	0101000110

s_0	$\boldsymbol{z}^{10}$
20	1000110100
21	0100011001
22	1101001101
23	0001100101
24	1101010001
25	0010110101
26	1010001100
27	0110101000
28	1011010100
29	0011010100
30	0110101000

.

Če pogledamo porazdelitev parov 00, 01, 10 in 11 čez vsa zaporedja, vidimo, da je $n_{00} = 54$, $n_{01} = 90$, $n_{10} = 90$, $n_{11} = 45$ (vseh parov v zaporedjih je $9 \cdot 31 = 279$). Torej lahko brez težav skonstruiramo napovedovalca naslednjega bita $\boldsymbol{B}_i(z_1, \ldots, z_{i-1}) = 1 - z_{i-1}$, saj vidimo, da enki v večini primerov sledi 0, ničli pa 1. Če upoštevamo še, da je na prvih 9 mestih v vseh zaporedjih 135 enk in 144 ničel, lahko izračunamo verjetnost, da $\boldsymbol{B}_i$ pravilno napove nasledji bit

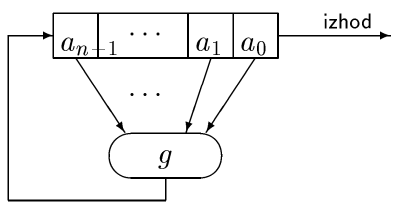
$$
\begin{aligned}
P(\text{pravilno }) = \\
&= P(\text{pravilno } | z_{i-1} = 0)P(z_{i-1} = 0) + P(\text{pravilno } | z_{i-1} = 1)P(z_{i-1} = 1) = \\
&= \frac{90}{144} \cdot \frac{144}{279} + \frac{90}{135} \cdot \frac{135}{279} = \frac{20}{31} \, .
\end{aligned}
$$

Torej je $\varepsilon = \frac{20}{31} - \frac{1}{2} = \frac{9}{62}$ in $\boldsymbol{B}_i$ je $\frac{9}{62}$-napovedovalec naslednjega bita za PRBG f. ▲

3.5 Povratni pomični registri – FSR

Povratni pomični registri, ponavadi linearni povratni pomični registri (LFSR), predstavljajo osnovne komponente mnogih generatorjev toka ključev, ki smo jih shematični predstavili na sliki 3.3. Seveda ni nujno, da jih uporabimo, saj se v praksi za bolj varne izkažejo tokovne šifre, ki so dobljene z modifikacijo bločnih. Kandidat BMGL, finalist za NESSIE tokovno šifro, uporablja Rijndael bločno šifro (zmagovalec ameriškega AES projekta) za enosmerno funkcijo v svoji zgradbi. Vendar pa take šifre v praksi ponavadi niso tako hitre, kot tiste, katerih zgradba sloni na uporabi LFSR generatorjev.

Povratni pomični register (**FSR** - feedback shift register) je sestavljen iz n spominskih celic, katerih vsebina se ob vsakem taktu pomakne za eno mesto naprej, vsebina zadnje celice je izhodni element takta, vsebina prve celice pa postane $f(a_0, \ldots, a_{n-1})$. Celotna shema je predstavljena na naslednji sliki



Slika 3.7: Shema povratnega pomičnega registra (FSR)

Izhodno zaporedje FSR je tako podano z n začetnimi členi $a_0, \ldots, a_{n-1}$ in z rekurzivno enačbo

$$a_{k+n} = f(a_k, a_{k+1}, \ldots, a_{k+n-1}) \, , \qquad k = 0, 1, \ldots \, ,$$

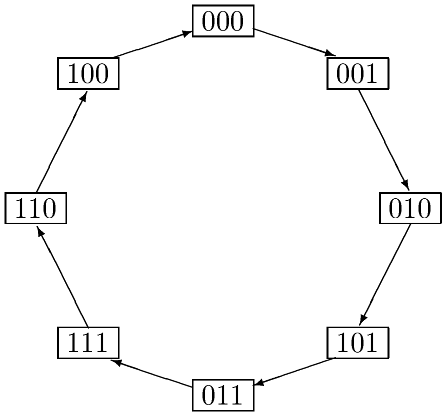
kjer je

$$f : \mathrm{GF}(q)^n \to \mathrm{GF}(q) \, .$$

Glede na izbrano funkcijo f lahko določimo **diagram stanj** za FSR f. Želeli bi si, da je diagram sestavljen iz enega samega cikla, kakor bo v naslednjem primeru. Vsekakor pa morajo imeti vsa zaporedja, dobljena s FSR, zelo dolgo periodo, sicer ne bi bilo težko napovedati naslednjega bita.

Primer. Vzemimo $n = 3$ in $f(x_0, x_1, x_2) = x_1x_2 + x_0 + x_2 + 1$ in določimo diagram stanj.

$x_0x_1x_2$	$y = f(x_0, x_1, x_2)$	x_1x_2y
000	1	001
001	0	010
010	1	101
011	1	111
100	0	000
101	1	011
110	0	100
111	0	110



▲

Izhodna zaporedja FSR f dolžine n, ki imajo periodo 2^n, imenujemo **De Bruijnova zaporedja**. Kako je perioda zaporedja povezana s funkcijo f, si lahko preberemo v Golomb [30], vendar pa je do sedaj objavljenih bolj malo rezultatov o tem. Zadnji primer je primer De Bruijnovega zaporedja. Vemo, da obstaja $|B|^{|A|}$ različnih funkcij $f : A \to B$, torej obstaja q^{q^n} različnih funkcij f za FSR dolžine n.

Poglavje 4

Linearni povratni pomični registri – LFSR

Linearni povratni pomični register (**LFSR** - Linear Feedback Shift Register) je povratni pomični register, določen s funkcijo f, ki je linearna v vsaki komponenti. V prvih treh razdelkih bomo predstavili osnove za razumevanje LFSR zaporedij ter določili njihovo periodo. Pri izpeljavi le te nam je v pomoč reprezentacija zaporedij s sledjo. Njegove prednosti so hitrost, možnost preproste softverske in hardverske implementacije ter zelo lepe statistične lastnosti R1 – R4, ki jih bomo predstavili v 4. razdelku. Ima pa veliko slabost, to je zelo majhna linearna zahtevnost izhodnega zaporedja. Če je ℓ linearna zahtevnost LFSR zaporedja, lahko kljub dolgi periodi rekonstruiramo celotno zaporedje iz poljubnih 2ℓ zaporednih bitov izhodnega zaporedja. Npr. LFSR dolžine 100 bitov nam lahko vrne m-zaporedje s periodo dolžine $2^{100} - 1$, vendar pa lahko iz poljubnega podzaporedja 200 zaporednih bitov s pomočjo Berlekamp-Masseyevega algoritma rekonstruiramo začetno stanje LFSR in s tem celotno zaporedje. Kljub temu je LFSR generator nepogrešljiv del pri tokovnih šifrah, ki zahtevajo veliko hitrost. Seveda pa moramo linearno zahtevnost povečati s pomočjo ene od metod nelinearizacije, vendar moramo paziti, da pri tem ne pride do korelacijskih anomalij izhodnih zaporedij.

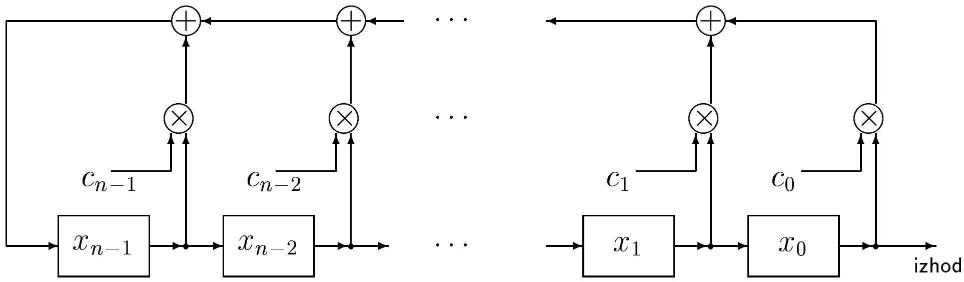
V 5. razdelku bomo opisali Berlekamp-Masseyev algoritem za sintezo LFSR generatorja, v 6. pa bomo z njegovo pomočjo izračunali pričakovano linearno zahtevnost in varianco naključnih zaporedij. V zadnjem razdelku so predstavljene najosnovnejše ideje za nelinearizacijo LFSR zaporedij, s katero ne smemo pokvariti lepih statističnih lastnosti.

4.1 LFSR in linearna zahtevnost zaporedij

Naj bo $g(x) = 1 - \sum_{i=1}^{n} c_{n-i}x_i = 1 - (c_{n-1}x + c_{n-2}x^2 + \cdots + c_0x^n)$ polinom nad končnim obsegom $\mathbb{F}$. Zaporedje $\boldsymbol{a} = (a_1, a_2, \ldots)$, ki je podano z linearno rekurzivno enačbo

$$a_{k+n} = \sum_{i=0}^{n-1} c_i a_{k+i} \qquad \text{oziroma}$$
$$a_{k+n} = c_0a_k + c_1a_{k+1} + \cdots + c_{n-1}a_{k+n-1}\,, \qquad k = 0, 1, \ldots\,, \tag{4.1}$$

imenujemo **LFSR zaporedje** nad obsegom $\mathbb{F}$ s **povratnim polinomom** g. Parameter n imenujemo **dolžina LFSR**. Slika 4.1 predstavlja delovanje **LFSR generatorja** nad



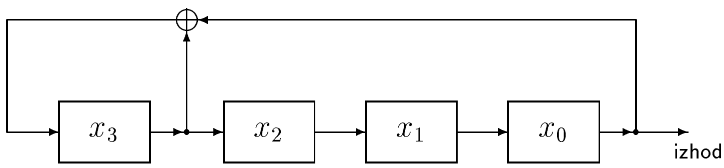
Slika 4.1: Delovanje LFSR nad $\mathbb{F} = \mathrm{GF}(q)$

poljubnim obsegom $\mathbb{F} = \mathrm{GF}(q)$. Povratni bit x_n izračunamo kot

$$x_n = \sum_{i=0}^{n-1} c_i x_i \qquad \text{oziroma} \qquad x_n = c_0x_0 + c_1x_1 + \cdots + c_{n-1}x_{n-1}\,.$$

Povratni polinom, ki nam da LFSR z željenimi lastnostmi mora biti primitiven, kot bomo videli. Običajno je trinom in tako da izračun naslednjega člena LFSR zaporedja ne terja $O(n)$, ampak le $O(1)$ operacij. Golombova hipoteza pravi, da obstaja neskončno mnogo takih n, pri katerih obstaja primitivni trinom stopnje n nad obsegom $\mathbb{Z}_2$.

Primer. Denimo, da imamo LFSR nad obsegom $\mathbb{Z}_2$ dolžine 4 s povratnim polinomom $1 + x + x^4$ (v $\mathbb{Z}_2$ je seštevanje enako odštevanju). Ponazorimo ga lahko s sliko



Če je začetno stanje registra enako $[0, 1, 0, 0]$, potem je izhodno zaporedje enako

$$s = \underbrace{0, 0, 1, 0}_{\text{seme}}, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, \underbrace{0, 0, 1, 0}_{\text{ponovitev}}, 0, 0, 1, \ldots\,.$$

Pri računanju smo si pomagali z enačbo $s_j = s_{j-1} \oplus s_{j-4}$. ▲

Za lažje dokazovanje izrekov vpeljimo nekaj novih oznak. Naj bo $\mathbb{F} = \mathrm{GF}(q)$. Z

$$V(\mathbb{F}) = \{\boldsymbol{a} = (a_0, a_1, \ldots) | a_i \in \mathbb{F}\}$$

označimo množico vseh neskončnih zaporedij nad obsegom $\mathbb{F}$. Z operacijama

$$\begin{aligned} \boldsymbol{a} + \boldsymbol{b} &= (a_0 + b_0, a_1 + b_1, \ldots) \qquad \text{in} \\ c\boldsymbol{a} &= (ca_0, ca_1, \ldots) \end{aligned}$$

postane $V(\mathbb{F})$ vektorski prostor nad obsegom $\mathbb{F}$. V pomoč nam bo tudi linearna preslikava **levi pomik** $L : V(\mathbb{F}) \to V(\mathbb{F})$

$$L\boldsymbol{a} = (a_1, a_2, \ldots)\,.$$

Sistem enačb (4.1) je ekvivalenten enačbi

$$\left(L^n - \sum_{i=0}^{n-1} c_i L^i\right) \boldsymbol{a} = \mathbf{0}\,. \tag{4.2}$$

Polinom $f(x) = x^n - \sum_{i=0}^{n-1} c_i x^i \in \mathbb{F}[x]$ imenujemo **karakteristični polinom** zaporedja $\boldsymbol{a}$ nad $\mathbb{F}$ (ni enoličen). Po dogovoru je $f(x) = 1$ karakteristični polinom ničelnega zaporedja. Označimo z $G(f)$ množico vseh LFSR zaporedij s karakterističnim polinomom f,

$$G(f) = \{\boldsymbol{a} \in V(\mathbb{F}) | f(L)\boldsymbol{a} = 0\}\,.$$

Ker je $G(f)$ jedro linearne preslikave $f(L)$, je vektorski podprostor prostora $V(\mathbb{F})$.

Poiskati želimo karakteristični polinom danega zaporedja z minimalno stopnjo, torej poiskati čimkrajši LFSR, ki generira celotno dano zaporedje. Naj bo $A(\boldsymbol{a})$ množica vseh karakterističnih polinomov zaporedja $\boldsymbol{a}$, torej je

$$f(x) \in A(\boldsymbol{a}) \iff f(L)\boldsymbol{a} = 0\,.$$

Monični polinom z najmanjšo stopnjo v $A(\boldsymbol{a})$ imenujemo **minimalni polinom** zaporedja $\boldsymbol{a}$, njegovo stopnjo, to je dolžino najkrajšega LFSR, ki generira zaporedje, pa imenujemo **linearna zahtevnost** zaporedja $\boldsymbol{a}$.

Lema 42. *(delovanje kolobarja polinomov na množici zaporedij) Naj bo* $\boldsymbol{a}$ *LFSR zaporedje nad obsegom* $\mathbb{F}$. *Potem za poljubna polinoma* $f(x)$ *in* $g(x) \in \mathbb{F}[x]$ *velja*

(a) $(f(L) + g(L))\,\boldsymbol{a} = f(L)\boldsymbol{a} + g(L)\boldsymbol{a}$

(b) $(f(L)g(L))\,\boldsymbol{a} = f(L)\,(g(L)\boldsymbol{a})$

Dokaz. Ker je k-ti člen zaporedja $f(L)\boldsymbol{a}$ enak

$$(f(L)\boldsymbol{a})_k = \sum_{i=0}^{\infty} f_i a_{k+i}\,,$$

je veljavnost točke (a) očitna. Za dokaz točke (b) pa izračunajmo

$$((f(L)g(L))\,\boldsymbol{a})_k = \sum_{i=0}^{\infty} (fg)_i a_{k+i} = \sum_{i=0}^{\infty} \sum_{j=0}^{i} f_j g_{i-j} a_{k+i}$$

Ker sta f in g polinoma, je dobljena zadnja dvojna vsota končna. Zato lahko zamenjamo vrstni red sumiranja in dobimo

$$\begin{aligned} ((f(L)g(L))\,\boldsymbol{a})_k &= \sum_{j=0}^{\infty} \sum_{i=j}^{\infty} f_j g_{i-j} a_{k+i} = \sum_{j=0}^{\infty} f_j \sum_{i=j}^{\infty} g_{i-j} a_{k+i} = \\ &= \sum_{j=0}^{\infty} f_j \sum_{i=0}^{\infty} g_{(i+j)-j} a_{k+(i+j)} = \sum_{j=0}^{\infty} f_j \sum_{i=0}^{\infty} g_i a_{k+j+i} = \\ &= \sum_{j=0}^{\infty} f_j \left(g(L)\boldsymbol{a}\right)_{k+j} = \left(f(L)\left(g(L)\boldsymbol{a}\right)\right)_k\,, \end{aligned}$$

kar smo želeli pokazati. ■

Iz zadnje leme sledi očitna posledica.

Posledica 43. *Naj bo* $\boldsymbol{a}$ *LFSR zaporedje nad obsegom* $\mathbb{F}$. *Potem velja*

(a) $0 \in A(\boldsymbol{a})$,

(b) $f(x), g(x) \in A(\boldsymbol{a}) \Longrightarrow f(x) \pm g(x) \in A(\boldsymbol{a})$ *in*

(c) $f(x) \in A(\boldsymbol{a}), h(x) \in \mathbb{F}[x] \Longrightarrow h(x)f(x) \in A(\boldsymbol{a})$. ■

Izrek 44. [31, Theorem 3.4] *Naj bo* $\boldsymbol{a} \in V(\mathbb{F})$. *Za minimalni polinom* $m(x)$ *zaporedja* $\boldsymbol{a}$ *velja:*

(a) *če obstaja, je enolično določen,*

(b) $m(L)\boldsymbol{a} = 0$ *in*

(c) *za vsak polinom* $f(x) \in \mathbb{F}[x]$ *velja:* $[f(L)\boldsymbol{a} = 0 \Longrightarrow m(x)|f(x)]$.

Dokaz. Točka (b) sledi neposredno iz definicije, za točko (c) pa uporabimo izrek o deljenju polinomov:
$$f(x) = q(x)m(x) + r(x)\,.$$
Z levo in desno stranjo zadnje enačbe delujemo na zaporedju $\boldsymbol{a}$ in dobimo
$$f(L)\boldsymbol{a} = 0 \Longleftrightarrow r(L)\boldsymbol{a} = 0 \Longleftrightarrow r(L) = 0\,,$$
saj je stopnja polinoma $m(L)$ minimalna pozitivna s to lastnostjo. Očitno pa je $r(L) = 0$ natanko tedaj, ko $m(x)|f(x)$. Enoličnost iz točke (a) pa je posledica lastnosti (b), saj če bi obstajal še minimalni polinom $m_1(x)$, potem bi $m(x)|m_1(x)$ in $m_1(x)|m(x)$ in ker sta to monična polinoma, morata biti enaka. ■

Posledica 45. *Naj bo* $f(x) \in \mathbb{F}[x]$, $f(x) \neq 0$ *in* $\boldsymbol{a} \in G(f)$. *Potem minimalni polinom* $m(x)$ *za zaporedje* $\boldsymbol{a}$ *obstaja in velja* $m(x)|f(x)$. *In še, če je polinom* $f(x)$ *nerazcepen, potem je* $m(x) = f(x)$.

Minimalen polinom zaporedja v splošnem ni nerazcepen. Za karakteristični polinom zaporedja $\boldsymbol{a}$ lahko vzamemo $f(x) = (1 + x + x^2)^2$, prvi štirje členi zaporedja pa naj bodo 0,0,0,1. Ker ima zaporedje šele na četrtem mestu enko, je linearna zahtevnost zaporedja $\boldsymbol{a}$ vsaj 4, torej je $f(x)$ tudi minimalen polinom za $\boldsymbol{a}$.

4.2 Perioda LFSR zaporedij

Vzemimo polinom $f(x) \in \mathbb{F}[x]$, kjer je $\mathbb{F} = \mathrm{GF}(q)$, in naj bo $f(x) = x^k g(x)$, kjer je $g(x) \in \mathbb{F}[x]$ in $g(0) \neq 0$. **Periodo polinoma** $f(x)$ definiramo kot
$$\mathrm{per} f(x) = \min\{r > 0 \,|\, g(x) \text{ deli } x^r - 1\}\,.$$
Ker ima $f(x)$ vse ničle v neki razširitvi $\overline{\mathbb{F}} = \mathrm{GF}(q^n)$ obsega $\mathbb{F}$ in ker za vsak $\beta \in \overline{\mathbb{F}}$ velja $\beta^{q^n} = \beta$ ter zato
$$x^{q^n} - x = \prod_{\beta \in \overline{\mathbb{F}}} (x - \beta)\,,$$

je perioda dobro definirana.

Za zaporedje $\boldsymbol{a} = a_0, a_1, \ldots$ pa pravimo, da je **periodično**, če obstaja $r > 0$, da je $a_{k+r} = a_k$ za vsak $k \geq 0$. Najmanjši tak r imenujemo **perioda zaporedja**. Povežimo periodo zaporedja s periodo njegovega minimalnega polinoma.

Trditev 46. [31, Theorems 3.6, 3.7] *Naj bo $\boldsymbol{a}$ LFSR zaporedje nad obsegom $\mathbb{F} = \mathrm{GF}(q)$ z minimalnim polinomom $m(x)$. Če je $m(0) \neq 0$, potem je zaporedje $\boldsymbol{a}$ periodično zaporedje in velja*

$$\mathrm{per}(\boldsymbol{a}) = \mathrm{per}(m(x))\,.$$

Če je polinom $m(x)$ nerazcepen in stopnje n, lahko njegovo periodo izračunamo kot

$$\mathrm{per}(m(x)) = \mathrm{red}(\alpha)\,,$$

kjer je α poljubna ničla polinoma $m(x)$.

Dokaz. Za vsak polinom smo pokazali, da je perioda dobro definirana. Naj bo r perioda polinoma $m(x)$. Ker je $m(0) \neq 0$, potem velja $m(x)|(x^r - 1)$, kar pomeni, da je $x^r - 1$ karakteristični polinom zaporedja $\boldsymbol{a}$ in zaporedje je potemtakem periodično.

Pokažimo najprej, da je $\mathrm{per}(\boldsymbol{a}) \leq r$, nato pa še obratno. Za število r po definiciji velja $m(x) \mid x^r - 1$, torej je $x^r - 1 = g(x)m(x)$ za nek polinom $g(x) \in \mathbb{F}[x]$ in zato po lemi 42 dobimo

$$(L^r - I)\boldsymbol{a} = g(L)m(L)\boldsymbol{a} = g(L)\mathbf{0} = \mathbf{0}\,,$$

kar pomeni, da je $x^r - 1$ karakteristični polinom zaporedja $\boldsymbol{a}$ in po izreku 44 velja $m(x)|(x^k - 1)$, kar pomeni $\mathrm{per}(\boldsymbol{a}) \leq r$. Analogno dobimo še obratno oceno. Če je $\mathrm{per}(\boldsymbol{a}) = k$, potem je $(L^k - I)\boldsymbol{a} = 0$ in po izreku 44 velja $m(x)|(x^k - 1)$, torej je $\mathrm{per}(m(x)) \leq k$, kar pomeni $r \leq \mathrm{per}(\boldsymbol{a})$.

Ker je polinom $m(x)$ nerazcepen, nam določa obseg $\mathrm{GF}(q^n)$, razširitev obsega $\mathbb{F}$. Naj bo α ničla polinoma $m(x)$. Potem je $m(x)$ minimalni polinom elementa α. Red elementa α je po definiciji najmanjši k, za katerega velja $\alpha^k = 1$. Ker so ničle minimalnega polinoma med sabo konjugirani elementi, imajo isti red k, ki je najmanjši tak, da $m(x)|(x^k - 1)$ in zato je k tudi perioda polinoma $m(x)$. ■

Posledica 47. *Če je minimalni polinom $m(x)$ LFSR zaporedja $\boldsymbol{a}$ nad obsegom $\mathbb{F} = \mathrm{GF}(q)$ primitiven polinom stopnje n, potem je $\mathrm{per}(\boldsymbol{a}) = q^n - 1$.*

Če je dolžina LFSR enaka n, potem je maksimalna možna perioda izhodnega zaporedja enaka $q^n - 1$. Takšna zaporedja imenujemo **m-zaporedja**. V priročniku [64, Alg. 4.78] je opisan algoritem za naključno generiranje primitivnih polinomov nad $\mathbb{Z}_2$, vendar pa je za konstrukcijo LFSR, ki generira zaporedja z največjo možno periodo, dovolj pogledati v tabelo primitivnih polinomov [64, Table 4.8] za stopnje od 1 do 229.

