UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

VANJA NASTRAN

# BAZE BINARNIH KONČNIH OBSEGOV

DIPLOMSKO DELO

Ljubljana, 2003

**Povzetek**

*V tem delu bomo definirali baze binarnih končnih obsegov, ki se v kriptografiji nad eliptičnimi krivuljami uporabljajo najpogosteje. To so polinomske, normalne in sebidualne baze. Pri polinomskih bazah bomo posebej obravnavali tiste, ki imajo za redukcijski polinom trinom ali pentonom. Pri normalnih bazah pa bomo definirali posebni vrsti baz, to so optimalne normalne in umbralne baze. Glede na različne baze bomo podali najhitrejše algoritme za osnovne operacije; to so seštevanje, množenje, kvadriranje in računanje inverza. Naredili bomo podrobno analizo algoritmov in primerjavo med njimi. Poleg tega bomo predstavili algebraične osnove končnih obsegov in eliptičnih krivulj, ki so pomembne za javno kriptografijo z eliptičnimi krivuljami.*

**Ključne besede:** Eliptične krivulje, polinomske baze, normalne baze, optimalne normalne baze, umbralne baze, sebidualne baze, končni obsegi.

**Abstract**

*In this thesis we will define bases for binary finite fields, which are most common in elliptic curve cryptography. These are polynomial, normal and self-dual bases. We will concentrate on those polynomial bases, that are defined with irreducible trinomial or pentonomial. Also two special kind of normal bases will be defined, optimal normal bases and umbral bases. We will describe the fastest algorithms for basic operations in different bases; these are addition, multiplication, squaring and inversion. We will make a detailed analysis of these algorithms and their comparison. We will also introduce some basics about finite fields and elliptic curves, which are of particular interest to public criptography with elliptic curves.*

**Key words:** Elliptic curves, polynomial bases, normal bases, optimal normal bases, umbral bases, self-dual bases, finite fields.

*Zahvaljujem se svojemu mentorju doc. dr. Aleksandru Jurišiću za vso pomoč pri pisanju tega dela.*
*Velika zahvala gre mojim staršem in sestri za spodbudo in podporo. Zahvaljujem se Tini za koristne nasvete. Hvala tudi vsem prijateljem, še posebej ljubemu Mitji, ki mi stoji ob strani.*

# PROGRAM DIPLOMSKEGA DELA

Delo naj predstavi matematične osnove, potrebne za razumevanje osnovnih računskih operacij končnih obsegov karakteristike 2, ki so posebno zanimivi za kriptografijo. Glavna cilja sta predstavitev in primerjava učinkovitosti seštevanja, množenja, kvadriranja in invertiranja v

(a) polinomskih bazah (posebej tistih, kjer si za nerazcepen polinom izberemo trinom, pentonom ali heptonom),

(b) normalnih bazah (posebej optimalnih in umbralnih oziroma Chebishevih), ter

(c) sebidualnih bazah.

za implementiranje seštevanja in podvajanja točk na eliptični krivulji.

**Literatura:**

D. Hankerson, J. L. Hernandez, and A. Menezes, Software implementation of elliptic curve cryptography over binary fields, Cryptographic hardware and embedded systems - CHES 2000, LNCS **1965**, 2000, 1–24.

A. J. Menezes (editor), I. F. Blake, CuHong Gao, R. C. Mullin, S. A. Vanstone, T. Yaghoobian, *Applications of Finite Fields*, Kluwer Academic Publishers, 1993.

Douglas R. Stinson, *Cryptography – Theory and Practice*, CRC Press, 1995.

A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press (Series on Discrete Mathematics and its Applications), 4th ed, 1999.

# Kazalo

# UVOD

Pričujoča diplomska naloga se ukvarja z učinkovito implementacijo aritmetike nad binarnimi končnimi obsegi. Ta ima izreden pomen pri javni kriptografiji nad eliptičnimi krivuljami. Le-te so se pojavile v algebraični teoriji že pred stoletji, a so bile sprva zanimive zgolj s teoretičnega vidika. Z razvojem računalnikov in kriptografije pa so se začele uporabljati v povsem praktične namene. Predstavljajo namreč pomembnejši del kriptografije z javnim ključem. Ta je danes eden najučinkovitejših načinov za zaščito in varnost podatkov, ki jih pošiljamo preko elektronskih kanalov. Zaradi vse manjših sistemov, ki potrebujejo učinkovito zaščito podatkov (npr. manjše prenosne naprave, baterije,...), in zaradi vse zmogljivejših računalnikov morajo biti tudi kriptosistemi vse učinkovitejši.

Kriptografijo delimo na dve pomembnejši veji, to sta kriptografija s simetričnim ključem (simetrična kriptografija) in kriptografija z javnim ključem (javna kriptografija). Pri simetrični kriptografiji potrebujemo isti ključ za šifriranje in dešifriranje. Ta način je v uporabi že stoletja in je matematično enostavno izvedljiv. Vendar obstaja problem varne izmenjave simetričnega ključa. Najpomembnejši kriptosistemi s simetričnim ključem so DES (Data Encryption Standard), AES (Advanced Encryption Standard), IDEA in njihove različice. Leta 1976 pa sta Whitfield Diffie in Martin Hellman predlagala nov koncept, to je kriptografijo z javnimi ključi. V tem primeru ima vsak udeleženec svoj par ključev, privatni ključ in javni ključ. Če želi nekdo poslati zaupno sporočilo, potem z javnim ključem prejemnika podatke zašifrira. Prejemnik zašifrirano sporočilo s privatnim ključem odšifrira. Če mora biti sporočilo še podpisano (avtentično), potem ga pošiljatelj zašifrira s svojim privatnim ključem, prejemnik pa odšifrira z javnim ključem pošiljatelja. Temu pravimo digitalni podpis. Varnost sistema temelji na težkih matematičnih problemih (npr. problem diskretnega logaritma, ki ga bomo opisali), katere ni moč rešiti tudi z najzmogljivejšimi računalniki v več tisoč letih. Ker so matematični algoritmi za kriptografijo z javnim ključem zapletenejši od tistih za simetrično kriptografijo in tudi zahtevajo več časa za implementacijo, navadno združimo oba pristopa. Za izmenjavo skrivnega simetričnega ključa uporabimo javno kriptografijo, nato pa komuniciramo s pomočjo simetrične kriptografije.