4.3 Predstavitev LFSR zaporedij s sledjo

Vemo, da je obseg $\mathbb{F} = \mathrm{GF}(q^n)$ razširitev obsega $K = \mathrm{GF}(q)$ in da za vsak element $x \in \mathbb{F}$ velja $x^q \in K$. Skladno s tem definiramo **sled elementa** x nad obsegom K kot

$$\mathrm{Tr}_{\mathbb{F}/K}(x) = x + x^q + x^{q^2} + \cdots + x^{q^{n-1}}\,.$$

Če je $q = p$ praštevilo, označimo sled kot $\mathrm{Tr}_{\mathbb{F}}(x)$ ali Tr_p^n ali pa kar $\mathrm{Tr}(x)$. Za funkcijo sled vemo, da je linearna in surjektivna kot preslikava iz vektorskega prostora $\mathbb{F}$ nad obsegom K na prostor K. Vemo pa tudi, da velja $\mathrm{Tr}_{\mathbb{F}/K}(x) = n\,x$ za vsak $x \in K$ in $\mathrm{Tr}_{\mathbb{F}/K}(x^q) = \mathrm{Tr}_{\mathbb{F}/K}(x)$ za vsak $x \in \mathbb{F}$. Vse lastnosti lahko izpeljemo s pomočjo dejstva, da je $(x+y)^q = x^q + y^q$ za vsak $x, y \in \mathbb{F}$. Podobno lahko izpeljemo tudi tranzitivnost sledi – če so K, $\mathbb{F}$ in E obsegi, za katere velja $K \leq \mathbb{F} \leq E$, potem za vsak element $x \in E$ velja

$$\mathrm{Tr}_{E/K}(x) = \mathrm{Tr}_{\mathbb{F}/K}\big(\mathrm{Tr}_{E/\mathbb{F}}(x)\big)\,.$$

Pokažimo, kako lahko LFSR zaporedje predstavimo s pomočjo sledi.

Trditev 48. [31, Theorem 3.10] *Naj bo* $\mathbb{F} = \mathrm{GF}(q^n)$ *razširitev obsega* $K = \mathrm{GF}(q)$, *določena z nerazcepnim polinomom* $f(x) = x^n + c_{n-1}x^{n-1} + \cdots + c_1 x + c_0 \in K[x]$ *in z njegovo ničlo* α. *Naj bo* $\mathrm{Tr} = \mathrm{Tr}_{\mathbb{F}/K}$ *in* $\boldsymbol{a} = \{a_i\}_{i\geq 0}$ *zaporedje v* $\mathbb{F}$. *Potem obstaja* $\beta \in \mathbb{F}$, *pri katerem je*

$$a_i = \mathrm{Tr}(\beta\alpha^i)\,, \qquad i \geq 0\,,$$

natanko tedaj, ko je $f(x)$ *karakteristični polinom zaporedja* $\boldsymbol{a}$.

Dokaz. ($\Rightarrow$) Za vsak $k \geq 0$ z upoštevanjem lastnosti sledi in dejstva $f(\alpha) = 0$ izpeljemo

$$a_{k+n} + \sum_{i=0}^{n-1} c_i a_{k+i} = \mathrm{Tr}(\beta\alpha^{k+n}) + \sum_{i=0}^{n-1} c_i \mathrm{Tr}(\beta\alpha^{k+i}) = \mathrm{Tr}\Bigg(\beta\alpha^k\Big(\alpha^n + \sum_{i=0}^{n-1} c_i\alpha^i\Big)\Bigg) = 0\,.$$

Torej je $f(L)\boldsymbol{a} = \boldsymbol{0}$, kar pomeni, da je $f(x)$ res karakteristični polinom zaporedja $\boldsymbol{a}$.

($\Leftarrow$) Naj bo sedaj $f(x)$ karakteristični polinom zaporedja $\boldsymbol{a}$. Ker je LFSR zaporedje s karakterističnim polinomom $f(x)$ natanko določeno s prvimi n členi (q^n možnosti) in ker obstaja največ q^n različnih zaporedij $\boldsymbol{a}_\gamma = \{\gamma\alpha^i\}_{i\geq 0}$, $\gamma \in \mathbb{F}$, je za obstoj elementa β, pri katerem je $\boldsymbol{a} = \{\beta\alpha^i\}_{i\geq 0}$, dovolj pokazati, da je $\boldsymbol{a}_{\gamma_1} \neq \boldsymbol{a}_{\gamma_2}$ za vsak $\gamma_1, \gamma_2 \in \mathbb{F}$, $\gamma_1 \neq \gamma_2$, saj smo v prvem delu dokaza te trditve pokazali, da ima vsako zaporedje $\boldsymbol{a}_\gamma$, $\gamma \in \mathbb{F}$, karakteristični polinom f. Ker je Tr linearna preslikava, je linearna tudi preslikava $\gamma \mapsto [\mathrm{Tr}(\gamma\alpha^0), \mathrm{Tr}(\gamma\alpha^1), \ldots, \mathrm{Tr}(\gamma\alpha^{n-1})]^T$, ki nam določi začetnih n elementov zaporedja $\boldsymbol{a}_\gamma$. Pokazati moramo torej le, da je jedro te preslikave trivialno, oziroma da iz $\boldsymbol{a}_\gamma = \boldsymbol{0}$ sledi $\gamma = 0$. Če je $\boldsymbol{a}_\gamma = \boldsymbol{0}$, potem z upoštevanjem definicije sledi in definicije $\boldsymbol{a}_\gamma$ za začetnih n členov dobimo linearni sistem

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha & \alpha^q & \alpha^{q^2} & \cdots & \alpha^{q^{n-1}} \\ \alpha^2 & \alpha^{2q} & \alpha^{2q^2} & \cdots & \alpha^{2q^{n-1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha^{n-1} & \alpha^{(n-1)q} & \alpha^{(n-1)q^2} & \cdots & \alpha^{(n-1)q^{n-1}} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \gamma \\ \gamma^q \\ \gamma^{q^2} \\ \vdots \\ \gamma^{q^{n-1}} \end{bmatrix}}_{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}\,.$$

Ker je $\det A$ Vandermondova determinanta, potem po formuli (2.1) s strani 40 velja $\det A = \prod_{0\leq j<i\leq n-1}(\alpha^{q^i} - \alpha^{q^j}) \neq 0$, saj so vsi faktorji v produktu neničelni, ker so elementi $1, \alpha, \ldots, \alpha^{n-1}$ paroma različni, saj vemo, da tvorijo bazo vektorskega prostora $\mathbb{F}$ nad obsegom K. To pomeni, da je matrika A obrnljiva in da je $b = A^{-1}\boldsymbol{0} = \boldsymbol{0}$. Torej je $\gamma = 0$, s čimer smo zaključili dokaz. ∎

4.4 Golombovi postulati naključnosti

Predstavimo sedaj tri **Golombove [30] postulate naključnosti** za periodična zaporedja. Kasneje bomo s pomočjo predstavitev LFSR zaporedij s sledjo pokazali, da m-zaporedja imajo te lepe statistične lastnosti.

Naj bo $\boldsymbol{a}$ zaporedje s periodo $q^n - 1$ nad $\mathbb{F} = \mathrm{GF}(p^r)$. Če so $\eta, \lambda, \xi \in \mathbb{F}$ ter sta η in ξ različna od λ, potem zaporednim elementom $\eta, \underbrace{\lambda \dots \lambda}_{k}, \xi$ v zaporedju $\boldsymbol{a}$ pravimo **stik** dolžine k iz elementov λ. **Avtokorelacijska funkcija** zaporedja $\boldsymbol{a}$ je definirana kot

$$C_{\boldsymbol{a}}(\tau) = \sum_{i=0}^{q^n-2} \chi(a_i)\overline{\chi(a_{i+\tau})}\,,$$

kjer je **kanonična aditivna karakteristika** elementa x definirana kot

$$\chi(x) = e^{2\pi i\, \mathrm{Tr}_p^r(x)/p}\,.$$

Očitno je $\chi(x) \in \mathbb{C}$, saj $\mathrm{Tr}_p^r(x) \in \mathbb{Z}_p$ za vsak $x \in \mathrm{GF}(p^r)$. Za avtokorelacijsko funkcijo bi si želeli, da je enaka 0 pri vsakem τ, ki ni večkratnik periode. Seveda je to prehuda zahteva in smo zadovoljni, če je norma $C_{\boldsymbol{a}}(\tau)$ majhna. To se zgodi v primeru, ko imajo istoležni elementi periode zaporedij $\boldsymbol{a}$ in $L^\tau \boldsymbol{a}$ čimbolj enakomerno porazdelitev sledi. V primeru binarnega zaporedja nam

$$C_{\boldsymbol{a}}(\tau) = \sum_{i=0}^{q^n-2} (-1)^{a_i + a_{i+\tau}}$$

predstavlja razliko med številom med seboj enakih in številom med seboj različnih istoležnih elementov periode zaporedij $\boldsymbol{a}$ in $L^\tau \boldsymbol{a}$.

R-1 V vsaki periodi zaporedja $\boldsymbol{a}$ se vsak neničelen element pojavi q^{n-1}-krat, element 0 pa $(q^{n-1} - 1)$-krat. Tej lastnosti pravimo **uravnoteženje zaporedja**.

R-2 Vsaka perioda zaporedja $\boldsymbol{a}$ vsebuje
1. $(q-1)^2 q^{n-k-2}$ cikličnih stikov dolžine k poljubnega elementa $\lambda \in \mathbb{F}$, $1 \leq k \leq n-2$,
2. $q - 2$ cikličnih stikov dolžine $n - 1$ poljubnega neničelnega elementa $\lambda \in \mathbb{F}$,
3. $q - 1$ cikličnih stikov dolžine $n - 1$ elementa 0 in
4. en ciklični stik dolžine n poljubnega neničelnega elementa $\lambda \in \mathbb{F}$.

R-3 Avtokorelacijska funkcija $C_{\boldsymbol{a}}(\tau)$ zavzame le dve vrednosti in sicer

$$C_{\boldsymbol{a}}(\tau) = \begin{cases} q^n - 1 & ;\ \tau \equiv 0 \pmod{q^n - 1} \\ -1 & ;\ \text{sicer} \end{cases}.$$

Lastnost **R-1** lahko posplošimo na **R-4**.

R-4 V vsaki periodi zaporedja $\boldsymbol{a}$ se vsaka neničelna n-terica $\lambda_1, \lambda_2, \dots, \lambda_n \in \mathbb{F}^n$ ciklično pojavi natanko enkrat.

Če se v vsaki periodi zaporedja $\boldsymbol{a}$ vsaka neničelna k-terica $\lambda_1, \lambda_2, \dots, \lambda_k \in \mathbb{F}^k$ ciklično pojavi natanko q^{n-k}-krat, k-terica ničel pa $(q^{n-k}-1)$-krat, potem pravimo, da ima zaporedje **idealno porazdelitev k-teric**.

Trditev 49. [31, Proposition 4.1] *Če ima zaporedje* $\boldsymbol{a}$ *lastnost* **R-4**, *potem ima idealno porazdelitev* k*-teric za vsak* k, $1 \leq k \leq n$.

Dokaz. Ker se vsaka neničelna n-terica v periodi ciklično pojavi natanko enkrat in ker lahko vsako neničelno k-terico $\lambda_1, \lambda_2, \ldots, \lambda_k$ razširimo na q^{n-k} načinov do n-terice $\lambda_1, \lambda_2, \ldots, \lambda_k, \lambda_{k+1}, \ldots, \lambda_n$, je torej q^{n-k} k-teric $\lambda_1, \lambda_2, \ldots, \lambda_k$ v zaporedju $\boldsymbol{a}$. Ker pa se ničelna n-terica v periodi ciklično ne pojavi in ker lahko ničelno k-terico razširimo na $q^{n-k}-1$ načinov do neničelne n-terice $\underbrace{0, \ldots, 0}_{k}, \lambda_{k+1}, \ldots, \lambda_n$, je torej $q^{n-k}-1$ ničelnih k-teric v zaporedju $\boldsymbol{a}$. ■

Trditev 50. [31, Property 4.1] *Če je* $\boldsymbol{a}$ *m-zaporedje, potem ima lastnost* **R-4**.

Dokaz. Ker je $\boldsymbol{a}$ m-zaporedje, ima periodo dolgo $q^n - 1$. Če se katera od neničelnih n-teric ne pojavi v periodi, potem je perioda zaporedja krajša od $q^n - 1$. In tudi če se katera od n-teric pojavi v periodi dvakrat, je perioda krajša od $q^n - 1$, saj so členi okoli poljubne n-terice enolično določeni s karakterističnim polinomom zaporedja in to n-terico. ■

Če združimo zadnji dve trditvi, je naslednja posledica očitna.

Posledica 51. *Če je* $\boldsymbol{a}$ *m-zaporedje, potem ima idealno porazdelitev* k*-teric za vsak* k, $1 \leq k \leq n$. ■

Če v prejšnji posledici vzamemo $k = 1$, dobimo:

Posledica 52. *Če je* $\boldsymbol{a}$ *m-zaporedje, potem ima lastnost* **R-1**. ■

Naslednjo posledico bomo potrebovali pri dokazovanju leme 55, s pomočjo katere bomo pokazali, da ima vsako m-zaporedje lastnost **R-3**.

Posledica 53. *Naj bo* $E = \mathrm{GF}(q^n)$ *in* $\mathbb{F} = \mathrm{GF}(q)$. *Potem ima za vsak* $c \in \mathbb{F}$ *in neničeln* $\beta \in E$ *enačba*

$$\mathrm{Tr}_{E/\mathbb{F}}(\beta x) = c$$

q^{n-1} *rešitev v* E.

Dokaz. Naj bo $\boldsymbol{b}$ m-zaporedje v $\mathbb{F}$ s periodo $q^n - 1$. Po trditvi 48 lahko to zaporedje predstavimo kot $b_i = \mathrm{Tr}_{E/\mathbb{F}}(\gamma\alpha^i)$, kjer je $f(x)$ nerazcepen polinom stopnje n z ničlo α, ki določa razširitev E obsega $\mathbb{F}$. Ker ima zaporedje $\boldsymbol{b}$ po zadnji trditvi lastnost **R-4**, se vsak $c \in \mathbb{F}$ v zaporedju pojavi natanko q^{n-1}-krat, to je $c = b_{i_1} = b_{i_2} = \ldots = b_{i_{q^{n-1}}}$. Torej so rešitve zgornje enačbe natanko $x \in \{\beta^{-1}\gamma\alpha^{i_1}, \beta^{-1}\gamma\alpha^{i_2}, \ldots, \beta^{-1}\gamma\alpha^{i_{q^{n-1}}}\}$, saj nam elementi $\gamma\alpha^i$, $0 \leq i \leq q^n - 1$, predstavljajo vse neničelne elemente v E. ■

Trditev 54. [31, Property 4.4] *Če je* $\boldsymbol{a}$ *m-zaporedje, potem ima lastnost* **R-2**.

Dokaz. Dokažimo vse štiri zahteve za lastnost **R-2**

1. Za vsak k, $1 \leq k \leq n-2$, lahko tvorimo n-terico s stikom dolžine k elementa λ od drugega mesta naprej,

$$(\eta_0, \underbrace{\lambda, \lambda, \ldots, \lambda}_{k}, \eta_{k+1}, \underbrace{\eta_{k+2}, \ldots, \eta_{n-1}}_{n-k-2}),$$

na $(q-1)^2 q^{n-k-2}$ načinov, če zahtevamo, da $\eta_0 \neq \lambda$ in $\eta_{k+1} \neq \lambda$. Ker ima m-zaporedje lastnost **R-4**, se vsaka neničelna n-terica ciklično pojavi natanko enkrat v periodi in zato se stik dolžine k elementa λ ciklično pojavi $(q-1)^2 q^{n-k-2}$-krat v periodi.

2. Za vsak $\lambda \neq 0$ lahko tvorimo n-terico s stikom dolžine $n-1$ elementa λ od drugega mesta naprej,
$$(\eta, \underbrace{\lambda, \lambda, \ldots, \lambda}_{n-1}),$$
na $(q-1)$ načinov, če zahtevamo, da je $\eta \neq \lambda$. Po lastnosti **R-4** se vsaka zgornja n-terica ciklično pojavi natanko enkrat v periodi. Ker se torej v periodi pojavi tudi n-terica $(\lambda, \lambda, \ldots, \lambda)$, se ena od $q-1$ zgoraj naštetih n-teric v zaporedju ciklično nadaljuje z λ in zato ne predstavlja cikličnega stika dolžine $n-1$. Zato se ciklični stik dolžine $n-1$ elementa λ pojavi $(q-2)$-krat v periodi.

3. Za $\lambda = 0$ velja podobno kot v točki 2, le da se n-terica ne more nadaljevati z 0, saj perioda ne vsebuje ciklične n-terice ničel. Zato se ciklični stik dolžine $n-1$ elementa 0 pojavi $(q-1)$-krat v periodi.

4. To je ravno lastnost **R-4**.

$\blacksquare$

S pomočjo naslednje leme bomo pokazali, da ima vsako m-zaporedje lastnost **R-3**.

Lema 55. *Kanonična aditivna karakteristika χ ima lastnosti*
$$\sum_{x \in \mathrm{GF}(q^r)} \chi(\beta x) = \begin{cases} 0 & , \quad \beta \in \mathbb{F},\ \beta \neq 0 \\ p^r & , \quad \beta \in \mathbb{F},\ \beta = 0 \end{cases} \qquad \textit{in}$$
$$\chi(x)\overline{\chi(y)} = \chi(x-y) \quad , \quad x, y \in \mathbb{F}.$$

Dokaz. Naj bo $\omega = e^{2\pi i/p}$. Potem je ω primitivni p-ti koren enote, torej $\omega^p - 1 = 0$ in zato
$$\sum_{i=0}^{p-1} \omega^i = 0\,.$$

Če je $\beta \neq 0$, ima po posledici 53 enačba $\mathrm{Tr}_p^r(\beta x) = y$ natanko p^{r-1} rešitev v $\mathbb{F} = \mathrm{GF}(p^r)$ pri izbranem $y \in \mathrm{GF}(p)$. Torej ima natanko p^{r-1} elementov iz $\mathrm{GF}(p^r)$ sled 0, natanko p^{r-1} sled 1, ... in natanko p^{r-1} elementov sled $p-1$. Zato je
$$\sum_{x \in \mathbb{F}} \chi(\beta x) = \sum_{x \in \mathbb{F}} e^{2\pi i\, \mathrm{Tr}_p^r(\beta x)/p} = \sum_{x \in \mathbb{F}} \omega^{\mathrm{Tr}_p^r(\beta x)} = p^{r-1} \sum_{i=0}^{p-1} \omega^i = 0\,.$$

Če pa je $\beta = 0$, je
$$\sum_{x \in \mathbb{F}} \chi(\beta x) = \sum_{x \in \mathbb{F}} e^{2\pi i\, \mathrm{Tr}_p^r(\beta x)/p} = \sum_{x \in \mathbb{F}} e^0 = p^r\,.$$

Z upoštevanjem linearnosti sledi dobimo za vsak $x, y \in \mathbb{F}$ tudi
$$\chi(x)\overline{\chi(y)} = e^{2\pi i\, \mathrm{Tr}_p^r(x)/p} \cdot e^{-2\pi i\, \mathrm{Tr}_p^r(y)/p} = e^{2\pi i (\mathrm{Tr}_p^r(x) - \mathrm{Tr}_p^r(y))/p} = e^{2\pi i (\mathrm{Tr}_p^r(x-y))/p} = \chi(x-y)\,.$$

$\blacksquare$

Trditev 56. [31, Property 4.5] *Če je* $\boldsymbol{a}$ *m-zaporedje, potem ima lastnost* **R-3**.

Dokaz. Naj ponovimo oznake, ki jih bomo uporabljali. Zaporedje $\boldsymbol{a}$ ima elemente iz obsega $\mathbb{F} = \mathrm{GF}(q)$, ki je razširitev obsega $K = \mathrm{GF}(p)$, $q = p^r$, reprezentacija s sledjo $a_i = \mathrm{Tr}_p^r(\beta\alpha^i)$ pa poteka v razširitvenem obsegu $E = \mathrm{GF}(q^n)$, saj je perioda zaporedja enaka $q^n - 1$. Označili smo tudi $\omega = e^{2\pi i/p}$. Z upoštevanjem definicije $C_{\boldsymbol{a}}$, zadnje leme, predstavitve zaporedja s sledjo, linearnosti sledi, definicijo $y = \beta(1-\alpha^\tau) \neq 0$, definicijo funkcije χ, tranzitivnosti sledi, ponovno definicijo funkcije χ, predzadnje leme, ter še dejstva $y = 0 \Leftrightarrow \alpha^\tau = 1 \Leftrightarrow \tau \equiv 0 \pmod{q^n - 1}$, vse to v naštetem vrstnem redu, dobimo

$$\begin{aligned}
C_{\boldsymbol{a}}(\tau) &= \sum_{i=0}^{q^n-2} \chi(a_i)\overline{\chi(a_{i+\tau})} = \sum_{i=0}^{q^n-2} \chi(a_i - a_{i+\tau}) = \\
&= \sum_{i=0}^{q^n-2} \chi\big(\mathrm{Tr}_{E/\mathbb{F}}(\beta\alpha^i) - \mathrm{Tr}_{E/\mathbb{F}}(\beta\alpha^{i+\tau})\big) = \\
&= \sum_{i=0}^{q^n-2} \chi\big(\mathrm{Tr}_{E/\mathbb{F}}(\beta(1-\alpha^\tau)\alpha^i)\big) = \sum_{i=0}^{q^n-2} \chi\big(\mathrm{Tr}_{E/\mathbb{F}}(y\alpha^i)\big) = \\
&= \sum_{x \in E^*} \omega^{\mathrm{Tr}_{\mathbb{F}/K}(\mathrm{Tr}_{E/\mathbb{F}}(y\,x))} = -1 + \sum_{x \in E} \omega^{\mathrm{Tr}_{E/K}(y\,x)} = \\
&= -1 + \sum_{x \in E} \chi(y\,x) = -1 + \begin{cases} 0 & , \ y \neq 0 \\ p^{nr} & , \ y = 0 \end{cases} \\
&= \begin{cases} q^n - 1 & , \ \tau \equiv 0 \pmod{q^n - 1} \\ -1 & , \ \text{sicer} \end{cases} .
\end{aligned}$$

Seveda bi bil dokaz za primer binarnega zaporedja $\boldsymbol{a}$ mnogo krajši. ■

4.5 Berlekamp-Masseyev algoritem

V tem razdelku bomo predstavili Masseyevo različico Berlekampovega algoritma. Da sta si problema iskanja rešitve ključne enačbe (2.14) s strani 52 in iskanja minimalnega polinoma danega zaporedja ekvivalentna, lahko vidimo, če na primer pogledamo matrični reprezentaciji sistema enačb danih problemov. V obeh primerih iščemo polinome minimalnih stopenj, ki ustrezajo rešitvi danega sistema. Če razmišljamo v smislu enačb, polinomov ali še bolje aproksimacij funkcij, pa lahko delimo ključno enačbo s σ in dobimo

$$1 + S_B \equiv \frac{\omega_B}{\sigma} \pmod{z^\delta},$$

kar pomeni, da iščemo (m,n)-**Padejevo aproksimacijo** funkcije $1+S_B$, kjer je $m = \ell - 1$ in $n = \ell$ in ℓ število neničelnih elementov vektorja napake. Pri Padejevi aproksimaciji želimo dano funkcijo čimbolje aproksimirati z racionalno funkcijo s predpisanima omejitvama stopnje števca in imenovalca. Več o tem si lahko preberemo v Bini in Pan [11], kjer pa avtorja ne omenjata hitrega Berlekamp-Masseyevega algoritma za reševanje, ampak počasnejšo različico z Evklidovim algoritmom. Če smo v obsegu $\mathbb{Q}$, ki ni končen, se izkaže Berlekamp-Masseyev algoritem za neprimerno hitrejšega od Evklidovega algoritma, prav tako pa tudi pri končnih obsegih.

Massey je opazil prednosti Berlekampovega algoritma za reševanje ključne enačbe pri dekodiranju BCH kod in je napisal članek [60], v katerem je predstavil za malenkost modificiran Berlekampov algoritem za izračun minimalnega polinoma danega zaporedja. Ta algoritem sedaj imenujemo Berlekamp-Masseyev algoritem (v nadaljevanju BM algoritem). BM algoritem je iterativen in kot vhodni podatek dobi binarno zaporedje $\boldsymbol{a}^N$ dolžine N, na n-tem koraku, $n < N$, pa nam vrne najkrajši LFSR, ki generira $\boldsymbol{a}^n$. Algoritem bomo izpeljali, njegovo pravilnost pa bomo dokazali neodvisno od dokaza pravilnosti Berlekampovega algoritma. Dokaz bomo povzeli po Gong [31, Chapter 6].

Če obstaja začetno stanje $(a_0, a_1, \ldots, a_{\ell-1})$, da je prvih N členov LFSR generatorja dolžine ℓ s karakterističnim polinomom $f(x)$ enakih

$$\boldsymbol{a}^N = a_0, a_1, \ldots, a_{N-1}\,,$$

potem pravimo, LFSR (f, ℓ) **generira** zaporedje $\boldsymbol{a}^N$. Za LFSR generator dolžine ℓ s karakterističnim polinomom $f(x)$ stopnje ℓ bomo uporabljali oznako (f, ℓ). Naslednja trditev je očitna posledica zadnje definicije.

Trditev 57. *Če je $\ell \geq N$, potem (f, ℓ) vedno generira zaporedje $\boldsymbol{a}^N$. Če pa je $\ell < N$, potem pa (f, ℓ) generira zaporedje $\boldsymbol{a}^N$ natanko tedaj, ko je*

$$a_{k+\ell} = \sum_{i=0}^{\ell-1} c_i a_{k+i} \qquad k = 0, 1, \ldots, N - \ell - 1\,, \tag{4.3}$$

kjer je $f(x) = x^\ell + \sum_{i=0}^{\ell-1} c_i x^i$ karakteristični polinom generatorja (recipročni od povratnega polinoma). ■

Linearno zahtevnost končnega zaporedja bomo označili z $LS(\boldsymbol{a}^N)$ in definirali kot dolžino najkrajšega LFSR, ki generira zaporedje $\boldsymbol{a}^N$, torej je

$$LS(\boldsymbol{a}^N) = \min_{f\in\mathbb{F}[x]} \{0 \leq \ell < \infty | (f, \ell) \text{ generira } \boldsymbol{a}^N\}\,.$$

Ker $(x^N + 1, N)$ vedno generira $\boldsymbol{a}^N$, je $LS(\boldsymbol{a}^N) \leq N$.

Trditev 58. *Če je $\ell < N$, potem pa (f, ℓ) generira zaporedje $\boldsymbol{a}^N$ natanko tedaj, ko je*

$$f(L)\boldsymbol{a} = (\underbrace{0, \ldots, 0}_{b=N-\ell}, *, *, \ldots)\,.$$

Dokaz. Ničle pomenijo, da enačba (4.3) drži pri indeksih $k = 0, 1, \ldots, N - \ell - 1$. ■

Zaporedju $\boldsymbol{a} = (a_0, \ldots, a_{N-1}, 0, 0, \ldots)$ pravimo **razširitev z ničlami** zaporedja $\boldsymbol{a}^N$. V BM algoritmu bomo v vsakem koraku potrebovali tudi izračun **določitvene spremenljivke** (bita pri $\mathbb{F} = \mathbb{Z}_2$, next discrepancy bits = naslednji protislovni bit) zaporedja $\boldsymbol{a}$ glede na polinom f:

$$\delta_{k+\ell} = a_{k+\ell} + \sum_{i=0}^{\ell-1} c_i a_{k+i} \qquad k = 0, 1, \ldots, N - \ell - 1\,.$$

V drugi obliki zapisano to pomeni

$$f(L)\boldsymbol{a} = (\delta_\ell, \delta_{\ell+1}, \ldots)\,.$$

Kaj nam predstavljajo določitvene spremenljivke, si oglejmo v naslednji lemi, v kateri bomo za zapis delovanja $f(L)$ na zaporedju $\boldsymbol{a}$ potrebovali tudi oznako

$$\boldsymbol{e}_k = (\underbrace{0, \ldots, 0}_{k}, 1, *, \ldots)\,,$$

ki nam predstavlja poljubno zaporedje, ki ima na prvih k mestih ničle, nato 1 in naprej poljubne elemente. Bodimo pozorni, $\boldsymbol{e}_k - \boldsymbol{e}_k$ nam ne predstavlja vedno ničelnega zaporedja, ima pa vedno na prvih $k+1$ mestih ničle.

Lema 59. *Naj bo $\boldsymbol{a}$ razširitev z ničlami zaporedja $\boldsymbol{a}^N$ in $n < N$. Naslednje tri izjave so ekvivalentne:*

1. *LFSR (f, ℓ) generira zaporedje $a_0, a_1, \ldots, a_{n-1}$, ne pa tudi zaporedja $a_0, a_1, \ldots, a_n$,*
2. *za določitvene spremenljivke velja $\delta_{k+\ell} = 0$, $k = 0, \ldots, n-1-\ell$, in $\delta_n \neq 0$,*
3. *$f(L)\boldsymbol{a} = (\underbrace{0, \ldots, 0}_{b=n-\ell}, \delta_n, *, \ldots) = \delta_n \boldsymbol{e}_b$, kjer je $\delta_n \neq 0$.*

Dokaz. ($1 \Leftrightarrow 2$): po definiciji δ_k, saj za vsak k velja $\delta_{k+\ell} = 0 \iff a_{k+\ell} = \sum_{i=0}^{\ell-1} c_i a_{k+i}$
($1 \Leftrightarrow 3$): uporabimo zadnjo trditev. ■

Z izračunom $f(L)\boldsymbol{e}_k$ lahko dokažemo naslednjo lemo.

Lema 60. *Naj bo f moničen polinom stopnje ℓ in $\ell \leq k$. Potem je*

$$f(L)\boldsymbol{e}_k = (\underbrace{0, \ldots, 0}_{k-\ell}, 1, *, \ldots) = \boldsymbol{e}_{k-\ell}\,.$$

■

Izrek 61 (Berlekamp-Masseyev algoritem). [31, Theorem 6.1] *Naj bo $\boldsymbol{a}^N$ končno zaporedje elementov iz $\mathbb{F} = \mathrm{GF}(q)$ in $N > 0$. Potem lahko določimo naslednji iteracijski postopek za konstruiranje zaporedja LFSR-jev (f_n, ℓ_n), kjer (f_n, ℓ_n) generira zaporedje $\boldsymbol{a}^{n+1} = a_0, \ldots, a_n$ za $n = 0, \ldots, N-1$.*

1. *Naj bo n_0 prvi neničelen element v $\boldsymbol{a}^N$. Določimo*

   $$\begin{array}{rcrl} f_0 = f_1 = \cdots = f_{n_0-1} = 1 & \textit{in} & \ell_0 = \ell_1 = \cdots = \ell_{n_0-1} = 0 & ,\ \textit{če je } n_0 > 0 \quad \textit{in} \\ f_{-1} = 1 & \textit{in} & \ell_{-1} = 0 & ,\ \textit{če je } n_0 = 0\,, \end{array}$$

   ter

   $$f_{n_0}(x) = x^{n_0+1} + 1 \quad \textit{in} \quad \ell_{n_0} = n_0 + 1\,.$$

2. *Predpostavimo, da smo že konstruirali $f_0, f_1, \ldots, f_{n-1}$, kjer LFSR (f_{n-1}, ℓ_{n-1}) generira zaporedje $a_0, \ldots, a_{n-1}$. Označimo*

   $$f_{n-1}(x) = x^{\ell_{n-1}} + \sum_{i=0}^{\ell_{n-1}-1} c_{n-1,i} x^i$$

   ter določitveno spremenljivko

   $$d_n = a_n + \sum_{i=0}^{\ell_{n-1}-1} c_{n-1,i} a_{n-\ell_{n-1}+i}\,.$$

(a) *Če je* $d_n = 0$, *potem LFSR* (f_{n-1}, ℓ_{n-1}) *generira tudi zaporedje* $a_0, \ldots, a_n$ *in vzamemo* $f_n = f_{n-1}$ *in* $\ell_n = \ell_{n-1}$.

(b) *Če je* $d_n \neq 0$, *potem LFSR* (f_{n-1}, ℓ_{n-1}) *generira zaporedje* $a_0, \ldots, a_{n-1}$, *ne pa tudi zaporedja* $a_0, \ldots, a_n$. *Naj bo* m *zadnji indeks, kjer je LFSR spremenil dolžino (dobro definiran, saj je* $n_0 \leq m < n$*), torej je*

$$\ell_{m-1} < \ell_m = \cdots = \ell_{n-1}\,.$$

Naj bo

$$a = m - \ell_{m-1} \quad \textit{in} \quad b = n - \ell_{n-1}\,.$$

i. *Če je* $a \geq b$, *potem določimo*

$$f_n(x) = f_{n-1}(x) - \delta_n \delta_m^{-1} x^{a-b} f_{m-1}(x) \quad \textit{in} \quad \ell_n = \ell_{n-1}\,.$$

ii. *Če je* $a < b$, *potem določimo* $(\ell_n > \ell_{n-1})$

$$f_n(x) = x^{b-a} f_{n-1}(x) - \delta_n \delta_m^{-1} f_{m-1}(x) \quad \textit{in} \quad \ell_n = \ell_{n-1} + b - a\,.$$

Dokaz.

1. Po definiciji je linearna zahtevnost ničelnega zaporedja 0, minimalni polinom pa 1.
2. (a) Neposredna posledica trditve 57.
   (b) Po lemi 59 je $f_{n-1}(L)\boldsymbol{a} = \delta_n \boldsymbol{e}_{n-\ell_{n-1}}$ in $f_{m-1}(L)\boldsymbol{a} = \delta_m \boldsymbol{e}_{m-\ell_{m-1}}$.
      i. Če je $a \geq b$, potem je $L^{a-b} f_{m-1}(L)\boldsymbol{a} = \delta_m \boldsymbol{e}_{n-\ell_{n-1}}$ in zato $\big(f_{n-1}(L) - L^{a-b} f_{m-1}(L)\big)\, \boldsymbol{a} = \delta_n \boldsymbol{e}_{n-\ell_{n-1}} - \delta_n \boldsymbol{e}_{n-\ell_{n-1}} = (\underbrace{0, \ldots, 0}_{n-\ell_{n-1}}, 0, *, *, \ldots)$, torej LFSR $(f_{n-1}(x) - \delta_n \delta_m^{-1} x^{a-b} f_{m-1}(x), \ell_{n-1})$ generira tudi zaporedje $a_0, \ldots, a_n$ in je res dolžine ℓ_{n-1}, saj je $\deg\big(x^{a-b} f_{m-1}(x)\big) = \ell_{n-1} - (n-m) < \ell_{n-1}$.
      ii. Analogno kot (i).

■

Primer. Za dano zaporedje $\boldsymbol{a}^{10} = 0111\ 1110\ 11$ določimo linearno zahtevnost in LFSR te dolžine, ki generira $\boldsymbol{a}^{10}$. Ali je ta LFSR enoličen?

zap. $\boldsymbol{a}^{10}$	ind. n	d.b. δ_n	$LS(\boldsymbol{a}^n)$ ℓ_n	LFSR f_n	a	b
0	0		0	1		
1	1		2	x^2+1	1	0
1	2	1	2	x^2+x+1	1	1
1	3	1	2	x^2+x	1	2
1	4	0	2	x^2+x	1	3
1	5	0	2	x^2+x	1	4
1	6	0	2	x^2+x	1	5
0	7	1	6	x^6+x^5+1	5	2
1	8	0	6	x^6+x^5+1	5	3
1	9	1	6	$x^6+x^5+x^4+x^3+1$	5	4

Berlekamp-Masseyev algoritem za izračun minimalnega polinoma $f(x)$ zaporedja $\boldsymbol{a}^N$

```
    Input: zaporedje a^N = a_0, a_1, ..., a_{N-1};
    Output: LFSR (f_{N-1}, ℓ_{N-1});
    Velja: na koncu n-tega koraka je
        f(x) minimalni polinom zaporedja a^{n+1} = a_0, ..., a_n in
        ℓ linearna zahtevnost zaporedja a^{n+1}.
1.  k = 0;
    while (a_k = 0 and k ≤ N − 1) begin
        k = k + 1;
    end;
2.  f(x) = x^{k+1} − 1;
    ℓ = k + 1;
    g(x) = 1;
    c = a_k;
    a = k;
    b = 0;
    T(x) = 0;
3.  for n = k + 1 to N − 1 do begin
        d = a_n + Σ_{i=0}^{ℓ−1} c_i a_{n−ℓ+i};
4.      if d = 0 then b = b + 1;
        else
5.      if 2ℓ > n then begin
            f(x) = f(x) − d c^{−1} x^{a−b} g(x);
            b = b + 1;
        end
6.      else begin
            T(x) = f(x);
            f(x) = x^{b−a} f(x) − d c^{−1} g(x);
            c = d;
            ℓ = n + 1 − ℓ;
            g(x) = T(x);
            a = b;
            b = n − ℓ + 1;
        end;
    end;
7.  return (f(x), ℓ)
```

Slika 4.2: Berlekamp-Masseyev algoritem (LFSR Synthesis)

Iz zadnje vrstice vidimo, da je linearna zahtevnost celotnega zaporedja enaka 6, karakteristični polinom minimalnega LFSR-ja, ki generira $\boldsymbol{a}^{10}$ je $f(x) = x^6 + x^5 + x^4 + x^3 + 1$ in ker $2 \cdot 6 \not\leq 10$, tak LFSR ni enolično določen (po izreku 69, ki še sledi). ▲

Masseyev [60] algoritem je skoraj ekvivalenten Berlekampovemu, le da ne uporablja funkcije $B(k)$, saj nas, glede na oznake iz Berlekampovega algoritma, rešitev ω ne zanima in tudi ne zahtevamo, da je njena stopnja enaka $\ell - 1$, lahko je nižja. Masseyev [60] algoritem se od na sliki 4.2 opisanega razlikuje le po tem, da išče minimalni povratni polinom, torej recipročni od minimalnega karakterističnega.

Dokazali smo že, da so z algoritmom najdeni polinomi karakteristični polinomi za dani del celotnega zaporedja. Pokažimo še, da so to minimalni polinomi.

Lema 62. [31, Lemma 6.3] *Naj LFSR* (f_{n-1}, ℓ_{n-1}) *generira zaporedje* $a_0, \dots, a_{n-1}$*, ne pa tudi zaporedja* $a_0, \dots, a_n$*, in naj LFSR* (f_n, ℓ_n) *generira zaporedje* $a_0, \dots, a_n$*. Potem velja*

$$\ell_n \geq n + 1 - \ell_{n-1} \qquad n \geq 1\,.$$

Dokaz. S protislovjem. Recimo, da je $\ell_n \leq n - \ell_{n-1}$. Po lemi 42 (delovanje polinomov na zaporedjih) je $f_{n-1}(L)f_n(L)\boldsymbol{a} = f_n(L)f_{n-1}(L)\boldsymbol{a}$, toda zlahka preverimo, da se zaporedji ne ujemata pri prvem neničelnem bitu od $f_n(L)f_{n-1}(L)\boldsymbol{a} = \boldsymbol{e}_{(n-\ell_{n-1})-\ell_n}$ (upoštevamo lemo 42 in lemo 60), saj je indeks vektorja $e_{n-\ell_{n-1}} - e_n$ dobro definiran. ∎

Če poleg predpostavk leme predpostavimo še $\ell_n \geq \ell_{n-1}$, je $\ell_n \geq \max\{\ell_{n-1}, n+1-\ell_{n-1}\}$, kar lahko uporabimo v naslednjih dveh trditvah.

Trditev 63. *Naj algoritem A generira tako zaporedje LFSR* (f_n, ℓ_n)*, da ima za vsak* $n \geq 0$ *LFSR* (f_n, ℓ_n) *minimalno dolžino izmed vseh LFSR-jev, ki generirajo* $\boldsymbol{a}^n$*. Potem velja*

1. *Če LFSR* (f_{n-1}, ℓ_{n-1}) *generira tudi zaporedje* $a_0, \dots, a_n$*, potem je* $\ell_n = \ell_{n-1}$*.*
2. *Če LFSR* (f_{n-1}, ℓ_{n-1}) *ne generira tudi zaporedja* $a_0, \dots, a_n$*, potem je* $\ell_n \geq \max\{\ell_{n-1}, n + 1 - \ell_{n-1}\}$*.* ∎

Trditev 64. [31, Theorem 6.2] *Če je zaporedje LFSR* (f_n, ℓ_n) *generirano z BM algoritmom iz izreka 61, potem velja*

1. *Če LFSR* (f_{n-1}, ℓ_{n-1}) *generira tudi zaporedje* $a_0, \dots, a_n$*, potem je* $\ell_n = \ell_{n-1}$*.*
2. *Če LFSR* (f_{n-1}, ℓ_{n-1}) *ne generira tudi zaporedje* $a_0, \dots, a_n$*, potem je* $\ell_n = \max\{\ell_{n-1}, n + 1 - \ell_{n-1}\}$*.*

Dokaz. 1. Očitno (točka 2.a v algoritmu)
2. Po poslidici leme 62 velja $\ell_n \geq \max\{\ell_{n-1}, n + 1 - \ell_{n-1}\}$. Z indukcijo pokažimo, da je ta maksimum dosežen. Indukcijska predpostavka: 'trditev drži za vse $m < n$'. Baza indukcije: korak 1 algoritma ($m = n_0$). Indukcijski korak: Po pogojih trditve smo v algoritmu v točki 2.b. Če smo v točki 2.b.i, je $\ell_n = \ell_{n-1}$. Če pa smo v 2.b.ii, pa je $\ell_n = b - a + \ell_{n-1} = (n - \ell_{n-1}) - (m - \ell_{m-1}) + \ell_{n-1}$, kjer je m zadnji indeks pred n, kjer je $\ell_{m-1} < \ell_m$. Ker je $\ell_{m-1} \neq \ell_m$, je po indukcijski predpostavki $\ell_m = m + 1 - \ell_{m-1}$, kar vstavimo v $\ell_n = n - (m - \ell_{m-1}) = n - (\ell_m - 1)$. Upoštevamo še, da je $\ell_{n-1} = \ell_m$, in dobimo $\ell_n = n + 1 - \ell_{n-1}$. ∎

Naslednji izrek je posledica zadnjih dveh trditev.

Izrek 65. *Za zaporedje LFSR* (f_n, ℓ_n)*, generirano z BM algoritmom, velja, da je za vsak* $n \geq 0$ *LFSR* (f_n, ℓ_n) *LFSR z minimalno dolžino izmed vseh LFSR-jev, ki generirajo* $\boldsymbol{a}^{n+1}$*.*

Posledica 66. ℓ_n *je linearna zahtevnost zaporedja* $a_0, \dots, a_n$

Posledica 67. *Za BM algoritem velja:*

1. $0 \leq \ell_0 \leq \ell_1 \leq \cdots \leq \ell_n \leq \cdots$
2. $\ell_n = \ell_{n-1}$ *ali* $\ell_n = n + 1 - \ell_{n-1}$*.*

3. *Če je* $2\ell_{n-1} > n$, *potem je* $\ell_n = \ell_{n-1}$.

4. *Če je*
$$\ell_{m-1} < \ell_m = \cdots = \ell_{n-1} < \ell_n\,,$$
potem je
$$n - m = \ell_n - \ell_{m-1}\,.$$

Dokaz. (1) in (2) očitno.
(3) Če je $\ell_{n-1} \neq \ell_n$, potem je $\ell_{n-1} < \ell_n$ in po točki točki (2) zadnje trditve je $\ell_{n-1} < n + 1 - \ell_{n-1}$, kar pomeni $2\ell_{n-1} < n + 1$, oziroma $2\ell_{n-1} \leq n$.
(4) Z upoštevanjem točke (2) izračunamo $\ell_n - \ell_{n-1}$. ■

Izpeljimo še izrek, v katerem bomo pokazali, koliko zaporednih bitov potrebujemo za rekonstrukcijo LFSR.

Lema 68. *V koraku 2.b v BM algoritmu iz izreka 61 je pogoj* $a \geq b$ *ekvivalenten pogoju* $2\ell_{n-1} > n$.

Dokaz. Iz koraka 2.b vidimo $a \geq b \iff \ell_n = \ell_{n-1}$. Torej je dovolj pokazati $\ell_n = \ell_{n-1} \iff 2\ell_{n-1} > n$.
($\Longrightarrow$) V enačbi $\ell_n = \max\{\ell_{n-1}, n + 1 - \ell_{n-1}\}$ upoštevamo $\ell_n = \ell_{n-1}$ in dobimo $\ell_{n-1} = \max\{\ell_{n-1}, n + 1 - \ell_{n-1}\}$. Torej je $\ell_{n-1} \geq n + 1 - \ell_{n-1}$, kar pomeni $2\ell_{n-1} \geq n + 1 > n$.
($\Longleftarrow$) Upoštevamo točko (3) v posledici 67. ■

Izrek 69. [31, Theorem 6.3] *Če poženemo BM algoritem na zaporedju* $a_0, \ldots, a_{N-1}$ *in če nam algoritem vrne LFSR* (f, ℓ), *kjer je* $2\ell \leq N$, *potem je* f *edini karakteristični polinom (minimalne) stopnje* ℓ *za kakršnokoli razširitev zaporedja* $\boldsymbol{a}^N$.

Dokaz. S pomočjo pogoja $2\ell \leq N$ lahko zaporedje $a_0, \ldots, a_{N-1}$ le na en način razširimo do zaporedja $\boldsymbol{a} \in V(\mathbb{F})$, da mu linearne zahtevnosti ne povečamo (ker je $2\ell \leq N$, je v BM algoritmu $a < b$, in ker zahtevamo isti ℓ naprej, se a ne bo spreminjal, b pa se bo le poveceval, torej se korak (b) algoritma ne sme (ne more) izvesti kar pomeni, da se lahko izvaja le korak (a)). Nato določimo enolični minimalni polinom zaporedja $\boldsymbol{a}$. Ker ni drugih možnosti za razširitev do $\boldsymbol{a} \in V(\mathbb{F})$, tudi ni drugih možnosti za f. ■

Posledica 70. 2ℓ *bitov je dovolj za rekonstrukcijo LFSR, ki je generiral zaporedje* $\boldsymbol{a}$ *z linearno zahtevnostjo* $LS(\boldsymbol{a}) = \ell$.

4.6 Pričakovana linearna zahtevnost in njena varianca

Izpeljali bomo pričakovano linearno zahtevnost naključnega zaporedja z elementi iz obsega $\mathbb{F} = \mathbb{Z}_2$, kot je to predstavil Masseyev študent Rueppel [80] na Eurocryptu ’85. Za splošen RNG nad GF(q) pa najdemo podobne rezultate v Gustavson [37].

Naj bo $\boldsymbol{a}^n$ zaporedje dolžine n in $LS(\boldsymbol{a}^n)$ njegova linearna zahtevnost. Označimo z $N_n(L)$ število vseh zaporedij dolžine n z linearno zahtevnostjo L :
$$N_n(L) = |\{\boldsymbol{a}^n; LS(\boldsymbol{a}^n) = L\}|\,.$$

Trditev 71. *Za zaporedje $N_n(L)$ velja naslednja rekurzivna formula*

$$N_{n+1}(L) = \begin{cases} 2N_n(L) + N_n(n+1-L) & ; \quad n < 2L-1 \\ 2N_n(L) & ; \quad n = 2L-1 \\ N_n(L) & ; \quad n \geq 2L \geq 0 \end{cases} .$$

S temi relacijami in začetnimim členom $N_1(0) = 1$ ter členi $N_n(n) = 1$ in seveda $N_n(L) = 0$ za $n < L$ je zaporedje natanko določeno.

Dokaz. Če je $2L > n$, vemo iz lastnosti BM algoritma, da ima katerokoli nadaljevanje $\boldsymbol{a}^{n+1}$ binarnega zaporedja $\boldsymbol{a}^n$ z linearno zahtevnostjo L tudi linearno zahtevnost L. Če pa je imel predhodnik od $\boldsymbol{a}^{n+1}$ linearno zahtevnost manjšo od L, jo je imel lahko le $n+1-L$, saj se na tak način spreminja linearna zahtevnost v BM algoritmu. Da pride do skoka linearne zahtevnosti pri $\boldsymbol{a}^n$, pa je možno le v primeru, ko zaporedje $\boldsymbol{a}^n$ nadaljujemo tako, da je $d \neq 0$, pogoj $2L > n$ pa je izpolnjen po predpostavki. Torej je v tem primeru $N_{n+1}(L) = 2N_n(L) + N_n(n+1-L)$. Izjema v tem primeru je le pri $n = 2L-1$, saj v tem primeru ne velja $n+1-L < L$ in je zato $N_{n+1}(L) = 2N_n(L)$. Če pa je $2L \leq n$, potem lahko katerokoli binarno zaporedje $\boldsymbol{a}^n$ z linearno zahtevnostjo L na en sam način nadaljujemo tako, da se linearna zahtevnost ne spremeni. In sicer tako, da je določitveni bit d v BM algoritmu enak 0. Torej je $N_{n+1}(L) = N_n(L)$. Začetni pogoji pa tudi držijo, saj je (0) edino zaporedje dolžine 1 z linearno zahtevnostjo 0 ter (0,...,0,1) edino zaporedje dolžine n z linearno zahtevnostjo n. ∎

Iz naslednje tabele lahko uganemo splošno formulo za $N_n(L)$.

L \ n	1	2	3	4	5	6	7	8	9	10
0	1	1	1	1	1	1	1	1	1	1
1	1	2	2	2	2	2	2	2	2	2
2		1	4	8	8	8	8	8	8	8
3			1	4	16	32	32	32	32	32
4				1	4	16	64	128	128	128
5					1	4	16	64	256	512
6						1	4	16	64	256
7							1	4	16	64
8								1	4	16
9									1	4
10										1

Trditev 72. *Za zaporedje $N_n(L)$ velja*

$$N_n(L) = \begin{cases} 2^{\min\{2n-2L, 2L-1\}} & ; \quad n \geq L > 0 \\ 1 & ; \quad n > L = 0 \end{cases} .$$

Dokaz. Z upoštevanjem zgornje formule dobimo $N_n(n+1-L) = 2^{\min\{2L-2, 2n-2L+1\}}$. Če je $n < 2L-1$, potem je $2n - 2L < 2(2L-1) - 2L = 2L-2$ in zato $N_n(n+1-L) = 2^{2n-2L+1}$. Prav tako dobimo tudi $N_n(L) = 2^{\min\{2n-2L,2L-1\}} = 2^{2n-2L}$ in $N_{n+1}(L) = 2^{2n-2L+2}$. Torej drži enačba $N_{n+1}(L) = 2N_n(L) + N_n(n+1-L)$. Če je $n = 2L-1$, potem je $N_n(L) = 2^{\min\{2L-2,2L-1\}} = 2^{2L-2}$ in $N_{n+1}(L) = 2^{\min\{2L,2L-1\}} = 2^{2L-1}$, torej drži tudi relacija $N_{n+1}(L) = 2N_n(L)$. Če pa je $n \geq 2L$, potem pa je $N_n(L) = 2^{\min\{2n-2L,2L-1\}} = 2^{2L-1}$ in $N_{n+1}(L) = 2^{\min\{2n-2L+2,2L-1\}} = 2^{2L-1}$, torej drži tudi zadnja relacija $N_{n+1}(L) =$

$N_n(L)$ iz prejšnje trditve. Očitno ustreza predpis tudi začetnima členoma, kar pomeni, da predpis v tej trditvi res določa zaporedje $N_n(L)$. ■

Če pogledamo zadnji stolpec tabele elementov $N_n(L)$, lahko vidimo, da ima večina zaporedij dolžine n linearno zahtevnost približno $n/2$. Naslednji izrek bo pokazal, da je pričakovana linearna zahtevnost za velike n približno enaka $n/2+2/9$, če je n sodo, oziroma $n/2+5/18$, če je n liho število. Pričakovan kvadrat odstopanja linearne zahtevnosti od pričakovane pa je za velike n približno enak $86/81$. To na primer pomeni, da če imamo PRBG, ki nam producira zaporedja dolžine 1000 z linearno zahtevnostjo razporejeno po intervalu $[445, 505]$, je z njim verjetno nekaj močno narobe. Vsekakor bi bil algoritem, ki nam s pomočjo BM algoritma za dano zaporedje izračuna linearno zahtevnost in nam vrne 1, če je ta zahtevnost različna od $n/2$, ε-razlikovalec med naklučno enakomerno porazdelitvijo in porazdelitvijo, ki jo ima naš PRBG. V našem primeru bi bil zaradi tako nizke variance, to je $86/81$, razlikovalec zelo dober, z ε-om zelo blizu optimalnemu, to je $1/2$.

Izrek 73. *Linearna zahtevnost enakomerno naključno izbranega (generiranega z RBG) binarnega zaporedja $\boldsymbol{a}^n$ dolžine n ima matematično upanje (pričakovano vrednost)*

$$E[LS(\boldsymbol{a}^n)] = \frac{n}{2} + \frac{4 + n \bmod 2}{18} - 2^{-n}\left(\frac{n}{3} + \frac{2}{9}\right)$$

in varianco

$$\operatorname{var}[LS(\boldsymbol{a}^n)] = \frac{86}{81} - 2^{-n}\left(\frac{14 - n \bmod 2}{27}n + \frac{82 - 2(n \bmod 2)}{81}\right) - 2^{-2n}\left(\frac{1}{9}n^2 + \frac{4}{27}n + \frac{4}{81}\right).$$

Dokaz. Izračunajmo

$$E[LS(\boldsymbol{a}^n)] = \sum_{\boldsymbol{a}^n} LS(\boldsymbol{a}^n)P(\boldsymbol{a}^n) = \sum_{L=0}^{n} L\frac{N_n(L)}{2^n} = 2^{-n}\left(1 + \sum_{L=1}^{n} L\, 2^{\min\{2n-2L, 2L-1\}}\right).$$

Ker je $2n-2L \le 2L-1$ natanko tedaj ko je $L \ge (2n+1)/4$, je

$$E[LS(\boldsymbol{a}^n)] = 2^{-n}\left(\sum_{1\le L<(2n+1)/4} L\, 2^{2L-1} + \sum_{(2n+1)/4\le L\le n} L\, 2^{2n-2L}\right).$$

Za izračun vsot bomo potrebovali formulo za izračun vsote $\sum_{j=1}^{m} jx^j$, ki jo dobimo s standardnimi analitičnimi metodami za izračun vsot s pomočjo odvajanja:

$$\sum_{j=1}^{m} jx^j = x\left(\sum_{j=1}^{m} x^j\right)' = x\left(\frac{x^{m+1}-1}{x-1} - 1\right)' = \frac{x(m\, x^{m+1} - (m+1)x^m + 1)}{(x-1)^2}.$$

Sedaj vsoto za $E[LS(\boldsymbol{a}^n)]$ prilagodimo za izračun

$$E[LS(\boldsymbol{a}^n)] = 2^{-n}\left(\frac{1}{2}\sum_{1\le L<(2n+1)/4} L\, 4^L + 2^{2n}\sum_{(2n+1)/4\le L\le n} L\,(1/4)^L\right).$$

V primeru, ko je n sodo število, je indeksacija prve vsote $\sum_{L=1}^{n/2}$, druge pa $\sum_{L=n/2+1}^{n}$. V primeru, da je n liho, pa imamo vsoti $\sum_{L=1}^{(n-1)/2}$ in $\sum_{L=(n+1)/2}^{n}$. Če upoštevamo še formulo za izračun $\sum_{j=1}^{m} jx^j$ pri $x=4$, oziroma $1/4$, dobimo

$$E[LS(\boldsymbol{a}^n)] = \frac{n}{2} + \frac{2}{9} - 2^{-n}\left(\frac{n}{3} + \frac{2}{9}\right)$$

za sode n in

$$E[LS(\boldsymbol{a}^n)] = \frac{n}{2} + \frac{5}{18} - 2^{-n}\left(\frac{n}{3} + \frac{2}{9}\right)$$

za lihe n.

Za izračun variance $\mathrm{var}[LS(\boldsymbol{a}^n)] = E[(LS(\boldsymbol{a}^n))^2] - (E[LS(\boldsymbol{a}^n)])^2$ pa moramo najprej izračunati $E[(LS(\boldsymbol{a}^n))^2]$. Izračun je analogen zgornjemu, le da si tu pomagamo z vsoto

$$\begin{aligned}\sum_{j=1}^{m} j^2 x^j &= x\Big(\sum_{j=1}^{m} jx^j\Big)' = x\bigg(x\Big(\sum_{j=1}^{m} x^j\Big)'\bigg)' = x\bigg(x\Big(\frac{x^{m+1}-1}{x-1} - 1\Big)'\bigg)' = \\ &= \frac{x(m^2x^{m+2} - (2m^2+2m-1)x^{m+1} + (m+1)^2x^m - x - 1)}{(x-1)^3}.\end{aligned}$$

Pri izračunu

$$E[(LS(\boldsymbol{a}^n))^2] = 2^{-n}\left(\frac{1}{2}\sum_{1\le L<(2n+1)/4} L^2\, 4^L + 2^{2n}\sum_{(2n+1)/4\le L\le n} L^2\,(1/4)^L\right)$$

zopet upoštevamo, da je v primeru, ko je n sodo število, indeksacija prve vsote $\sum_{L=1}^{n/2}$, druge pa $\sum_{L=n/2+1}^{n}$. V primeru, da je n liho, pa imamo zopet vsoti $\sum_{L=1}^{(n-1)/2}$ in $\sum_{L=(n+1)/2}^{n}$. Sedaj upoštevamo še formulo za izračun $\sum_{j=1}^{m} j^2x^j$ pri $x = 4$, oziroma $1/4$, in dobimo

$$E[(LS(\boldsymbol{a}^n))^2] = \frac{1}{4}n^2 + \frac{2}{9}n + \frac{10}{9} - 2^{-n}\left(\frac{1}{3}n^2 + \frac{8}{9}n + \frac{10}{9}\right)$$

za sode n in

$$E[(LS(\boldsymbol{a}^n))^2] = \frac{1}{4}n^2 + \frac{5}{18}n + \frac{41}{36} - 2^{-n}\left(\frac{1}{3}n^2 + \frac{8}{9}n + \frac{10}{9}\right)$$

za lihe n, kar pomeni, da je

$$E[(LS(\boldsymbol{a}^n))^2] = \frac{1}{4}n^2 + \frac{4 + n \bmod 2}{18}n + \frac{40 + n \bmod 2}{36} - 2^{-n}\left(\frac{1}{3}n^2 + \frac{8}{9}n + \frac{10}{9}\right)$$

in če sedaj izračunamo $\mathrm{var}[LS(\boldsymbol{a}^n)]$ kot $\mathrm{var}[LS(\boldsymbol{a}^n)] = E[(LS(\boldsymbol{a}^n))^2] - (E[LS(\boldsymbol{a}^n)])^2$, dobimo željeno formulo iz izjave izreka. ■

Za konstrukcijo standardnega algoritma za preverjanje linearne zahtevnosti in ocenjevanje odstopanja od pričakovane, moramo danemu zaporedju določiti tako imenovano p-vrednosti, glej Novak [71]. En način bi bila uporaba Čebiševe neenakosti:

$$P\left(\Big|LS(\boldsymbol{a}^n) - E[LS(\boldsymbol{a}^n)]\Big| \ge k\right) \le \frac{\mathrm{var}[LS(\boldsymbol{a}^n)]}{k^2},$$

ki je kot ocena za p-vrednost pri merjenju linearne zahtevnosti neuporabna, saj je

$$[N_n(L)]_{L=0}^{n} = [1, 2, 8, 32, \ldots, 2\cdot 4^{n/2-1}, 4^{n/2-1}\ldots, 64, 16, 4, 1]$$

za sode n in

$$[N_n(L)]_{L=0}^{n} = [1, 2, 8, 32, \ldots, 2\cdot 4^{(n-1)/2-1}, 4^{(n-1)/2}\ldots, 64, 16, 4, 1]$$

za lihe n, kar pomeni, da je zgornja verjetnost približno enaka

$$P\left(\Big|LS(\boldsymbol{a}^n) - E[LS(\boldsymbol{a}^n)]\Big| \geq k\right) \approx 1 - \sum_{i=1}^{2k} 2^{-i} = 2^{-2k}\,.$$

Pri $k = 10$, dobimo s Čebiševo oceno, da ima največ 1 odstotek zaporedij dolžine n linearno zahtevnost, ki odstopa od pričakovane za več kot 10. Prava ocena je seveda, da je delež takih zaporedij 0.000001. Za preverjanje naključnosti zaporedja $\boldsymbol{a}^n$ bi zaporedju določili p-vrednost

$$p = 2^{-2|LS(\boldsymbol{a}^n)-n/2|}\,,$$

statistični test pa bi pri primerno majhnji stopnji značilnosti testa α zavrnil tista zaporedja, pri katerih bi bilo $p < \alpha$.

Ker nam BM algoritem omogoča sprotno merjenje, oziroma spreminjanje linearne zahtevnosti, moramo narediti algoritem, ki nam to sprotno preverjanje tudi izvaja. Povzeli bomo dva testa, katerih utemeljitve lahko najdemo v Carter [16]. Vsakemu indeksu k, $0 \leq k \leq n-1$, za katerega je $L(\boldsymbol{a}^k) > L(\boldsymbol{a}^{k-1})$ pri računanju linearne zahtevnosti zaporedja $\boldsymbol{a}$, rečemo **skok** (jump).

Testiranje števila skokov. Število vseh skokov označimo z F. Za velike n je slučajna spremenljivka F porazdeljena po zakonu $N\left(\frac{1}{4}n, \frac{1}{8}n\right)$, zato je statistika

$$\frac{F - \frac{1}{4}n}{\sqrt{\frac{1}{8}n}}$$

potem porazdeljena standardno normalno. Slaba zaporedja so in tista s premalo in tudi s preveč skoki, saj sicer lahko vedno s pomočjo BM algoritma sestavimo napovedovalca naslednjega bita. Zato bomo za kritično območje vzeli unijo intervalov $[0, \frac{\alpha}{2}] \cup [1 - \frac{\alpha}{2}, 1]$.

Testiranje porazdelitve skokov. Z f_h označimo število skokov linearne zahtevnosti z višino h. Višina skoka pri indeksu i je definirana kot $h_i = L(s_i) - L(s_{i-1})$. Verjetnost, da ima skok višino h, je $p_h = (\frac{1}{2})^h$. Matematično upanje za število skokov višine h je enako $E_h = p_h \cdot F$. Največjo višino skoka h, za katero je $E_h > 5$ označimo z m. Statistika

$$\chi^2 = \sum_{h=1}^{m} \frac{(f_h - E_h)^2}{E_h}$$

je pri zaporedju, generiranim s pravim RBG, porazdeljena po zakonu $\chi^2(m-1)$. Za spremenljivko

$$\alpha = \int_0^{\chi^2} f_{\chi^2(m-1)}(t)dt$$

vzamemo kritično območje $[1-\alpha, 1]$.

Vemo, da imajo LFSR zaporedja, ki jih uporabljamo v praksi, periodo dolgo $2^n - 1$, kjer je n dolžina zaporedja. S postopki nelinearizacije, ki jih bomo opisali v naslednjem razdelku, ohranimo periodo, izhodno zaporedje pa postane nelinearno. Toda! Ali je varno uporabljati večkratne ponovitve periode brez strahu, da bi obstajala možnost obstoja ε-razlikovalca od porazdelitve RBG. Rueppel [80, str. 52] je pokazal, da če sestavimo zaporedje $\boldsymbol{a} = a_0, a_1, \ldots$ kot periodično s periodo $\boldsymbol{a}^T$ dolžine $T = 2^n$, potem je pričakovana linearna zahtevnost zaporedja $\boldsymbol{a}$ enaka

$$E[LS(\boldsymbol{a})] = 2^n - 1 + 2^{-2^n}\,.$$

To pomeni, da ne smemo uporabljati več kot dve periodi zaporedja skupaj, saj potem linearna zahtevnost izhodnega zaporedja ni več ustrezna. Če ne prej, moramo PRBG ponovno reinicializirati vsaj po $2T$ izhodnih bitih.

4.7 Nelinearizacija LFSR generatorja

Ker nam linearnost LFSR povzroči, da LFSR ni več kriptografsko varen PRBG, bi radi naredili podoben PRBG, tako da bi ohranili dobre osnovne statistične lastnosti izhodnega zaporedja, linearnost pa bi odpravili. Na slikah 4.3 in 4.4 sta prikazana dva splošna koncepta, kako to storiti s pomočjo nelinearne funkcije, na sliki 4.5 pa kako to storiti na način, da en LFSR generator kontrolira drugega. V priročniku [64, Chapter 6] so opisani tudi konkretni primeri in podane reference za napade na njih.

Naj bo $\boldsymbol{a} = a_0, a_1, a_2, \ldots$ m-zaporedje nad obsegom $\mathbb{F}$ z linearno zahtevnostjo n. Izberimo m tap-položajev $0 \le d_1 < d_2 < \cdots < d_m \le n-1$ in označimo $\mathcal{T} = \{d_1, \ldots, d_m\}$. S funkcijo $f : \mathbb{F}^m \to \mathbb{F}$ imamo določeno **filtrirno zaporedje**,

$$s_i = f(a_{i+d_1}, a_{i+d_2}, \ldots, a_{i+d_m}), \qquad i \ge 0,$$

na sliki 4.3 pa imamo prikazan ustrezen FSR generator, takoimenovan **filtrirni generator**. Naštejmo nekaj dejstev, ki veljajo za binarna filtrirna zaporedja (brez dokaza)

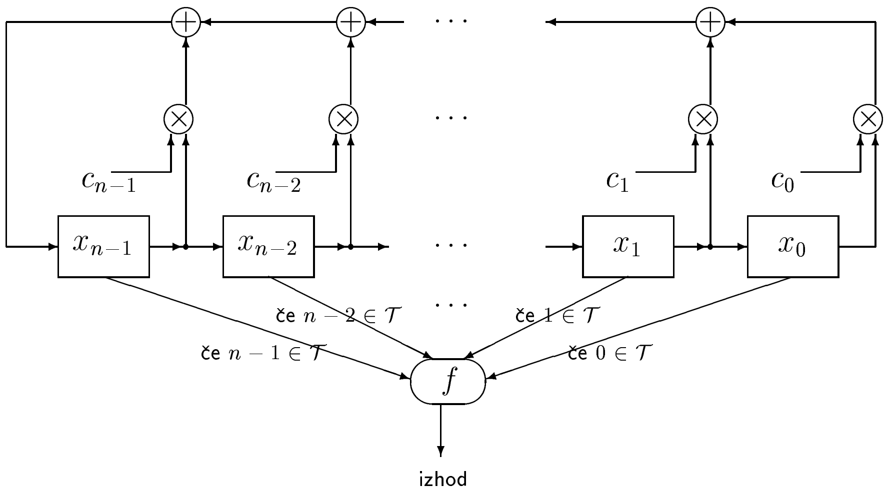
1. perioda je enaka $2^n - 1$,
2. linearna zahtevnost je navzgor omejena z
$$LS(\boldsymbol{s}) \le \sum_{k=1}^{m} \binom{n}{k},$$
glej Key [51], vendar pa ni znane konstrukcije kako se približati tej meji,
3. za določen izbor tap-položajev imamo linearno zahtevnost navzdol omejeno z
$$LS(\boldsymbol{s}) \ge \binom{n}{m},$$
to se zgodi na primer pri izboru, pri katerem je $d_i - d_{i-1} = d$, $2 \le i \le m$, pri čemer je $d > 0$; drugi način pa je izbor $d_i = 2^{i-1}$, $1 \le i \le m$, pri čemer mora biti $2^{m-1} < n$,
4. ostale statistične lastnosti še niso raziskane.

Drug način uporabe funkcije $f : \mathbb{F}^m \to \mathbb{F}$ pa je naslednja konstrukcija. Naj bodo $\boldsymbol{a}_j$ m-zaporedja z linearno zahtevnostjo n_j, $1 \le j \le m$. Zaporedje

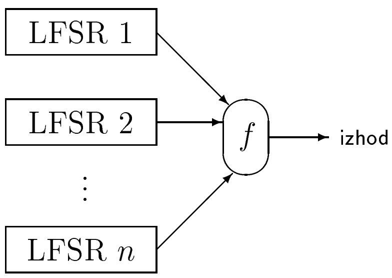
$$s_i = f(a_{1,i}, a_{2,i}, \ldots, a_{m,i}), \qquad i \ge 0,$$

imenujemo **zaporedje dobljeno s kombinatorno funkcijo**. Na sliki 4.4 imamo prikazan ustrezen FSR generator, takoimenovan **generator s kombinatorno funkcijo**.

Naštejmo nekaj dejstev, ki veljajo za binarna zaporedja, dobljena s kombinatorno funkcijo (brez dokaza)



Slika 4.3: Filtrirni generator



Slika 4.4: Generator s kombinatorno funkcijo

1. perioda zaporedja deli $\prod_{j=1}^{m}(2^{n_j}-1)$ in je enaka

$$\mathrm{per}(\boldsymbol{s}) = \prod_{j=1}^{m}(2^{n_j}-1)$$

natanko tedaj, ko si si $n_1,\ldots,n_j$ paroma tuja,

2. ni znanih splošnih rezultatov za ostale statistične lastnosti pri poljubni funkciji f.

Funkcija f mora biti uravnotežena, glej lastnost **R-1**, in imeti mora nizko korelacijo v primerjavi z vsemi afinimi funkcijami (takoimenovana nelinearnost). Tako preprečimo **korelacijski napad**, glej Siegenthaler [84].

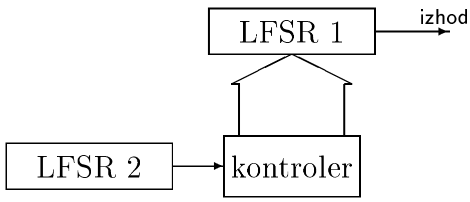
Če zaporedje $\boldsymbol{b}$ določa, kateri členi zaporedja $\boldsymbol{a}$ bodo nastopili v izhodnem zaporedju $\boldsymbol{s}$ in na katerem mestu, potem takšnemu FSR generatorju rečemo **generator, kontroliran z uro** in je shematsko prikazan na sliki 4.5. Primer takega generatorja je **stop-and-go generator**, pri katerem je izhodno zaporedje določeno z

$$s_i = a\left(\sum_{j=0}^{i} b_j\right), \qquad i \geq 0\,.$$

Če je na primer $\boldsymbol{a} = a_0, a_1, a_2, \ldots$ in $\boldsymbol{b} = 1001011$, potem je $\boldsymbol{s} = a_1, a_1, a_1, a_2, a_2, a_3, a_4, \ldots$. Drug primer generatorja, kontroliranega z uro, je **stiskalni generator** (Shrinking Generator), pri katerem je izhodno zaporedje določeno kot

$$s_i = a\left(\min\left\{k \,\middle|\, \sum_{j=0}^{k} b_j = i\right\}\right), \qquad i \geq 0\,.$$

Če je na primer $\boldsymbol{a} = a_0, a_1, a_2, \ldots$ in $\boldsymbol{b} = 1001011$, potem je $\boldsymbol{s} = a_0, a_3, a_5, a_6, \ldots$.



Slika 4.5: Generator, kontroliran z uro

Poglavje 5

Testiranje varnosti tokovnih šifer

V tem razdelku bomo predstavili osnovne teste, katerim morajo zadoščati tokovne šifre, in napade, proti katerim morajo biti odporne. Kriteriji so razumni, vendar tako strogi, da do sedaj v kriptografski javnosti še ni bila predstavljena tokovna šifra, ki bi bila hitrejša od bločnih in bi zadoščala vsem varnostnim kriterijem. Varnost do sedaj predstavljenih tokovnih šifer temelji na varnosti bločnih ali pa ne zadoščajo v celoti vsem statističnim testom, oziroma proti njim obstaja napad, ki omogoča odkritje ključa hitreje od metode preizkusa vseh ključev (brute force attack). Vse to ne pomeni, da tokovnih šifer ni varno uporabljati, ampak da kriptografska javnost čaka na tako šifro, da se bo kot standard lahko obdržala mnogo let.

Tokovne šifre so nepogrešljive pri telekomunikacijah, tako na primer GSM aparati uporabljajo različice šifre A5 (A5/0,A5/1,A5/2,A5/3), ki pa ima mnogo slabosti, ki jih bomo predstavili na koncu razdelka. Pri zaščiti multimedijskih izdelkov (glasba, filmi, pay-for-view TV ipd) ni nujno šifrirati celotne vsebine, saj je dovolj, če šifriramo na primer pas širok 5 cm na televizijskem ekranu. Zato v tem primeru ni nujna uporaba tokovnih šifer. Standarda na tem področju industrije še ni, seveda pa je potreben, saj bodo morali proizvajalci multimedijskih aparatov (radio, TV, DVD ipd) omogočiti uporabo algoritmov in kompatibilnost aparatov s standardi.

V zadnjih treh letih (2000–2003) se je v Evropski uniji izvajal **projekt NESSIE**[1] (New European Schemes for Signatures, Integrity and Encryption), analogija ameriškega projekta AES (Advanced Encryption Standard), ki je v letu 2000 standardiziral bločno šifro Rijndael. Na projekt so se odzvali posamezniki, raziskovalne skupine in podjetja, ki so podali 42 predlogov za kriptografske algoritme (bločne šifre, tokovne šifre, MAC algoritmi in zgoščevalne funkcije (hash functions), digitalni podpisi, identifikacijske sheme), ki so bili na štirih delavnicah (workshops) predstavljeni ter potem še do potankosti stestirani. Kakšni testi so bili izvedeni in kakšni napadi preizkušeni bomo opisali v naslednjih podrazdelkih. Na spletni strani projekta lahko dobimo poročilo *NESSIE security report* [69], v katerem so opisani predlogi algoritmov s predstavitvami, zahteve ter konkretni napadi na izločene algoritme. Za tokovne šifre bomo nekatere teste in napade bolj podrobno opisali v tem poglavju. NESSIE za uporabo ne priporoča nobenega od predstavljenih petih predlogov z utemeljitvijo, da je algoritem ali prepočasen, ali pa da ni odporen proti kateremu od poznanih napadov.

Statistične nepravilnosti izhodnega zaporedja nam omogočijo razvitje primernega napada na šifro. Na ta način razvite metode ponavadi klasificiramo kot

[1] www.cryptonessie.org

Razlikovanje od Bernoulijeve porazdelitve (Distinguishing Attack)
Metoda, s katero preverjamo, ali je izhodno zaporedje generatorja naključno. Ugotavljamo, ali za generator obstaja ϵ-razlikovalec od Bernoulijeve porazdelitve (neodvisni meti poštenega kovanca), glej razdelek 3.4.

Napoved naslednjega bita (Prediction)
Metoda, s katero preverjamo, ali lahko iz predhodnih bitov izhodnega zaporedja generatorja napovemo naslednji bit z večjo verjetnostjo, kot je ugibanje. Ugotavljamo, ali za generator obstaja ϵ-napovedovalec $\boldsymbol{B}_i$ naslednjega bita, glej razdelek 3.4.

Razkritje ključa (Key Recovery)
Metoda, s katero ugotavljamo, ali se da iz izhodnega zaporedja generatorja ugotoviti skrivni ključ.

Opisane metode seveda temeljijo na **Kerckhoffovem principu**, ki pravi, da naj pri šifriranju ne bo tajen algoritem, ki ga uporabljamo, temveč le ključ preslikave. Od zgoraj opisanih metod je namočnejše razkritje ključa, saj če lahko iz izhodnega zaporedja ugotovimo ključ, potem nam algoritem ne zagotavlja nikakršne varnosti. Prvi dve metodi pa nam data zelo močno informacijo za implementacijo konkretnega napada.

5.1 Kriptoanaliza tokovnih šifer

Za posamezno tokovno šifro niso vsi napadi primerni, kakor tudi ne vsi uspešni. Klapper [52] je leta 1996 dokazal obstoj razreda tokovnih šifer, ki so odporne proti vsem determinističnim napadom. Pri tem je naredil dober model definicij, ki povejo, kdaj je generator varen, hiter, primerne dolžine ipd. Pomankljivost tega razreda je, da generatorji niso odporni proti vsem napadom, torej tudi proti mnogokrat uporabljenim napadom z verjetnostnimi algoritmi. Na podlagi ideje napada, lahko le te klasificiramo na naslednje skupine.

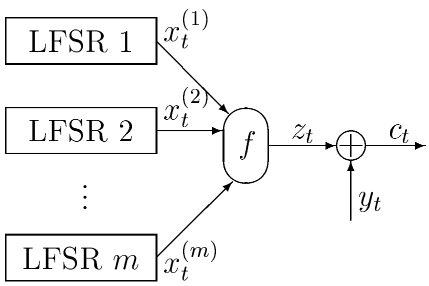
Preizkus vseh ključev (Exhaustive Key Search)
To je napad, s katerim preizkusimo vse možne ključe. Uporaben je pri vseh kriptografskih algoritmih, saj je najprimitivnejši napad. Dolžina skrivnega ključa naj bi bila tolikšna, da je ključ nemogoče ugotoviti s preizkusom vseh ključev in primerjati dobljeno izhodno zaporedje z danim. Zelo uporaben je napad tipa **time-memory tradeoff**, s katerim si shranimo nekaj relacij med ključi in njim prirejenimi izhodnimi zaporedji generatorja v spomin. Tako si pri konkretni izvedbi napada prihranimo nekaj časa, ko nekaj podatkov o ključu dobimo s pomočjo podatkov iz spomina celoten ključ pa dobimo s preizkusom ostalih možnosti. Za tokovne šifre sta izvedbo tega napada opisala Biryukov in Shamir [12].

Napadi pogojeni s periodo in statističnimi lastnostmi
(Periodic and Statistical Attacks)
Pri nekaterih sinhronih šifrah, glej sliko 3.3 na strani 76, na primer pri generatorju s kombinatorno funkcijo, glej sliko 4.3 na strani 105, je izhodno zaporedje periodično, zato je pri daljši komunikaciji/šifriranju potrebno menjavanje ključa. Seveda mora izhodno zaporedje zadoščati statističnim testom, da preprečimo napade tipa ϵ-razlikovalec in ϵ-napovedovalec.

Linearna zahtevnost (Linear Complexity $\sim$ Linear Span)
Linearno zahtevnost ter z njo povezano število ter porazdelitev njenih skokov, opisano na strani 103, lahko določimo s pomočjo Berlekamp-Masseyevega algoritma, glej stran 97. Če je linearna zahtevnost premajhna, lahko reproduciramo celotno zaporedje kot LFSR zaporedje.

Zahtevnost največjega reda (Maximum Order Complexity)
Linearna zahtevnost je zahtevnost prvega reda. Če iščemo najkrajši FSR generator, ki generira dano zaporedje s povratno funkcijo, ki bi imela monome največ n-tega reda, pravimo, da določamo **zahtevnost n-tega reda**. Zahtevnost največjega reda pa je dolžina najkrajšega FSR generatorja, ki generira dano zaporedje. Prav tako kot pri testiranju linearne zahtevnosti tudi tu določimo število ter porazdelitev skokov zahtevnosti.

Korelacijski napad (Correlation Attacks)
To je tip napada, ki je največkrat uspešno uporabljen proti generatorjem, ki uporabljajo LFSR. Pri generator s kombinatorno funkcijo, predstavljenem na sliki 4.4 na strani 105, je mnogokrat moč najti korelacijo med zaporedjem, ki ga producira eden od LFSR generatorjev ter izhodnim zaporedjem. Hitro implementacijo (deli in vladaj) te ideje je predstavil Siegenthaler [84], Meier in Stafellbach [63], Filiol [24], Palit in Roy [72] ter mnogi drugi pa so te ideje še izboljšali ter predstavili hitre implementacije napada. Napad je zelo močan, saj na podlagi le dela tajnopisa lahko brez poznavanja ustreznega čistopisa in brez razkritja ključa ugotovimo začetno stanje generatorja in s tem celoten čistopis. Na spodnji sliki ponovno prikazujemo generator.



Opis ideje napada: Na podlagi n bitov tajnopisa $\boldsymbol{c}^n$, kombinatorne funkcije f ter korelacijske šibkosti želimo odkriti začetno stanje generatorja, ki s kombinatorno funkcijo združuje m LFSR-jev. Z $\boldsymbol{x}^{(i)}$ označimo zaporedje, ki ga generira i-ti LFSR, z $\boldsymbol{z}$ izhodno zaporedje celotnega generatorja, z $\boldsymbol{y}$ čistopis in s $\boldsymbol{c}$ tajnopis. Privzemimo, da poznamo

1. povratne polinome vseh LFSR-jev,
2. n bitov tajnopisa,
3. $P(y_t = 0) = p_0 > \frac{1}{2}$ (čistopis ni naključen),
4. $P(z_t \oplus x_t^{(i)}) = q_i > \frac{1}{2}$, $1 \le i \le m$ (korelacijska šibkost).

Če fiksiramo začetno stanje i-tega LFSR in označimo z $\boldsymbol{X}^{(i)}$ njegovo izhodno zaporedje ter $p_i = P(c_t = X_t^{(i)})$, potem je

$$\begin{aligned} p_i &= P(y_t \oplus z_t = X_t^{(i)}) \\ &= P(y_t = 0 \wedge z_t = X_t^{(i)}) + P(y_t = 1 \wedge z_t \neq X_t^{(i)}) \\ &= p_0\, P(z_t = X_t^{(i)}) + (1 - p_0)\,(1 - P(z_t = X_t^{(i)}))\,. \end{aligned}$$

Definirajmo

$$\beta = \sum_{t=1}^{n} c_t \oplus X_t^{(i)} .$$

Če smo uganili pravo začetno stanje i-tega LFSR, potem je $p_i = p_0 q_i + (1-p_0)(1-q_i)$ in slučajna spremenljivka β je porazdeljena $N(n(1-p_i), n(1-p_i)p_i)$, sicer pa je $p_i = \frac{1}{2}$ in porazdelitev β je blizu $N(n/2, n/4)$. S pomočjo vrednosti statistike β lahko določimo, ali smo uganili pravo začetno stanje i-tega LFSR.

Zeng et al. [97] so naredili LS (Linear Syndrome) algoritem za izvedbo napada, ki izkorišča korelacijsko šibkost. Prvih n bitov izhodnega zaporedja $\boldsymbol{z}$ generatorja zapišemo kot vsoto $\boldsymbol{z}^n = \boldsymbol{a}^n + \boldsymbol{x}^n$, kjer je $\boldsymbol{a}$ izhodno zaporedje LFSR generatorja in $\boldsymbol{x}^n$ vektor napake. S pomočjo metode dekodiranja s sindromi lahko določimo vektor napake, ki naj bi imel čimmanjšo težo. Zeng et al. [99] so algoritem tudi pospešili (ILS – Improved LS Algorithm). ILS algoritem je bil uspešno uporabljen proti Geffe-jevemu in Beth-Piperjevemu generatorju.

Korelacijski napad višjega reda (Higher Order Correlation Attacks)
Korelacijski napad poskuša najti aproksimacijo izhodnega zaporedja FSR generatorja z linearnim, to je LFSR, generatorjem. Lahko pa poskušamo aproksimirati izhodno zaporedje z generatorjem višjega rada. S tem dobimo predefiniran sistem enačb z več neznankami. Za rešitev teh enačb lahko uporabimo XL metodo, glej Courtois et al. [20]. Bolj podroben opis tega napada najdemo v Courtois [19].

Napadi tipa deli in vladaj / Napadi z ugibanjem dela ključa
(Divide-and-Conquer Attacks / Sub-Key Guessing Attacks)
Napad lahko uporabimo kot pospešitev preizkusa vseh ključev. Na vsakem koraku fiksiramo del ključa, nato pa iz danega zaporedja določimo preostanek ključa. Napad je uspešen, če je določitev preostanka ključa hitrejše od preizkusa vseh možnosti.

Pri nekaterih tokovnih šifrah je mogoče z ugibanjem dela ključa ugotoviti celoten ključ z izvajanjem določenega konsistenčnega testa, glej Zeng et al. [98]. Tako je pri generatorjih baziranih na LFSR-jih mogoče s fiksiranjem dela ključa na K_1 mogoče sestaviti linearen sistem enačb $A(K_1)x = b$, kjer je b izhodno zaporedje generatorja, x pa iskani preostanek ključa. Temu napadu sta podlegla Jenningov generator [44], Multiple Speed Generator ter Perfect Linear Cipher, za izvedbo napada glej Zeng et al. [98] Prav tako so Coppersmith et al. [18] idejo uspešno uporabili na tresočem generatorju, glej stran 106.

Napadi po menjavi ključev (Rekeying Attacks)
Mnogokrat je potrebna menjava ključev po določenem številu izhodnih bitov, recimo zaradi dolžine periode izhodnega zaporedja. Možno je, da algoritem za menjavo ključa ni varno izveden in da iz same menjave ugotovimo ključ.

Napadi po robu (Sidechannel Attacks)
Možno je, da algoritem zahteva večjo porabo energije (hardverska implementacija) ali pa časa pri določenih notranjih stanjih generatorja. Z uporabo te informacije lahko ugotovimo notranje stanje generatorja, kar je mnogokrat dovolj za napoved naslednjih bitov, lahko pa tudi razkritje ključa. Podrobnejše opise teh napadov najdemo v NESSIE porocilu.

5.2 Statistični testi

S pomočjo statističnih testov izhodnega zaporedja generatorja preverjamo, ali obstaja ϵ-razlikovalev od Bernoulijeve porazdelitve binarnega zaporedja. Na podlagi morebitnih slabosti, ki jih pokažejo testi, lahko določimo tip napada primernega za dano tokovno šifro, oziroma PRBG generator. Dostopnih je mnogo programskih paketov, ki vsebujejo implementacije teh testov, kot so Crypt-XS paket, Diehard paket, NESSIE paket in drugi. Testi iz prvih dveh paketov so opisani v Novak [71, 4. poglavje]. Statistične teste, ki jih bomo predstavili, pa večino najdemo v paketu NESSIE.

Test trkov (Collision Test)
Izhodno zaporedje razdelimo na posamezne bloke fiksnih dolžin. Če se določen blok ponovi več kot enkrat pravimo, da se je zgodil trk. S testom preverjamo, ali porazdelitev števila trkov naključna.

Test blokov in vrzeli (Constant Runs Test)
Test je opisan v Novak [71, razdelek 4.3.5], z njim pa preverimo porazdelitev blokov in vrzeli fiksnih dolžin, pri čemer je **blok** maksimalno dolgo podzaporedje enk in maksimalno dolgo podzaporedje ničel.

Avtokorelacijski test (Autocorelation Test)
Namen testa je preveriti podobnost med zaporedjem $\boldsymbol{s}$ in zaporedjem dobljenim s premikom zaporedja $\boldsymbol{s}$ za d mest (izbrisom prvih d členov), glej Novak [71, razdelek 4.1.5]. Z XOR operatorjem na istoležnih bitih prvotnega in novega zaporedja bi morali ponovno dobiti naključno zaporedje. Na tem zaporedju naredimo test pogostosti enk.

Test zbiranja nalepk (Coupon Collector's Test)
Dano binarno zaporedje razdelimo na bloke fiksnih dolžin. Preštejemo po koliko blokih so prisotni vsi možni bloki.

Test Dyad zahtevnosti (Dyadic Complexity Test)
Test te zahtevnosti sta predlagala Goretzky in Klapper [53]. Mnogokrat mu podležejo generatorji, ki hranijo carry bit, dobljen s pomočjo celoštevilskega izračuna funkcije iz prejšnje iteracije.

Hitri spektralni test (The Fast Spectral Test)
Test naredimo s pomočjo Walshove transformacije na danem binarnem zaporedju.

Test pogostosti (Frequency Test)
Izhodno zaporedje razdelimo na posamezne bloke fiksnih dolžin. Statistično preverimo porazdelitev blokov v zaporedju.
Monobit test. Cilj tega testa je ugotoviti, ali ima dano binarno zaporedje $\boldsymbol{s}$ približno toliko enk kot ničel, glej Novak [71, razdelek 4.1.1].
Test parov (two-bit test / serial test) Cilj tega testa je ugotoviti, ali ima dano zaporedje približno enako število pojavitev podzaporedij `00`, `01`, `10` in `11`, glej Novak [71, razdelek 4.1.2].

Test praznin (Gap Test)
Izhodno zaporedje razdelimo na posamezne bloke fiksnih dolžin. Posamezni blok

nam predstavlja neko naravno število. Vrzel je maksimalno podzaporedje, ki ne vsebuje števila iz nekega določenega intervala. Statistično preverimo porazdelitev dolžin vrzeli.

Test linearne zahtevnosti (Linear Complexity Test)
Test je pravzaprav že opisan napad z izračunom linearne zahtevnosti.

Test linearne zahtevnosti z napakami (Error Linear Complexity Test)
Linearna zahtevnost s k napakami (k-error linear complexity) zaporedja $\boldsymbol{a}$ je enaka minimumu linearnih zahtevnosti zaporedij, ki se od zaporedja $\boldsymbol{a}$ razlikujejo na največ k koordinatah. Označimo jo z $LS_k(\boldsymbol{a})$. Zaenkrat so predstavljeni le učinkoviti algoritmi za zaporedja s periodo $l = 2^n$, kar pa je premalo za kriptografsko uporabnost. Druga pomankljivost teh algoritmov pa je, da na vsakem koraku obdelujejo celotno zaporedje. Tako sta Games in Chan [27] leta 1983 predstavila algoritem za računanje linearne zahtevnosti binarnega zaporedja s periodo $l = 2^n$ v $O(l)$ času. Leta 1993 pa sta Stamp in Martin [86] razširila Games-Chanov algoritem na računanje linearne zahtevnosti s k napakami. Njun algoritem je časovne zahtevnosti $O(l \log l)$. Lauder in Paterson [55] sta ga leta 2000 razširila in predstavila algoritem za računanje **spektra linearne zahtevnosti s k napakami** (error linear complexity spectrum), definiranega kot $\{(k, LS_k(\boldsymbol{a}) \mid k \in \mathbb{N} \cup \{0\}\}$. Algoritem deluje le na binarnih zaporedjih s periodo $l = 2^n$ v $O(l)$ in potroši le $O(l(\log_2 l)^2)$ bitnih operacij.

Test aproksimacije višjega reda (Maximum Order Complexity Test)
Test je pravzaprav že opisan napad z izračunom zahtevnosti največjega reda.

Test prekrivajočih se m-teric (Overlaping m-tuple Test)
Iz zaporedja določimo porazdelitev vseh možnih m-teric (podzaporedij dolžine m).

Test pronicljivosti (Percolation Test)
Test je simulacija hitrosti razširjanja gozdnega požara, pri čemer nam dano zaporedje predstavlja položaje dreves.

Poker test (Poker Test)
Dano zaporedje razdelimo na bloke fiksnih dolžin, ki nam predstavljajo karte. Nato vzamemo k zaporednih kart, ki nam predstavljajo karte enega igralca. Karte določenega igralca nam predstavljajo neko vrednost (par, full, poker ipd). Testiramo porazdelitev posameznih vrednosti v danem zaporedju. Primer poker testa najdemo v Novak [71, razdelek 4.1.3].

Test rangov (Rank Test)
Z biti danega zaporedja napolnimo matrike. Nato izračunamo range teh matrik nad obsegom $\mathbb{Z}_2$ ter določimo porazdelitev dobljenih rangov. Diehard paket vsebuje več različnih testov rangov, glej Novak [71, razdelki 4.4.3, 4.4.4, 4.4.5].

Test naraščanja (Run Test)
Dano zaporedje razdelimo na bloke fiksnih dolžin, ki nam predstavljajo realna števila z intervala $[0, 1]$. Statistično preverimo porazdelitev maksimalnih naraščajočih podzaporedij (ang. run), glej Novak [71, razdelek 4.4.16].

Univerzalni Maurerov test (Universal Maurer Test)
Test razdeli dano zaporedje na bloke fiksnih dolžin ter izračuna razdaljo med posameznimi bloki. Rezultat testa je zelo povezan z entropijo danega zaporedja. Test je

analogija programa za arhiviranje (stiskanje, kompresiranje brez izgube) podatkov, kot je na primer WinZip. Test je opisan tudi v Novak [71, razdelek 4.3.8].

Test Ziv-Lempelove zahtevnosti (Ziv-Lempel Complexity Test)
Test razdeli dano zaporedje na bloke fiksnih dolžin ter preverimo hitrost pojavljanja novih vzorcev v zaporedju teh blokov.

Test pojavljanja vzorcev (Sequence complexity test)
S tem testom lahko nadomestimo osnovni autokorelacijski test, saj če zaporedje pade na autokorelacijskem testu, pade tudi na sequence complexity testu. Test določi število c različnih podzaporedij v zaporedju $\boldsymbol{s} = s_1, \dots, s_n$, ko ga pregledujemo od začetka do konca, podrobnejši opis najdemo v Novak [71, razdelek 4.3.6]. Test zavrne zaporedja, pri katerih je vrednost c v kritičnem območju, torej na intervalu $[c_{\mathrm{tv}}, 1]$, z argumentom, da so vzorčasta (patterned).

Test razpolavljanja (Change point test)
Test je bolj podrobno opisan v Novak [71, razdelek 4.3.3], preveri pa, ali ima zaporedje bitov $\boldsymbol{s} = s_1, \dots, s_n$ do nekega delilnega mesta t približno toliko enic kot po delilnem mestu. Zaporedje $\boldsymbol{s}$ lahko ustreza monobit testu, a pade na change point testu.

Poleg zgoraj opisanih testov, od katerih je večina v paketu testov uporabljenih pri ovrednotenju NESSIE predlogov, je v javnosti zelo poznan tudi **Diehard paket**, ki ga je opisal in implementiral Marsaglia [57, 58]. Diehard testi ne preverjajo osnovnih lastnosti, ampak bolj "neobičajne", podobne recimo zgoraj omenjen test pronicljivosti. Nekateri so posredno povezani z zgradbo mnogih generatorjev. Večina Diehard testov vrne vrednost spremenljivke p, ki naj bi bila slučajna in enakomerno porazdeljena na intervalu $[0, 1)$. To je pravzaprav statistična domneva, ki jo preverjamo. Ti p-ji so dobljeni iz porazdelitve statistike X, ki jo računamo v posameznem testu.

Tokovno šifro pa lahko testiramo tudi s testi, ki so predvideni predvsem za bločne šifre. To je pomembno predvsem pri inicializaciji začetnega notranjega stanja (**key loading**), oziroma pri algoritmu zamenjave ključa po določenem številu izhodnih bitov. Začetno notranje stanje generatorja mora biti neodvisno od skrivnega ključa. Podobno lahko teste, ki so razviti za tokovne šifre, uporabimo pri bločnih šifrah, kot je bilo to storjeno pri testiranju algoritma DES, glej Golomb in Gong [32].

Poglavje 6

PRAŠTEVILO je v P

Pri mnogih kriptografskih algoritmih, kot na primer RSA, DSA, ECDSA ipd, glej Stinson [87], je potrebno neprestano izbirati naključna praštevila. Običajno to storimo tako, da izberemo naključno število z določenega intervala, nato pa preverimo, ali je praštevilo. Zato je ključnega pomena, da znamo hitro odgovoriti na vprašanje, ali je dano število praštevilo (t.i. odločitveni problem **PRAŠTEVILO**). To seveda ni tako mlad problem, kot je s 27 leti mlada kriptografija z javnimi ključi, saj je okupiral matematike že v antičnih časih. Karl Friedrich Gauss (1777-1855), je v svoji knjigi Disquistiones Arithmeticae (1801) zapisal tudi: *"Menim, da čast znanosti narekuje, da z vsemi sredstvi iščemo rešitev tako elegantnega in tako razvpitega problema."* Od leta 1960 dalje se je s prihodom računalnikov spremenil poudarek od iskanja matematične formule do iskanja učinkovitega algoritma.

V lanskem letu je profesor Agrawal iz indijskega inštituta za tehnologijo (IIT) v Kampurju prišel na dan s čudovito novico. S študentoma Kayalom in Saxeno so našli determinističen algoritem polinomske zahtevnosti za problem praštevilo, katerega pravilnost so znali tudi dokazati. Algoritem sedaj po kraticah priimkov avtorjev imenujemo **AKS algoritem**. Agrawal je za ta rezultat prejel prestižno nagrado 'Clay Research Award'.

Ker je bilo na rešitev tega preprostega problema potrebno čakati toliko desetletij, so znanstveniki predvidevali, da bo rezultat (če ga bodo našli) dolg in kompliciran. Zgodilo pa se je prav nasprotno, in kot je v nekem intervijuju[1] dejal Lenstra, dober poznavalec teorije števil, je dokaz pravilnosti 12 vrstic dolgega AKS algoritma *'kombinacija perfektno elementarnih prijemov algebre in analize'*. Le za dokaz časovne zahtevnosti se je bilo potrebno sklicati na dva netrivialna rezultata o gostoti praštevil. Pomerance, ki je bil do leta 2002 lastnik najboljšega rezultata, je dejal: *"Ko pride do presenečenj na tem področju, mora biti kriptografom tesno pri srcu."*

Za problem faktorizacije se predvideva, da je težji od problema praštevil. Edini dokaz za to pa je, da ga še nihče ni rešil. Agrawal upa, da bodo posamezne tehnike njihovega algoritma uporabne tudi pri problemu faktorizacije, vendar pa je dejal, da bo najprej raje poskušal reševati druge probleme iz teorije števil, za katere obstajajo le verjetnostni algoritmi.

Ker je AKS algoritem s svojo časovno zahtevnostjo še prepočasen, se bodo v kriptografiji še nekaj časa uporabljali do sedaj znani postopki, opisani v razdelkih 6.1 in 6.6. V razdelkih 6.2 – 6.4 bomo po Agrawal et al. [1] povzeli prvi determinističen algoritem `PrastQ` za rešitev problema `PRAŠTEVILO` ter dokaz njegove pravilnosti in polinomske časovne zahtevnosti. V razdelku

[1] SIAM News, Volume 35, Number 7, September 2002
`http://www.siam.org/siamnews/09-02/primality.htm`

6.5 pa sta predstavljeni hipotezi, od katerih bi veljavnost prve izboljšala oceno časovne zahtevnosti AKS algoritma, veljavnost druge pa bi potrdila obstoj determinističnega algoritma za problem PRAŠTEVILO kubične časovne zahtevnosti glede na velikost zapisa vhodnih podatkov.

6.1 Pregled poznanih algoritmov

Polinomske algoritme za problem PRAŠTEVILO delimo na deterministične ter verjetnostne. Šele leta 2002 smo z odkritjem AKS algoritma spoznali prvi determinističen algoritem polinomske časovne zahtevnosti, katerega pravilnost znamo dokazati. Omenimo najpomembnejše prelomnice pri iskanju čimboljšega algoritma za problem PRAŠTEVILO. Najstarejši poznan algoritem je Eratostenovo rešeto iz leta 240 p.n.š., toda njegova časovna zahtevnost je $O(n)$, torej eksponentna glede na dolžino zapisa obravnavanega števila n. Večji korak naprej je napravil Fermat v 17. stoletju, ko je dokazal, da za vsako naravno število a in za vsako praštevilo p, ki ne deli a, velja $a^{p-1} \equiv 1 \pmod p$. Izrek imenujemo **Fermatov mali izrek** in je uporabljen pri dokazovanju pravilnosti mnogih algoritmov za problem PRAŠTEVILO. Leta 1976 je Miller predstavil determinističen algoritem polinomske časovne zahtevnosti, katerega pravilnost pa ni v celoti dokazana, saj temelji na predpostavki, da velja razširjena Riemannova hipoteza[2]. Nato sta Solovay in Strassen leta 1977 predstavila verjetnosten algoritem časovne zahtevnosti $O(\log^3 n)$. Rabin je leta 1980 modificiral Millerjev test in sestavil verjetnostni algoritem, katerega pravilnost delovanja je dokazal brez uporabe Riemannove hipoteze. Ta algoritem je še danes med najučinkovitejšimi. Največji preskok na poti do determinističnega algoritma polinomske časovne zahtevnosti pa so napravili Adleman, Pomerance in Rumely leta 1983 z determinističnim algoritmom časovne zahtevnosti $O(\log n^{O(\log\log\log n)})$. Goldwasser in Kilian sta leta 1986 napisala verjetnosten algoritem z uporabo eliptičnih krivulj, katerega pričakovana časovna zahtevnost je polinomska za skoraj vse možne vhodne podatke, oz. za vse pod neko hipotezo. Leta 2002 pa so Agrawal, Kayal in Saxena napisali determinističen algoritem časovne zahtevnosti največ $O\left(\log^{12} n\right)$ - v praksi $O\left(\log^6 n\right)$. Predstavili so tudi determinističen algoritem časovne zahtevnosti $O\left(\log^3 n\right)$, vendar pa njegove pravilnosti še niso uspeli v celoti dokazati. Lenstra pa je že izboljšal dokaz časovne zahtevnosti AKS algoritma na $O\left(\log^8 n\right)$.

Opise Eratostenovega rešeta, Solovay-Strassenovega testa ter Miller-Rabinovega najdemo v učbeniku za kriptografijo Stinson [87] ali pa v Bressoud [15], idejo verjetnostnih algoritmov pa bomo spoznali v razdelku 6.6.

Oznaka P predstavlja množico problemov, za katere obstaja determinističen algoritem $\mathcal{A}$ polinomske časovne zahtevnosti glede na dolžino zapisa vhodnih podatkov. Torej obstaja polinom $p(x) \in \mathbb{Z}[x]$, da je $T_{\mathcal{A}}(x) \leq p(n)$ za vsak vhodni podatek x dolžine n, kjer je $T_{\mathcal{A}}(x)$ oznaka za število korakov algoritma. Deterministične algoritme lahko bolj natančno opredelimo kot Turingove stroje, o katerih je za Obzornik pisal že Pisanski [77], časovno zahtevnost pa kot število izpolnjenih ukazov stroja. Več o algoritmih in časovnih zahtevnostih si lahko preberemo tudi v Horowitz in Sahni [41].

Vpeljimo še nekaj oznak, ki jih bomo pogosto uporabljali v tem poglavju. S $\mathbb{P}$ označimo množico praštevil, s $P(n)$ največji praštevilski delitelj števila n, $\mathrm{GF}(p^d)$ pa naj bo končni obseg moči p^d. Pri izbranem polinomu $t(n)$ naj bo $\widetilde{O}(t(n))$ oznaka za $O(t(n)\ \mathtt{poli}(\log t(n)))$, kjer je poli nek polinom. Oznako srečamo na primer, ko s

[2] http://www.claymath.org/prizeproblems/riemann.htm

pomočjo rekurzivnega izračuna izboljšamo časovno zahtevnost določenega algoritma za en red, s tem da pridobimo logaritemski faktor $\log n$. Ker je $\log n$ asimptotično zanemarljiv v primerjavi z n, ponavadi namesto $\widetilde{O}$ pišemo kar O. V tem poglavju je log oznaka za logaritem z osnovo 2, $o_r(n)$ pa red elementa n v grupi $\mathbb{Z}_r^*$.

Trditev "PRAŠTEVILO je v P" bomo dokazali v treh razdelkih. V 6.2 bomo predstavili determinističen algoritem za problem PRAŠTEVILO, povzet po Agrawal et al. [1], v 6.3 bomo dokazali, da je polinomske časovne zahtevnosti, v 6.4 pa še pravilnost delovanja.

6.2 Glavna ideja in algoritem

Poglejmo si karakterizacijo praštevil, ki je ključnega pomena pri konstrukciji algoritma.

Trditev 74. *Naj bosta a in n tuji si naravni števili. Potem je n praštevilo natanko tedaj, ko je*

$$(x-a)^n \equiv (x^n - a) \pmod n \qquad \textit{(kot polinom)}. \tag{6.1}$$

Dokaz. Naj bo $w(x) = (x-a)^n - (x^n - a)$, kar je z upoštevanjem binomske formule enako $w(x) = \sum_{0<i<n} (-1)^i \binom{n}{i} a^{n-i} x^i + (-1)^n a^n + a$. Trditev dokažemo v vsako smer posebej.

($\Rightarrow$) Naj bo $n \in \mathbb{P}$. Potem je $\binom{n}{i} \equiv 0 \pmod n$ za vsak i, $0 < i < n$. Če upoštevamo še Fermatov izrek, dobimo $w(x) \equiv 0 \pmod n$ za vsak $n \in \mathbb{P}$.

($\Leftarrow$) Naj bo n sedaj sestavljeno število. Potem obstaja $q \in \mathbb{P}$, $1 < q < n$, in $k \in \mathbb{N}$, tako da $q^k | n$ in $q^{k+1} \nmid n$. Ker je

$$\binom{n}{q} = \frac{n(n-1)\cdots(n-q+1)}{q(q-1)\cdots 1},$$

velja $q^{k-1} | \binom{n}{q}$ ter $q^k \nmid \binom{n}{q}$ in zato $n \nmid \binom{n}{q}$. Ker je n tuj a, je $\mathrm{D}(n, a^{n-q}) = 1$ in zato koeficient $(-1)^q \binom{n}{q} a^{n-q}$ polinoma $w(x)$ pri potenci x^q ni deljiv z n. ∎

Če preverimo kongruenco (6.1) za nek $a < n$, pri katerem je $\mathrm{D}(a,n) = 1$, npr. $a = 1$, potem se lahko na podlagi trditve 74 odločimo, ali je n praštevilo. Za izračun $(x-a)^n$ uporabimo algoritem 'kvadriraj in množi', vendar pa je časovna zahtevnost takega izračuna eksponentna glede na dolžino zapisa števila n, saj je kvadriranje polinoma $(x-a)^{\lfloor n/2 \rfloor}$, ki je stopnje $\lfloor n/2 \rfloor$, časovne zahtevnosti vsaj $O(n)$. Algoritem izboljšamo tako, da izberemo "primeren" $r \in \mathbb{N}$ in za $a \in \{1, \ldots, \lfloor 2\sqrt{r} \log n \rfloor\}$ preverjamo

$$(x-a)^n \equiv (x^n - a) \pmod{x^r - 1, n}. \tag{6.2}$$

To so kongruence v faktorskem kolobarju $\mathbb{Z}[x]/(n, x^r - 1)$, to je faktorskem kolobarju kolobarja $\mathbb{Z}[x]$ po idealu $(n, x^r - 1)$. Ker je kolobar $\mathbb{Z}[x]/(n, x^r - 1)$ izomorfen kolobarju $\mathbb{Z}_n[x]/(x^r - 1)$, lahko veljavnost kongruence (6.2) preverimo tako, da najprej izračunamo ostanek polinomov na posamezni strani kongruence pri deljenju s polinomom $x^r - 1$, nato pa koeficiente dobljenih polinomov reduciramo še po modulu števila n. Če je n praštevilo, je kongruenca (6.2) posledica trditve 74. Težava pa je v tem, da kongruenca (6.2) lahko velja tudi pri nekaterih sestavljenih številih. Za eksponent r bomo dejali, da je "primeren", če bo

1. $r \in \mathbb{P}$,

Procedure : `PrastQ`

Input: `celo število` $n > 1$
Output: `SESTAVLJENO / PRAŠTEVILO`

if n `je oblike` a^b, $b > 1$ **output** `SESTAVLJENO;`
$r = 2$;
while $r < n$ {
  if $\mathrm{D}(n, r) \neq 1$ **output** `SESTAVLJENO;`
  if r `je praštevilo`
    `naj bo` q `največji praštevilski delitelj od` $r - 1$;
    if $q \geq 4\sqrt{r} \log n$ **and** $n^{(r-1)/q} \not\equiv 1 \pmod r$
      break; `(* while *)`
  $r \leftarrow r + 1$;
}
for $a = 1$ **to** $2\sqrt{r} \log n$
  if $(x - a)^n \not\equiv (x^n - a) \pmod{x^r - 1, n}$ **output** `SESTAVLJENO;`
output `PRAŠTEVILO;`

Slika 6.1: AKS algoritem za testiranje praševilskosti

2. $r = O(\log^6 n)$ in

3. $r - 1$ vsebuje praštevilski delitelj q velikosti vsaj $r^{\frac{1}{2}+\delta}$, $\delta > 0$, ki deli red števila n v grupi $\mathbb{Z}_r^*$, tj. $q | o_r(n)$.

Tako dobljen algoritem za testiranje praštevilskosti bomo poimenovali `PrastQ` in je predstavljen na sliki 6.1. Pokazali bomo, da če pri takem r velja kongruenca (6.2) za vsako naravno število a, ki ni večje od $2\sqrt{n} \log n$, potem je n praštevilo. V algoritmu pogoj za velikost q preverimo s $q \geq 4\sqrt{r} \log n$, iz $n^{(r-1)/q} \not\equiv 1 \pmod r$ pa sledi $q | o_r(n)$, saj je $n^{r-1} \equiv 1 \pmod r$.

6.3 Časovna zahtevnost

Program `PrastQ` si lahko ogledamo tudi na internetu[3]. Napisan je v programskem paketu za simbolno računanje `Mathematica` in uporablja funkcije `AnabQ`, `LargePrime`, `PoliPowerMod` in `CongruenceQ`.

`AnabQ[n]` preveri, ali je število n oblike a^b. Naj bo m tak, da je $n = m^m$, torej je $m = O(\log n)$. Algoritem preizkusi vse celoštevilske $a \in [2, m]$ in preveri, če $b = \log_a n \in \mathbb{Z}$, nato pa še vse celoštevilske $b \in [2, m]$ in preveri, če $a = \sqrt[b]{n} \in \mathbb{Z}$. Skupaj je to $2m$ korakov in časovna zahtevnost tega algoritma je $O(\log^3 n)$ bitnih operacij.

`LargePrime[n]` s preizkusom vseh možnih deliteljev števila n, ki so manjši od $\sqrt{n}$, ugotovi največji praštevilski delitelj števila n in je časovne zahtevnosti $O(\sqrt{n})$

`PoliPowerMod[u,r,v,n]` izračuna $u(x)^r \pmod{v(x), n}$ s pomočjo algoritma 'kvadriraj in množi' (glej Stinson [87, str. 127]), kjer na vsakem koraku rezultat tudi reduciramo po modulu $x^r - 1$ in po modulu n. To storimo tudi čisto na začetku. Potrebnih je torej $O(\log r)$ korakov. Če uporabimo Fourierovo transformacijo za množenje polinomov (glej

[3] `http://www.fmf.uni-lj.si/~novakta/research/PrastQ.htm`

Bini in Pan [11, str. 8–13]) in še hitro metodo za deljenje polinomov (glej Bini in Pan [11, str. 18–23]), je časovna zahtevnost tega algoritma $\widetilde{O}((\log n)(\deg u + \log r \deg v))$.

`CongruenceQ[a,n,r]` nam s pomočjo `PoliPowerMod` preveri, ali velja

$$(x-a)^n \equiv (x^n - a) \pmod{x^r - 1, n}\,.$$

Za to potrebuje $\widetilde{O}(r \log^2 n)$ bitnih operacij.

V nadaljevanju bomo najprej navedli dve lemi, katerih dokaza sta preobsežna in ju bomo izpustili. Lemi bomo potrebovali v dokazu trditve 77, s katero bomo pokazali, da algoritem `PrastQ[n]` najde primeren r v največ $O(\log^6 n)$ korakih (algoritem se lahko ustavi še predno najde primeren r). Časovna zahtevnost enega koraka v `while` zanki je $\widetilde{O}(\log^3 n)$, v `for` zanki pa $\widetilde{O}(\log^8 n)$. Skupaj algoritem `PrastQ[n]` potroši največ $\widetilde{O}(\log^{12} n)$ bitnih operacij, kar pomeni, da je `PRAŠTEVILO` v P.

Lema 75 (Fouvry [26], Baker in Harman [5]). *Obstajata konstanti $c > 0$ in n_0, tako da za vsako naravno število $n \geq n_0$ velja*

$$|\{p \,|\, p \text{ je praštevilo, } p \leq n \text{ in } P(p-1) > n^{2/3}\}| \geq c\frac{n}{\log n}\,,$$

kjer je $P(a)$ oznaka za največji praštevilski delitelj naravnega števila a.

Lema 76 (Apostol [3]). *Naj bo $\pi(n)$ število praštevil, ki niso večja od n. Potem za vsak $n \in \mathbb{N}$ velja*

$$\frac{n}{6\log n} \leq \pi(n) \leq \frac{8n}{\log n}\,.$$

Trditev 77. *Obstajajo pozitivne konstante c_1, c_2 in n_1 pri katerih za vsak $n \geq n_1$ obstaja praštevilo r na intervalu $[c_1 \log^6 n, c_2 \log^6 n]$ za katerega velja, da ima $r-1$ praštevilski delitelj q, ki zadošča pogojema $q \geq 4\sqrt{r}\log n$ in $n^{(r-1)/q} \not\equiv 1 \pmod{r}$.*

Dokaz. Naj bosta c in n_0 konstanti iz leme 75. Izberimo pozitivni konstanti c_1 in c_2 tako, da velja

$$c_2 > c_1 > 0, \qquad \frac{c\,c_2}{7} - \frac{8c_1}{6} > 0 \qquad \text{ter} \qquad c_1^4 > 4^6 c_2^3\,. \tag{6.3}$$

Ni se težko prepričati, da je $\{(c_1, c_2) \,|\, c_2 > c_1 > 0, c\ c_2/7 - 8c_1/6 > 0 \text{ in } c_1^4 > 4^6 c_2^3\}$ neprazno območje v $\mathbb{R}^2$.

Praštevila p, za katera velja $P(p-1) > p^{2/3}$ poimenujmo 'posebna praštevila'. S pomočjo leme 75 in leme 76 ocenimo, koliko je posebnih praštevil na intervalu $[c_1 \log^6 n, c_2 \log^6 n]$, kjer je $n \geq n_0$ in $\log n \geq c_2$:

$$\begin{aligned}
&|\{p \in \mathbb{P} \mid p \in [c_1 \log^6 n, c_2 \log^6 n], P(p-1) > p^{2/3}\}| \geq \\
&\geq \quad |\{p \in \mathbb{P} \mid p \leq c_2 \log^6 n, P(p-1) > p^{2/3}\}| - |\{p \in \mathbb{P} \mid p \leq c_1 \log^6 n\}| \geq \\
&\geq \quad c\frac{c_2 \log^6 n}{\log(c_2 \log^6 n)} - \frac{8(c_1 \log^6 n)}{\log(c_1 \log^6 n)} \\
&\geq \quad \frac{\log^6 n}{\log\log n}\left(\frac{c\,c_2}{7} - \frac{8c_1}{6}\right) = c_3 \frac{\log^6 n}{\log\log n}\,,
\end{aligned}$$

kjer smo definirali $c_3 = c\,c_2/7 - 8c_1/6$. Po definiciji (6.3) za c_1 in c_2 je $c_3 > 0$.

Naj bo $x = c_2 \log^6 n$ in

$$N = (n-1)(n^2-1)(n^3-1)\cdots(n^{\lfloor x^{1/3}\rfloor}-1)\,.$$

Pokažimo, da obstaja posebno praštevilo na intervalu $[c_1 \log^6 n, c_2 \log^6 n]$, ki ne deli število N. Najprej ocenimo število praštevilskih deliteljev števila N:

$$\begin{aligned}
&|\{p \in \mathbb{P} \mid p \text{ deli } N\}| \leq \\
&\quad \leq \ \log N = \log(n-1) + \log(n^2-1) + \cdots + \log(n^{\lfloor x^{1/3}\rfloor}-1) \leq \\
&\quad \leq \ \log(n) + 2\log n + \cdots + \lfloor x^{1/3}\rfloor \log n = \frac{\lfloor x^{1/3}\rfloor(\lfloor x^{1/3}\rfloor+1)}{2}\log n\,.
\end{aligned}$$

Za dovolj velik n in s tem tudi x je to manj kot $x^{2/3}\log n$, oziroma $c_2^{2/3}\log^5 n$, kar je za dovolj velik n manj kot $c_3 \log^6 n/\log\log n$. Torej obstaja posebno praštevilo na intervalu $[c_1 \log^6 n, c_2 \log^6 n]$, ki ne deli N. Izberimo si enega in ga označimo z r. Pokažimo, da praštevilo r ustreza pogojem iz trditve. Ocenimo velikost števila $q = P(r-1)$:

$$q \geq r^{2/3} \geq c_1^{2/3}\log^4 n = \frac{c_1^{2/3}}{c_2^{1/2}}(c_2 \log^6 n)^{1/2}\log n \geq 4\sqrt{r}\log n\,.$$

Če hočemo pokazati $n^{(r-1)/q} \not\equiv 1 \pmod r$, je dovolj, da pokažemo $q = o_r(n)$ in $(r-1)/q < o_r(n)$. Ker r ne deli N, potem r ne deli $(n^i - 1)$ za noben i, $1 \leq i \leq \lfloor x^{1/3}\rfloor$, kar pomeni, da $n^i \not\equiv 1 \pmod r$ za noben i, $1 \leq i \leq \lfloor x^{1/3}\rfloor$, torej je $o_r(n) > x^{1/3}$.

Ker je $(r-1)/q \leq r/r^{2/3} = r^{1/3} \leq (c_2 \log^6 n)^{1/3} = x^{1/3} < o_r(n)$ in ker red elementa deli moč grupe, torej tudi $o_r(n)$ deli $(r-1)$, je $q = o_r(n)$. ■

6.4 Dokaz pravilnosti algoritma

Če je n praštevilo, se algoritem ne more ustaviti v vrsticah 1, 4 ali 12, torej nam vrne `PRAŠTEVILO`. Za dokaz, da nam za sestavljeno število n algoritem vrne `SESTAVLJENO`, pa bomo potrebovali nekaj splošnih dejstev iz algebre (o grupah in končnih obsegih), ki jih bomo dokazali v naslednjih dveh lemah. Nato bomo v trditvi 81 definirali grupo G in ocenili njeno velikost. Definirali bomo tudi množico $I_{g(x)}$ in v lemah 82 in 83 pokazali njene lastnosti, nato pa bomo v izreku 84 s protislovjem pokazali, da če je n sestavljeno število, potem algoritem vrne `SESTAVLJENO`.

Lema 78. *Naj bo G grupa z enoto e in $a \in G$. Potem je*

$$a^n = e \Longleftrightarrow \mathrm{red}(a)|n\,.$$

■

Lema 79. *Naj bosta p in r različni praštevili. Potem velja:*

1. *Multiplikativna grupa obsega $\mathrm{GF}(p^d)$, $d \in \mathbb{N}$, je ciklična.*

2. *Če je $f(x)$ polinom s celoštevilskimi koeficienti, potem velja*

$$f(x)^p \equiv f(x^p) \pmod p\,.$$

3. *Če je $h(x)$ poljuben faktor polinoma $x^r - 1$ in $m \equiv m_r \pmod r$, potem velja*
$$x^m \equiv x^{m_r} \pmod{h(x)}.$$

4. *Polinom $(x^r-1)(x-1)$ se faktorizira v nerazcepne polinome stopnje $o_r(p)$.*

Dokaz.
(1) Dokaz najdemo v vsakem standardnem učbeniku iz algebre o končnih obsegih, npr. Vidav [92].

(2) Naj bo $f(x) = a_0 + a_1x + \cdots + a_dx^d$. Koeficient pri potenci x^i v $f(x)^p$ je enak
$$\sum \left\{ \frac{p!}{i_0!i_1!\cdots i_d!} a_0^{i_0}a_1^{i_1}\cdots a_d^{i_d} \,\middle|\, i_0+i_1+\cdots+i_d = p \text{ in } i_1+2i_2+\cdots+di_d = i \right\}.$$
Ta koeficient je deljiv s p natanko tedaj, ko so v vsoti vsi indeksi i_k manjši od p. Upoštevamo še Fermatov izrek in dobimo $f(x)^p \equiv \sum_{k=0}^d a_kx^{kp} \equiv f(x^p) \pmod p$.

(3) Naj bo $m = kr + m_r$, kjer je $k \in \mathbb{Z}$. Če kongruenco $x^r \equiv 1 \pmod{x^r-1}$ na obeh straneh potenciramo na k in nato še obe strani pomnožimo z x^{m_r}, dobimo $x^m \equiv x^{m_r} \pmod{x^r-1}$. Ker $h(x)$ deli x^r-1, je tudi $x^m \equiv x^{m_r} \pmod{h(x)}$.

(4) Naj bo $s = o_r(p)$, $Q(x) = \frac{x^r-1}{x-1}$, $h(x)$ nek nerazcepen faktor od $Q(x)$ in $k = \deg h(x)$. Pokažimo, da $k|s$ in $s \le k$, torej da velja $k = s$.
Naj bo $\mathbb{F} = \mathrm{GF}(p^k)$ definiran s polinomom $h(x)$ in naj bo $g(x)$ generator grupe $\mathbb{F}^*$. Po 2. točki leme je
$$g(x)^p \equiv g(x^p) \pmod p,$$
oziroma, če to točko uporabimo s-krat, dobimo
$$g(x)^{p^s} \equiv g(x^{p^s}) \pmod p.$$
Ker je $p^s \equiv 1 \pmod r$, je po 3. točki leme
$$g(x^{p^s}) \equiv g(x) \pmod{h(x)}.$$
Predzadnja kongruenca velja tudi po modulu $h(x)$, saj če sta dva elementa enaka v kolobarju $\mathbb{Z}_p[x]$, potem predstavljata isti element v faktorskem kolobarju $\mathbb{Z}_p[x]/h(x)$, ki pa je izomorfen $\mathbb{Z}[x]/(h(x),p)$. Podobno velja za zadnjo kongruenco in zato je
$$g(x)^{p^s} \equiv g(x) \pmod{h(x),p}.$$
Torej polinom $h(x)$ deli $g(x)(g(x)^{p^s-1}-1)$ v obsegu $\mathbb{F}$. Ker je $h(x)$ nerazcepen v $\mathbb{F}$, deli vsaj enega od faktorjev. Če bi $h(x)$ delil $g(x)$, bi bil $g(x)$ ničeln element v $\mathbb{F}$, torej je
$$g(x)^{p^s-1} \equiv 1 \pmod{h(x)}.$$
Ker je $g(x)$ generator grupe $\mathbb{F}^*$, je njegov red enak moči te grupe, tj. $\mathrm{red}(g(x)) = p^k-1$. Po lemi 78 velja $p^k-1 \,|\, p^s-1$, to pa je res natanko tedaj, ko $k|s$.
Pokažimo še, da velja $s \le k$. Ker $h(x)$ deli x^r-1 v $\mathbb{Z}_p[x]$, je
$$x^r \equiv 1 \pmod{h(x),p}.$$
Iz leme 78 sledi, da $\mathrm{red}(x) \,|\, r$ in ker je $r \in \mathbb{P}$ in ker x ni enica v $\mathbb{F}$, je $\mathrm{red}(x) = r$. Ker red vedno deli moč grupe, velja $r|(p^k-1)$, torej $p^k \equiv 1 \pmod r$. Po definiciji s je $s \le k$. ■

V nadaljevanju naj bo n sestavljeno število in predpostavimo, da se algoritem ne ustavi v vrsticah 1 ali 4. Naj bo r praštevilo, pri katerem se ustavi zanka `while`, in $q = P(r-1)$.

Lema 80. *Pri pravkar definiranih pogojih obstaja tak praštevilski faktor p števila n, da q deli $o_r(p)$.*

Dokaz. Naj bo $n = \prod_i p_i^{k_i}$. Potem $o_r(n)$ deli $\mathrm{v}_i(o_r(p_i))$, kjer je v oznaka za najmanjši skupni večkratnik. Npr. če bi bil $n = \prod_i p_i$, potem bi bil $o_r(n) = \mathrm{v}_i(o_r(p_i))$. Ker q deli $o_r(n)$ (eden od pogojev za primernost števila r), potem q deli tudi $\mathrm{v}_i(o_r(p_i))$ in ker je q praštevilo, obstaja i, tak da $q|o_r(p_i)$. ■

Ker iz $(x-a)^n \equiv (x^n - a) \pmod{x^r - 1, n}$ sledi

$$(x-a)^n \equiv (x^n - a) \pmod{h(x), p},$$

lahko na kongruence iz algoritma gledamo kot na relacije v obsegu $\mathbb{F} = \mathbb{Z}_p[x]/h(x)$, kjer je $h(x)$ nerazcepen faktor polinoma $x^r - 1$ stopnje $d = o_r(p)$, glej 4. točko leme 79.

Trditev 81. [1, Lemma 4.4] *Naj bo $\ell = 2\sqrt{r}\log n$. V obsegu $\mathbb{Z}_p[x]/h(x)$ je multiplikativna podgrupa*

$$G = \left\{ \prod_{1 \le a \le \ell} (x-a)^{\alpha_a} | \alpha_a \ge 0 \right\}$$

ciklična in moči $> (\frac{d}{\ell})^\ell \ge n^{2\sqrt{r}}$.

Dokaz. Naj bo $\mathbb{F} = \mathbb{Z}_p[x]/h(x)$. Ker je $\mathbb{F}^*$ ciklična, je tudi $G \le \mathbb{F}^*$ ciklična. Ocenimo še velikost grupe G s pomočjo množice

$$S = \left\{ \prod_{1 \le a \le \ell} (x-a)^{\alpha_a} \,\middle|\, \alpha_a \ge 0, \sum_{1 \le a \le \ell} \alpha_a \le d - 1 \right\}.$$

Ker se algoritem ne ustavi v vrsticah 1 ali 4, nam najde r, za katerega velja $r > q \ge 4\sqrt{r}\log n > \ell$ in $\mathrm{D}(n,b) = 1$ za vsak $b \le r$. Če bi bila katera $a_1, a_2 \in \mathbb{N}$, $1 \le a_1 < a_2 \le \ell$, kongruentna po modulu p, potem bi p delil $a_2 - a_1$, kar pa je v protislovju z $\mathrm{D}(n, a_2 - a_1) = 1$, saj je $0 < a_2 - a_1 < a_2 \le \ell < r$.

Ker so vsi polinomi v množici S stopnje $\le d - 1 < \deg h(x)$ in ker so vsi a-ji različni po modulu p, nam elementi v S predstavljajo paroma različne elemente v obsegu $\mathbb{F}$. Vsak polinom v množici S je produkt $d-1$ faktorjev, pri čemer imamo $\ell + 1$ možnih faktorjev na izbiro, to so 1, $x-1$, …, $x - \ell$. Iz formule za kombinacije s ponavljanjem dobimo

$$\begin{aligned} |G| \ &\ge \ |S| = \binom{(d-1)+(\ell+1)-1}{d-1} = \binom{d+\ell-1}{\ell} = \\ &= \frac{(d+\ell-1)(d+\ell-2)\cdots d}{\ell!} > \left(\frac{d}{\ell}\right)^\ell . \end{aligned}$$

Ker q deli $o_r(p)$, je $d = o_r(p) \ge q \ge 4\sqrt{r}\log n = 2\ell$ in zato

$$|G| \ge \left(\frac{2\ell}{\ell}\right)^\ell = 2^\ell = 2^{2\sqrt{r}\log n} = n^{2\sqrt{r}}.$$

■

Naj bo polinom $g(x)$ generator grupe G iz trditve 81. Označimo z o_g red elementa $g(x)$ v obsegu $\mathbb{Z}_p[x]/h(x)$. Ker je število o_g enako moči grupe G, velja $o_g \ge n^{2\sqrt{r}}$. Definirajmo

$$I_{g(x)} = \{m \in \mathbb{N} \mid g(x)^m \equiv g(x^m) \pmod{x^r - 1, p}\}$$

in pokažimo nekaj njenih osnovnih lastnosti.

Lema 82. *Množica $I_{g(x)}$ je zaprta za množenje.*

Dokaz. Naj bosta $m_1, m_2 \in I_{g(x)}$. To pomeni

$$g(x)^{m_1} \equiv g(x^{m_1}) \pmod{x^r - 1, p} \qquad \text{in} \tag{6.4}$$
$$g(x)^{m_2} \equiv g(x^{m_2}) \pmod{x^r - 1, p}\,. \tag{6.5}$$

Če v kongruenci (6.5) pišemo x^{m_1} namesto x in upoštevamo, da $x^r - 1$ deli $x^{m_1 r} - 1$, dobimo $g(x^{m_1})^{m_2} \equiv g(x^{m_1 m_2}) \pmod{x^r - 1, p}$, kar nam da skupaj s kongruenco (6.4)

$$g(x)^{m_1 m_2} \equiv g(x^{m_1 m_2}) \pmod{x^r - 1, p}\,,$$

to pa pomeni, da $m_1 m_2 \in I_{g(x)}$. ■

Lema 83. *Če sta $m_1, m_2 \in I_{g(x)}$, potem iz $m_1 \equiv m_2 \pmod{r}$ sledi $m_1 \equiv m_2 \pmod{o_g}$.*

Dokaz. Ker je $m_2 \in I_{g(x)}$ in ker $h(x)$ deli $x^r - 1$, je $g(x)^{m_2} \equiv g(x^{m_2}) \pmod{h(x), p}$. Ker je $m_1 \equiv m_2 \pmod{r}$, je $m_2 = m_1 + kr$ za $k \in \mathbb{Z}$ in če upoštevamo še 3. točko leme 79, dobimo $g(x)^{m_1} g(x)^{kr} \equiv g(x^{m_1}) \pmod{h(x), p}$. Ker je $m_1 \in I_{g(x)}$, dobimo, da $h(x)$ deli $g(x)^{m_1}(g(x)^{kr} - 1)$ v $\mathbb{Z}_p[x]$. Ker je $h(x)$ nerazcepen nad $\mathbb{Z}_p$, velja $h(x)$ deli $g(x)$ ali $h(x)$ deli $g(x)^{kr} - 1$. Če bi bil $g(x)$ deljiv s $h(x)$, bi bil ničeln element v $\mathbb{F}$. Torej je $g(x)^{kr} \equiv 1 \pmod{h(x), p}$ in po lemi 78 o_g deli kr in s tem $m_2 - m_1$. ■

Izrek 84. [1, Lemma 4.7] *Če je n sestavljeno število, algoritem vrne* `SESTAVLJENO`.

Dokaz. Predpostavimo, da je n sestavljeno število in da algoritem vrne `PRAŠTEVILO`. Pokažimo najprej, da je $\{1, p, n\} \subseteq I_{g(x)}$. Po definiciji $I_{g(x)}$ je očitno, da $1 \in I_{g(x)}$, iz druge točke leme 79 pa sledi $p \in I_{g(x)}$. Če upoštevamo, da je $g(x) \in G$, torej da je oblike

$$g(x) = \prod_{1 \le a \le l} (x - a)^{\gamma_a}$$

in še, da so pravilne vse kongruence v zanki `for`, saj algoritem vrne `PRAŠTEVILO`, dobimo

$$g(x)^n \equiv g(x^n) \pmod{x^r - 1, p}\,.$$

Sedaj definirajmo

$$E = \{n^i p^j \mid 0 \le (i, j) \le \lfloor \sqrt{r} \rfloor\}\,.$$

Ker je $\{1, p, n\} \subseteq I_{g(x)}$ in ker je $I_{g(x)}$ zaprta za množenje, je $E \subseteq I_{g(x)}$. Ker je $|E| = (1 + \sqrt{r})^2 > r$, obstajata po Dirichletovem principu taka $n^{i_1} p^{j_1}$ in $n^{i_2} p^{j_2}$ iz E, $(i_1, j_1) \neq (i_2, j_2)$, da velja $n^{i_1} p^{j_1} \equiv n^{i_2} p^{j_2} \pmod{r}$. Z uporabo zadnje leme dobimo $n^{i_1} p^{j_1} \equiv n^{i_2} p^{j_2} \pmod{o_g}$. Ker je p delitelj števila n, $o_g > n^{2\sqrt{r}}$ in ker sta $n^{i_1} p^{j_1}$, $n^{i_2} p^{j_2} \le n^{2\sqrt{r}}$, je $n^{i_1} p^{j_1} = n^{i_2} p^{j_2}$. Torej je $n^{i_1 - i_2} = p^{j_2 - j_1}$ in algoritem bi se moral ustaviti v vrstici 1. To pa je v protislovju s predpostavko, da algoritem vrne `PRAŠTEVILO`. ■

6.5 Pospešitve in odprti problemi

Predstavili bomo dve hipotezi s področja, ki obravnava praštevila. Če velja hipoteza 85, potem je časovna zahtevnost algoritma `PrastQ` največ $\widetilde{O}(\log^6 n)$, s pomočjo hipoteze 87 pa dobimo determinističen algoritem časovne zahtevnosti $\widetilde{O}(\log^3 n)$.

Če sta r in $\frac{r-1}{2}$ praštevili, imenujemo $\frac{r-1}{2}$ **Sophie Germain** praštevilo in r **ko-Sophie Germain** praštevilo.

Hipoteza 85 (Hardy in Littlewood [40]). *Število ko-Sophie Germain praštevil manjših od n je asimptotično enako $D\,n/\log^2 n$, kjer je $D \in \mathbb{R}$.*

Konstanti D rečemo **konstanta praštevilskih dvojčkov**. Hipotezo sta avtorja preverila za števila $r \leq 10^{10}$. Če hipoteza velja, potem nam naslednja lema zagotavlja, da je časovna zahtevnost algoritma `PrastQ` največ $\widetilde{O}(\log^6 n)$.

Lema 86. [1, Lemma 5.3] *Če velja hipoteza 85, potem obstajajo take pozitivne konstante n_0 in c_1 in c_2, da obstaja 'primeren' r na intervalu $[c_1 \log^2 n, c_2 \log^2 n]$ za vsak $n \geq n_0$.*

Dokaz. Zaradi preprostejšega izračuna v dokazu vzemimo $c_1 = 100$ in $c_2 = 101$, čeprav bi bila ustrezna že $c_1 \approx c_2 \approx 64$. Definirajmo množico $T = [c_1 \log^2 n, c_2 \log^2 n] \cap \mathbb{P}_{ko-Sop-Ger}$ in s pomočjo hipoteze 85 asimptotično ocenimo njeno moč:

$$|T| \approx \frac{D(c_2 \log^2 n)}{(\log(c_2 \log^2 n)^2} - \frac{D(c_1 \log^2 n)}{(\log(c_1 \log^2 n)^2} \approx \frac{D(c_2 - c_1)\log^2 n}{4(\log\log n)^2}\,.$$

Pokažimo, da za praštevila $r \in T$ velja $P(r) \geq 4\sqrt{r}\log n$:

$$\begin{aligned} P(r) &= \frac{r-1}{2} \geq \frac{c_1 \log^2 n - 1}{2} \geq \frac{c_1 - 1}{2}(\log n)(\log n) \geq \\ &\geq \frac{c_1 - 1}{2}\sqrt{\frac{r}{c_2}}\log n \geq \frac{c_1 - 1}{2\sqrt{c_2}}\sqrt{r}\log n \geq 4\sqrt{r}\log n\,. \end{aligned}$$

Na intervalu $[c_1 \log^2 n, c_2 \log^2 n]$ je asimptotično $O(\log^2 n/(\log\log n)^2)$ ko-Sophie Germain praštevil. Neprimerna so tista, pri katerih je $n^{(r-1)/q} \equiv 1 \pmod r$, torej $n^2 \equiv 1 \pmod r$, kar pomeni $n^2 - 1 = kr$. Neprimernih je torej največ $\log(n^2 - 1) \approx 2\log n$, saj ima $n^2 - 1$ največ $\log(n^2 - 1)$ praštevilskih deliteljev (možnosti za r). Seveda pa je $O(\log n)$ asimptotično manj kot $O(\log^2 n/(\log\log n)^2)$, torej za dovolj velik n obstaja 'primeren' $r \in T$. ■

Sedaj bomo predstavili še hitrejši algoritem, katerega pravilnost še ni dokazana, temelji pa na naslednji hipotezi, ki sta jo Kayal in Saxena [50][4] preverila za pare števil r in n, kjer je $r \leq 100$ in $n \leq 10^{10}$. Da smo za `PrastQ` dobili grupo G moči vsaj $n^{2\sqrt{r}}$, smo morali vzeti $l = 2\sqrt{r}\log n$. Konstanto l lahko ustrezno zmanjšamo in še vedno dobimo grupo zahtevane moči.

Hipoteza 87 (Bhattacharjee in Pandey [9][5]). *Če r ne deli n in če velja*

$$(x-1)^n \equiv (x^n - 1) \pmod{x^r - 1, n}\,,$$

potem je n praštevilo ali pa $n^2 \equiv 1 \pmod r$

Če hipoteza 87 drži, lahko sestavimo hiter algoritem za testiranje praštevilskosti, glej sliko 6.2. Poiščemo r, ki ne deli niti n niti n^2-1. Po lemi 88 obstaja na intervalu $[2, 6\ln n]$. Nato za ta r preverimo kongruenco iz hipoteze 87 in časovna zahtevnost algoritma je tako omejena z $\widetilde{O}(\log^3 n)$.

Lema 88. *Na intervalu $[2, 6\ln n]$ obstaja r, ki ne deli niti n niti $n^2 - 1$.*

Dokaz. Po učbeniku Apostol [3] za analitično teorijo števil povzamemo, da za vsak $x \geq 5$ velja $\prod\{p \mid p \in \mathbb{P} \text{ in } p \leq x\} \geq e^{x/2}$. Če bi vsa praštevila $p \leq 6\ln n$ delila $n^2 - 1$ ali n, torej njun produkt $n^3 - n$, potem bi bil $\prod\{p \mid p \in \mathbb{P} \text{ in } p \leq 6\ln n\} \leq n^3 - n$. Iz zgornje ocene pa vemo, da je $\prod\{p \mid p \in \mathbb{P} \text{ in } p \leq 6\ln n\} \geq e^{6(\ln n)/2} = n^3$. ■

[4] `http://www.cse.iitk.ac.in/research/btp2002/primality.htm`

[5] `http://www.cse.iitk.ac.in/research/btp2001/primality.htm`

```
Procedure : PrastHipoQ
Input:  celo število n > 1
Output: SESTAVLJENO / PRAŠTEVILO

r = 2;
while r < n {
    if n ≡ 0 (mod r) output SESTAVLJENO;
    if n^2 ≢ 1 (mod r) break; (* while *)
    r ← r + 1;
}
if (x − 1)^n ≡ (x^n − 1) (mod x^r − 1, n)
    output PRAŠTEVILO;
else
    output SESTAVLJENO;
```

Slika 6.2: Algoritem za testiranje praštevilskosti, pravilnost temelji na hipotezi 87

```
Procedure : PrastVerjTestQ
Input:  celo število n > 1
Output: SESTAVLJENO / PRAŠTEVILO

naključno izberi celo število a, 1 ≤ a ≤ n − 1;
if (a/n) ≡ a^((n−1)/2) (mod n)
    output PRAŠTEVILO;
else
    output SESTAVLJENO;
```

Slika 6.3: Solovay-Strassenov verjetnostni test za testiranje praštevilskosti

6.6 Iskanje praštevil v praksi

V praksi mnogokrat potrebujemo naključno izbrano veliko praštevilo. Pri implementaciji RSA kriptosistema, glej Stinson [87, str. 124], potrebujemo za 1024 bitni modul naključno izbrano 512 bitno praštevilo. Dobimo ga tako, da izberemo naključno 512 bitno liho število in preverimo, ali je praštevilo. Glede na lemo 76 najdemo praštevilo v $O(\log(n))$ korakih. Če bi v tem programu uporabljali algoritem `PrastQ`, bi ob predpostavki, da velja hipoteza 85, potrošil največ 512^6 bitnih operacij (od vseh korakov pri iskanju praštevila je najdaljši tisti, ko najdemo praštevilo). Ker je $512^6 = 1.8 \cdot 10^{16}$, bi računalnik z $1GHz = 10^9 Hz$ procesorjem potreboval za iskanje enega 512 bitnega praštevila velikostnega reda 10^7 sekund, oziroma 4 mesece, kar pa je nesprejemljivo. Zato se v praksi še vedno uporabljajo verjetnostni algoritmi za problem `PRAŠTEVILO`. Verjetnostni algoritmi potrebujejo funkcijo za generiranje naključnih števil. Posebna zvrst teh algoritmov so Monte-Carlo algoritmi, pri katerih je posebnost to, da vrnejo enega od dveh tipov odgovorov, npr. `TRUE` ali `FALSE`, in eden od odgovorov je vedno pravilen, drugi pa je lahko tudi napačen.

Najbolj znana sta Solovay-Strassenov test ter Miller-Rabinov test, glej Stinson [87, str. 129–138]. Čeprav je Miller-Rabinov test v praksi nekoliko hitrejši, si bomo raje ogledali Solovay-Strassenov, glej sliko 6.3, saj je na videz bolj preprost in bomo z njim lažje prikazali, kako delujejo Monte-Carlo algoritmi.

Časovna zahtevnost testa je $O(\log^2 n)$, definicije in postopek za izračun vrednosti

Jacobijevega simbola $\left(\frac{a}{n}\right)$ pa bralec najde v Stinson [87]. Pri delovanju testa je pomembno to, da če vrne `SESTAVLJENO`, potem je n res sestavljeno število, če pa vrne `PRAŠTEVILO`, pa odgovor ni nujno pravilen. Verjetnost napake je v našem primeru manjša od $1/2$:

$$P(\text{algoritem vrne } \texttt{PRAŠTEVILO} \mid n \text{ ni praštevilo}) < 1/2\,.$$

Če test ponovimo m-krat potem z nekaj truda (glej Stinson [87, str. 135]) za velike n izpeljemo

$$P(n \text{ ni praštevilo} \mid \text{algoritem } m\text{-krat vrne } \texttt{PRAŠTEVILO}) \le \frac{\ln n - 2}{\ln n - 2 + 2^{m+1}}$$

in ne $1-(1/2)^m$, kakor bi intuitivno pričakovali. Če test ponovimo $\log n$-krat in vedno dobimo odgovor `PRAŠTEVILO`, je torej verjetnost, da n ni praštevilo velikostnega reda $\frac{1}{n}$, kar je dovolj zanesljivo za uporabo v praksi. Časovna zahtevnost Solovay-Strassenovega algoritma za preverjanje, ali je n praštevilo, je torej $O(\log^3 n)$. Prav tolikšna je tudi za Miller-Rabinov algoritem. Podobno kot prej lahko dobimo, da na $1GHz$ računalniku stestiramo 512-bitno praštevilo v približno 0,13 sekunde.

V programskem paketu `Mathematica 4` sta za testiranje praštevilskosti vgrajeni funkciji `PrimeQ` in `ProvablePrimeQ`, pri slednji je potrebno predhodno naložiti paket `NumberTheory‘PrimeQ‘`.

Za testiranje praštevilskosti funkcija `PrimeQ` najprej preveri, če ima število n kak majhen praštevilski delitelj, nato uporabi Miller-Rabinov test (krepki psevdopraštevilski test) pri bazi 2 in bazi 3, na koncu pa še Lucasov test[6].

Funkcija `ProvablePrimeQ` uporablja verjetnostni algoritem, ki vedno vrne pravilen rezultat. Pričakovana vrednost za časovno zahtevnost algoritma je polinomska za skoraj vsa naravna števila, oz. za vsa pod neko hipotezo. Algoritem uporablja eliptične krivulje in sta ga prva predstavila Goldwasser in Kilian [29], Atkin pa je s pomočjo uporabe idej množenja kompleksnih števil (del teorije števil, ki združuje kompleksno analizo, Galoisovo teorijo in modularne forme) pokazal, kako bi se ga dalo uporabiti v praksi, glej Atkin et al. [4]. To je leta 1989 storil Morain [65] in to je tudi prvi algoritem, ki je certificiral več kot 1000 mestna praštevila (titanic primes) brez upoštevanja njihovih posebnih lastnosti (npr. praštevilo $2^{2^{11}}-1$). Za certificiranje praštevila $10^{100}+267$ potrebuje Mathematica na 300 MHz računalniku 35 sekund.

Čeprav je bilo problemu certificiranja praštevil v znanosti posvečeno že precej pozornosti, bomo še vedno čakali na bolj eleganten algoritem. Mogoče še dolgo časa, saj so za varnost RSA kriptosistema bolj pomembni rezultati in algoritmi za problem faktorizacije, ki pa je še vedno odprt.

[6] `http://mathpages.com/home/kmath473.htm`

Sklep

Področij uporabe Berlekamp-Masseyevega (BM) algoritma je toliko, da bi težko vse opisali v tem delu, zato smo se osredotočili na Reed-Solomonove kode ter LFSR generatorje. Pri prvih ima BM algoritem ključno vlogo pri hitrem dekodiranju. Da pa lahko ocenimo njegovo vrednost, smo bolj podrobno opisali lastnosti in zmožnosti uporabe kod ter BM algoritmu konkurenčni Guruswami-Sudanov algoritem za dekodiranje v sezname za RS kode. Pri LFSR generatorjih pa lahko BM algoritem uporabimo na več načinov, to je za določanje profilov linearnih zahtevnosti zaporedij, za izpeljavo pričakovane vrednosti in variance za linearno zahtevnost naključnih zaporedij itd. Upamo pa tudi, da bo pomagal pri izpeljavi algoritma za določanje linearne zahtevnosti s k napakami za poljubna končna zaporedja ter pri določitvi formule za pričakovano kvadratično zahtevnost naključnega zaporedja. Poleg naštetih pa je še mnogo drugih načinov uporabe BM algoritma, zato jih bomo na splošno našteli in s tem zaključili magistrsko delo v celoto.

Schurova oblika BM algoritma.
Če hočemo implementirati hiter paralelni algoritem za izračun polinoma lokatorja napake pri dekodiranju BCH kod (reševanje ključne enačbe), lahko uporabimo Schurovo obliko BM algoritma, glej Zarowski [96]. Le ta je analogna Schurovemu algoritmu za Toeplitzove matrike, ki nastopa v matrični obliki zapisa ključne enačbe. Če koda konstruirana tako, da lahko popravi t napak, potem je časovna zahtevnost algoritma enaka $O(t \log t)$.

Večdimenzijska zaporedja.
Massey je leta 1972 predstavil algoritem, ki določi minimalno linearno (nad obsegom $\mathbb{F}$) rekurzivno formulo za zaporedje vektorjev $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$, kjer so $\boldsymbol{a}_i \in \mathbb{F}^k$, $1 \leq i \leq n$. Algoritem je analogija BM algoritma, ki deluje za $k = 1$, njegovo pravilnost pa je dokazal Ding [22] leta 1988. Veliko rezultatov na tem področju je objavil S. Sakata, zelo dober pregled pa je s 40 referencami leta 2001 predstavil Norton [67]. Isti avtor je tudi predstavil n-dimenzijonalno ključno enačbo in njej ustrezni dekodirni algoritem. Novejši algoritem za rešitev problema LFSR sinteze zaporedja vektorjev pa je **Osnovni iterativni algoritem** (**FIA** - Fundamental Iterative Algorithm), ki sta ga leta 1991 predstavila Feng in Tzeng [23].

Posplošitev BM algoritma na kolobarje
BM algoritem deluje na zaporedjih z elementi iz končnega obsega, Reeds in Sloane [79] pa sta leta 1985 predstavila razširitev BM algoritma za LFSR sintezo na delovanje nad kolobarjem $\mathbb{Z}_m$, kjer je $m \in \mathbb{N}$ poljubno velik s poznano faktorizacijo. Interlando et al. [43] so leta 1997 na kolobar $\mathbb{Z}_m$ razširili tudi Petersonov (Berlekampov) algoritem za dekodiranje. Dve leti kasneje pa sta de Andrade in Palazzo [2] dekodiranje posplošila na vse končne komutativne kolobarje z enico. Norton in Sălăgean [68] sta preučevala reševanje ključne enačbe s pomočjo Gröbnerjevih baz.

Kvadratična zahtevnost zaporedja
je definirana kot dolžina najkrajšega FSR generatorja s polinomom stopnje največ 2 kot povratno funkcijo. Youssef in Gong [95] s teoretičnimi in eksperimentalnimi rezultati predvidevata, da pričakovano kvadratično zahtevnost lahko določimo podobno kot je to storil Rueppel [80] za linearno zahtevnost, glej razdelek 4.6. Postavila sta hipotezo, da je pričakovana kvadratična zahtevnost naključnega binarnega zaporedja dolžine n enaka $\sqrt{2n}$.

Diskretna Fourierova transformacija,
ki smo jo uporabili že pri dokazu ekvivalentnosti dveh definicij Reed-Solomonovih kod, nam da še en lep rezultat, to je Blahutov [13] izrek iz leta 1979, ki pravi, da je

$$LS(\boldsymbol{b}) = w(\boldsymbol{B}) \quad \text{in} \quad w(\boldsymbol{b}) = LS(\boldsymbol{B}),$$

kjer je LS oznaka za linearno zahtevnos zaporedja, w oznaka za Hammingovo težo, $\boldsymbol{b}$ izbrano končno zaporedje nad $\mathbb{F}$, $\boldsymbol{B}$ pa njegova diskretna Fourierova transformacija (DFT). Več o DFT si lahko preberemo v Gong [31, Chapter 5].

S pomočjo Blahutovega izreka je Massey [59] pokazal **izrek položajev ničel** (ang. Zero-Location Theorem), ki pravi, da je minimalna razdalja ciklične (n, k)-kode enaka minimumu linearnih zahtevnosti neničelnih periodičnih zaporedij s periodo n, ki vsebujejo 0 na vseh tistih mestih v prvi periodi, kjer je 0 v DFT transformaciji zaporedja koeficientov generatorskega polinoma kode. S pomočjo tega rezultata je izpeljal tudi nekaj novih ocen za popravljanje grozdnih napak s cikličnimi kodami.

Profili linearnih zahtevnosti zaporedij z zamiki.
Piperova zahteva za kriptografsko varnost tokovnih šifer je tudi, da morajo biti profili linearne zahtevnosti zaporedja $\boldsymbol{a} = a_1, \ldots, a_n$ sprejemljivi tudi za zaporedja $\boldsymbol{a}^i = a_{i+1}, \ldots, a_n$, $1 \leq i \leq n-1$ (izbris prvih i členov oz. zamik začetka zaporedja za i mest). Lahko bi $(n-1)$-krat uporabili BM algoritem in s tem potrošili $O(n^3)$ operacij, lahko pa uporabimo Niederreiter-Vielhaberjev algoritem [66] iz leta 1999 za izračun **verižnih ulomkov** (**CFE** - continued fraction expansion) in s tem potrošimo $O(n^2 \log n)$ operacij nad $\mathbb{F}$.

Slike

Literatura

[1] M. Agrawal, N. Kayal, N. Saxena, *PRIMES is in P*, Preprint, 6. avg. 2002. `http://www.cse.iitk.ac.in/primality.pdf`

[2] A. A. de Andrade, R. Jr. Palazzo, *Construction and decoding of BCH codes over finite commutative rings*, Linear Algebra Appl., **286, no. 1-3** (1999), 69–85.

[3] T. M. Apostol, *Introduction to Analytic Number Theory*, Springer-Verlag, 1996.

[4] A. O. L. Atkin, F. Morain, *Elliptic Curves and Primality Proving*, Mathematics of Computation, 1993, 29-68.

[5] R. C. Baker, G. Harman, *The Brun-Titchmarsh Theorem on average*, Proceedings of a conference in Honor of Heini Halberstam, Volume 1, 1996, 39–103.

[6] L. A. Bassalygo, *New upper bounds for error correcting codes*, Problems of Information Transmission, **Vol. IT 1, No.1** (1965), 32–35.

[7] E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York etc, 1968.

[8] E. R. Berlekamp, *Algebraic Coding Theory, Revised 1984 Edition*, Aegean Park Press, Walnut Creek CA, 1984.

[9] R. Bhattacharjee, P. Pandey, *Primality testing. Technical report*, IIT Kanpur, 2001.

[10] P. Billingsley, *Probability and Measure, 2nd ed.*, John Wiley, New York, 1986.

[11] D. Bini, V. Pan, *Polynomial and Matrix Computation, Fundamental Algorithms, Vol. 1*, Birkhauser, Boston-Basel-Berlin, 1994.

[12] A. Biryukov, A. Shamir, *Structural cryptanalysis of SASAS*, Proceedings of Eurocrypt '01, B. Pfitzmann (editor), LNCS 2045, Springer-Verlag, 2001, 394–405.

[13] R. E. Blahut, *Transform techniques for error-control codes*, IBM. J. Res. Devel., **Vol. 23** (1979), 299–315.

[14] R. C. Bose, D. Ray-Chaudhuri, *On a class of error correcting binary group codes*, Info. and Control, **3** (1960), 68–79.

[15] D. Bressoud, *Factorization and Primality Testing*, SpringerVerlag, 1989.

[16] G. Carter, *A statistical test for randomness based on the linear complexity profile of a binary sequence*, Technical Report for Racal Comsec Ltd., 1987.

[17] R. T. Chien, *Cyclic decoding procedures for Bose-Chaudhuri-Hocquenghem codes*, IEEE Transactions on Information Theory, **Vol. IT 10, No.4** (October 1964), 357–363.

[18] D. Coppersmith, H. Krawczys in Y. Mansour, *The schrinking generator*, Advances in Cryptology - CRYPTO '93, D. R. Stinson (editor), LNCS 773, Springer-Verlag, 1994, 22–39.

[19] N. T. Courtois, *Higher order correlation attacks, XL algorithm, and cryptanalysis of toyocrypt*, Proceedings of ICISC '02, K. Kim (editor), LNCS 2587, Springer-Verlag, 2002.

[20] N. T. Courtois, A. Klimov, J. Patarin in A. Shamir, *Efficient algorithms for solving overdefined systems of multivariate polynomial equations*, Proceedings of Eurocrypt '00, B. Preneel (editor), LNCS 1807, Springer-Verlag, 2000, 392–407.

[21] W. Diffie, M. E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory, **22** (1976), 644–654.

[22] C. S. Ding, *Proof of Massey's conjectured algorithm*, Advances in cryptology – EUROCRYPT '88 (Davos, 1988), LNCS 330, Springer-Verlag, 1988, 345–349.

[23] G. L. Feng, K. K. Tzeng, *A generalization of the Berlekamp-Massey algorithm for multisequence shift-register synthesis with applications to decoding cyclic codes*, IEEE Trans. Inform. Theory, **Vol. 37** (Sept. 1991), 1274–1287.

[24] E. Filiol, *Decimation attack of stream ciphers*, Progress in Cryptology - INDOCRYPT '00, LNCS 1977, Springer-Verlag, 2000, 31–42.

[25] G. D. Forney, *Concatenated Codes*, The M.I.T. Press, Cambridge, Massachusetts, 1966.

[26] E. Fouvry, *Theoreme de Brun-Titchmarsh application au theoreme de Fermat*, Invent. Math., **79** (1985), 383–407.

[27] R. A. Games, A. H. Chan, *A fast algorithm for determining the linear complexity of a pseudorandom sequence with period* 2^n, IEEE Transactions on Information Theory, **Vol. 29, No.1** (January 1983), 144–146.

[28] O. Goldreich, R. Rubinfeld, M. Sudan, *Learning polynomials with queries: The highly noisy case*, Procedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, 1995, 294–303.

[29] S. Goldwasser, J. Kilian, *Almost All Primes Can Be Quickly Certified*, Proceedings of 18th STOC, 1986, 316-329.

[30] S. W. Golomb, *Shift Register Sequences, Revised Edition*, Aegean Park Press, 1982.

[31] Guang Gong, *Sequence Analysis*, Lecture Notes for CO739x, University of Waterloo, Winter 1999 (45–75, 109–125).

[32] S. W. Golomb, G. Gong, *Transform domain analysis of DES*, IEEE Transactions on Information Theory, **Vol. 45, No.6** (September 1999), 2065–2073.

[33] D. C. Gorenstein, N. Zierler, *A class of error correcting codes in* p^m *symbols*, J. Soc. Indus. Applied Math., **9** (1961), 207–214.

[34] D. Grigoriev, *Factorization of Polynomials over a Finite Field and the Solution of Systems of Algebraic Equations*, Translated from Zapiski Nauchnykh Seminarov Lenningradskogo Otdeleniya Matematicheskogo Instituta im. V. A. Steklova AN SSSR, Vol. 137, 1984, 20–79.

[35] V. Guruswami, M. Sudan, *Improved decoding of Reed-Solomon and algebraic-geometric codes*, IEEE Transactions on Information Theory, **Vol. 45** (1999), 1757–1767.

[36] V. Guruswami, M. Sudan, *Reflections on "Improved decoding of Reed-Solomon and algebraic-geometric codes"*, IEEE Information Theory Newsletter, (Mar. 2002), .
`http://www.cs.washington.edu/homes/venkat/pubs/papers.html`
(V. Guruswamijeva domača stran)

[37] F. F. Gustavson, *Analysis of the Berlekamp-Massey linear feedback shift-register synthesis algorithm*, IBM J. Res. Develop., 1976.

[38] R. W. Hamming, *Error detecting and error correcting codes*, Bell Systems Technical Journal, **29** (April, 1950), 147–160.

[39] D. R. Hankerson in ostali, *Coding Theory and Cryptography : The Essentials; Second Edition, Revised and Expanded*, M. Dekker, New York, Basel, 2000.

[40] G. H. Hardy, J. E. Littlewood, *Some problems of 'Partitio Numerorum' III: On the expression of a number as a sum of primes*, Acta Mathematica, **44** (1922), 1–70.

[41] E. Horowitz, S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, 1978.

[42] C. Huffman, V. Pless, *Handbook of Coding Theory, Volumes 1* & *2*, Elsevier Sciences, 1998.

[43] J. C. Interlando, R. Jr. Palazzo, M. Elia, *On the decoding of Reed-Solomon and BCH codes over integer residue rings*, IEEE Trans. Inform. Theory, **Vol. 43, no. 3** (1997), 1013–1021.

[44] S. M. Jennings, *A Special Class of Binary Sequences*, Phd Thesis, University of London, 1980.

[45] A. Jurišić, *Napake niso večne*, Presek, **30** (2002/03), 361–366.

[46] J. Justesen, *A class of constructive asymptotically good algebraic codes*, IEEE Transactions on Information Theory, **Vol. 18** (1972), 652–656.

[47] J. Justesen, *Bounds on List Decoding of MDS codes*, Manuscript, April 1998.

[48] E. Kaltofen, *A Polynomial-Time Reduction from Bivariate to Univariate Integral Polynomial Factorization*, In 23rd Annual Symposium on Foundations of Computer Science, 1982, 57–64.

[49] E. Kaltofen, *Polynomial Factorization 1987–1991*, LATIN '92, I. Simon (editor), LNCS 583, Springer-Verlag, 1992, 294–313.

[50] N. Kayal, N. Saxena, *Towards a deterministic polynomial-time test. Technical report*, IIT Kanpur, 2002.

[51] E. L. Key, *Analysis of the structure and complexity of nonlinear binary sequence generators*, IEEE Transactions on Information Theory, **IT-22** (November 1976), 732–736.

[52] A. Klapper, *On the existence of secure keystream generators*, J. Cryptology, **vol.14, no. 1** (2001), 1–15. (Reviewer: Edlyn E. Teske)

[53] A. Klapper, M Goresky, *Feedback shift registers, 2-adic span, and combiners with memory*, Journal of Cryptology, **Vol. 10, No. 2** (1997), 111–147.

[54] S. Klavžar, *O teoriji kodiranja, linearnih kodah in slikah z Marsa*, Obzornik za matematiko in fiziko, **45** (1998), 97–106.

[55] A. G. B. Lauder, K. G. Paterson, *Computing the error linear complexity spectrum of a binary sequence of period* 2^n, poslano v objavo, (2000), .
`http://www.hpl.hp.com/techreports/1999/HPL-1999-128R1.html`
(HP Labs Technical Reports)

[56] F. J. MacWilliams, N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North Holland, Amsterdam, 1977.

[57] G. Marsaglia, *A current view of random number generators*, Proceedings of the Sixteenth Symposium on the Interface (Atlanta, Georgia, March 1984), Computer Science and Statistics, Elsevier Science Publishers, New York, 1985, str. 3–10.

[58] G. Marsaglia, *Monkey tests for random number generators*, Computers & Mathematics with Applications, **9** (1993), 1–10.

[59] J. L. Massey, *Linear complexity in coding theory*, Coding theory and applications (Cachan, 1986), LNCS 311, Springer-Verlag, 1988, 19–32.

[60] J. L. Massey, *Shift-register synthesis and BCH decoding*, IEEE Transactions on Information Theory, **Vol. IT 15, No.1** (January 1969), 122–127.

[61] H. F. Mattson Jr., G. Solomon, *A new treatment of Bose-Chaudhuri codes*, J. Soc. Indust. Appl. Math., **9** (1961), 654–669.

[62] R. J. McEliece, E.R. Rodemich, H.C. Rumsey, L.R. Welch, *New upper bounds on the rate of a code via the Delsarte-Macwilliams inequalities*, IEEE Transactions on Information Theory, **Vol. IT 23** (1977), 157–166.

[63] W. Meier, O. Staffelbach, *Fast correlation attacks on certain stream ciphers*, Journal of Cryptology, **Vol. 1, No. 3** (1989), 159–176.

[64] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, Series on Discrete Mathematics and its Applications, 4th ed., CRC Press, 1999.

[65] F. Morain, *Implementation of the AtkinGoldwasserKilian Primality Testing Algorithm*, INRIA Research Report, #911, Oktober 1988.

[66] H. Niederreiter, M. Vielhaber, *An algorithm for shifted continued fraction expansions in parallel linear time*, Theoretical Computer Science, **226** (1999), 93–104.

[67] G. H. Norton, *On n-dimensional sequences, I*, Journal of Symbolic Computation, **Vol. 20, no. 1** (1995), 71–92.

[68] G. H. Norton, A. Sălăgean, *On the key equation over a commutative ring*, Des. Codes Cryptogr., **20, no. 2** (2000), 125–141.

[69] B. Preneel in ostali (skupaj 20 avtorjev), *NESSIE Security Report, Version 2.0*, February 19, 2003.
`www.cryptonessie.org`

[70] B. Preneel, *NESSIE Project Announces Final Selection of Crypto Algorithms*, February 27, 2003.
`www.cryptonessie.org`

[71] T. Novak, *Preverjanje naključnosti v kriptografiji*, diplomsko delo, Fakulteta za matematiko in fiziko, Univerza v Ljubljani, 2000.

[72] S. Palit, K. Roy, *Cryptanalysis of LFSR-encryption codes with unknown combining function*, Advances in Cryptology - ASIACRYPT '99, K. Y. Lam in ostali (editors), LNCS 1716, Springer-Verlag, 1999, 306–320.

[73] R. Petek, *RSA kriptosistem, diplomsko delo*, Ljubljana, 2000.

[74] W. W. Peterson, *Encoding and error-correction procedures for Bose-Chaudhuri codes*, IRE Transactions on Information Theory, **Vol. IT 6** (1960), 459-470.

[75] W. W. Peterson, *Error-Correcting Codes*, The M.I.T. Press, Cambridge, Massachusetts, 1961.

[76] M. Petkovšek, T. Pisanski, *Izbrana poglavja iz računalništva, 1. del: Izračunljivost in rešljivost, jeziki, NP-polnost, naloge*, DMFA SR Slovenije, Ljubljana, 1986.

[77] T. Pisanski, *Izračunljivost in rešljivost*, Obzornik za matematiko in fiziko, **25** (1978), 41–54.

[78] I. S. Reed, G. Solomon, *Polynomial codes over certain finite fields*, J. SIAM, **8** (1960), 300–304.

[79] J. A. Reeds, N. J. A. Sloane, *Shift-register synthesis (modulo m)*, SIAM J. Comput., **Vol. 14** (1985), 505–513.

[80] R. A. Rueppel, *Analysis and Design of Stream Ciphers*, Springer-Verlag, 1986.

[81] C. E. Shannon, *A mathematical theory of communication*, Bell Systems Technical Journal, **27** (1948), 379–423, 623–656.

[82] C. E. Shannon, *Communication theory of secrecy systems*, Bell Systems Technical Journal, **28** (1949), 656–715.

[83] M. A. Shokrollahi, *Computing Roots of Polynomials over Function Fields of Curves*, Manuscript, Junij 1998.

[84] T. Siegenthaler, *Decrypting a class of stream cipher using ciphertext only*, IEEE Transactions on Computers, **Vol. 34, No. 1** (January 1985), 81–85.

[85] R. C. Singleton, *Maximum distance q-nary codes*, IEEE Transactions on Information Theory, **Vol. IT 10** (1964), 116–118.

[86] M. Stamp, C. F. Martin, *An algorithm for the k-error linear complexity of binary sequences of period* 2^n, IEEE Transactions on Information Theory, **Vol. 39** (July 1993), 1398–1401.

[87] D. R. Stinson, *Cryptography: Theory and Practice*, CRC Press, 1995.

[88] M. Sudan, *Decoding of Reed-Solomon codes beyond the error-correction bound*, Journal of Complexity, **13** (1997), 180–193.

[89] M. Sudan, *Coding theory: Tutorial & survey*, Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, Las Vegas, Nevada, 14-17 October, 2001, 36–53.
`http://theory.lcs.mit.edu/ madhu/papers.html`
(M. Sudanova domača stran)

[90] Y. Sugiyama, M. Kasahara, S. Hirasawa, T. Namekawa, *A method for solving key equation for decoding Goppa codes*, Inform. Contr., **Vol. IT 27** (1975), 87–99.

[91] M. A. Tsfasman, S. G. Vlăduţ, T. Zink, *Modular curves, Shimura curves and Goppa codes, better than Varshamove-Gilbert bound*, Math. Nachr., **Vol. 104** (1982), 13–28.

[92] I. Vidav, *Algebra, 2. natis*, DMFA SRS, IMFM in FNT, Ljubljana, 1980.

[93] S. A. Vanstone, P. C. van Oorschot, *An Introduction to Error Correcting Codes with Applications*, Kluwer Academic Publishers, Boston-Dordrecht-London, 1989.

[94] T. Vidmar, *Računalniška omrežja in storitve*, Atlantis, Ljubljana, 1997.

[95] A. M. Youssef, G. Gong, *On On the Quadratic Span of Binary Sequences*, CACR Technical Report, University of Waterloo, Canada, 2000.

[96] C. J. Zarowski, *A divisionless form of the Schur Berlekamp-Massey algorithm*, Information theory and applications (Rockland, ON, 1993), LNCS 793, Springer-Verlag, 1994, 38–44.

[97] K. Zeng, M. Huang, *On the linear syndrom analysis method in cryptoanalysis*, Advances in Cryptology - CRYPTO '88, S. Goldwasser (editor), LNCS 403, Springer-Verlag, 1990, 469–478.

[98] K. Zeng, C. H. Yang, T. R. N. Rao, *On the linear consistency test (LCT) in cryptoanalysis with applications*, Advances in Cryptology - CRYPTO '89, G. Brassard (editor), LNCS 435, Springer-Verlag, 1990, 164–174.

[99] K. Zeng, C. H. Yang, T. R. N. Rao, *An improved linear syndrom analysis algorithm in cryptoanalysis with applications*, Advances in Cryptology - CRYPTO '90, A. Menezes in S. A. Vanstone (editors), LNCS 537, Springer-Verlag, 1991, 34–47.

Izjava

Podpisani Tadej Novak izjavljam, da sem avtor tega magistrskega dela.