Najprej povejmo, da velja matematični problem za težkega, če ne obstaja učinkovit algoritem, ki bi ga rešil v določenem času. Algoritem je učinkovit, če je njegova časovna zahtevnost polinomska glede na vhodne podatke. Če je časovna zahtevnost eksponentna, smatramo algoritem za neučinkovitega, oziroma je matematični problem težek. Odkar se je pojavila javna kriptografija, so predlagali mnogo kriptografskih sistemov z javnimi ključi. Vsi ti sistemi temeljijo na težko rešljivih matematičnih problemih. Mnoge so kasneje zavrgli, saj se je izkazalo da so lažje rešljivi

kot se je sprva zdelo, ali pa so bili nepraktični. Danes veljajo za najbolj varne in učinkovite trije javni kriptosistemi. To so sistemi, ki temeljijo na naslednjih matematičnih problemih:

- faktorizacija velikih števil (takšen sistem je RSA);
- reševanje diskretnega logaritma (na primer sistem DSA);
- reševanje diskretnega logaritma na eliptični krivulji (na primer sistem ECDSA).

Za nobenega ni dokazano, da je res težko rešljiv problem. Opiramo se lahko le na dejstvo, da vodilni matematiki še niso našli učinkovitega algoritma, ki bi jih rešil. Sistem je tem varnejši, čim dlje časa potrebuje najučinkovitejši poznani algoritem na najbolj zmogljivem računalniku oziroma mreži računalnikov za odkritje tajnega ključa.

Prvi praktični kriptosistem z javnimi ključi so leta 1977 predlagali Ron Rivest, Adi Shamir in Len Adleman. To je RSA (ime je dobil po svojih izumiteljih). Opira se na težavnost problema **faktorizacije velikih števil**. Glavna ideja je, da težko poiščemo faktorja naravnega števila $n$, ki je produkt dveh velikih praštevil. Glede na najboljšo poznano računalniško tehnologijo velja danes sistem RSA za varnega, če je število $n$ dolgo vsaj 230 decimalnih številk (to je približno 760 bitov).

Naslednji problem, ki se uporablja za javno kriptografijo, je **problem diskretnega logaritma** po modulu praštevila $p$ (DLP). Podano imamo praštevilo $p$, naravno število $g$ med 0 in $p-1$ in število $y$, ki je neka potenca števila $g$ po modulu $p$. Torej velja med številoma $g$ in $y$ relacija

$$y = g^x \pmod{p}$$

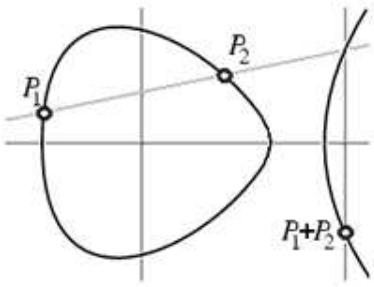
za neko število $x$, ki ga moramo poiskati. Ne poznamo učinkovitega algoritma, ki bi rešil problem diskretnega logaritma po modulu $p$. Prvi tak kriptosistem sta predlagala leta 1976 Whitfield Diffie in Martin Hellman. Danes je znan kot Diffie-Hellmanova izmenjava ključev. Najbolj znan sistem, ki temelji na problemu diskretnega logaritma, je DSA (Digital Signature Algorithm). DSA je avgusta leta 1991 predlagal ameriški Nacionalni inštitut za standarde in tehnologijo (NIST). Je različica ElGamalovega kriptosistema (Taher ElGamal). Danes velja DSA za varnega, če sestavlja praštevilo $p$ vsaj 230 digitalnih števil oziroma 760 bitov, kar je toliko kot pri sistemu RSA.

Zgoraj smo definirali DLP po modulu praštevila $p$. To pa ni edina struktura, nad katero lahko definiramo DLP. Leta 1985 sta Neal Koblitz in Victor Miller (neodvisno drug od drugega) predlagala kriptosistem nad eliptičnimi krivuljami (angl.

Elliptic Curve Cryptosystem, kratica ECC). Njegova varnost temelji na reševanju **problema diskretnega logaritma v grupi na eliptični krivulji** (ECDLP). Na kratko opišimo eliptične krivulje nad obsegom realnih števil. **Eliptična krivulja** $\mathcal{E}$ je množica rešitev $(x, y)$ enačbe oblike

$$y^2 = x^3 + ax + b,$$

za neki števili $a$ in $b$, skupaj s točko v neskončnosti $\mathcal{O}$. Če par $(x, y)$ zadošča zgornji enačbi, je $P = (x, y)$ točka na eliptični krivulji $\mathcal{E}$. Seštevanje in podvajanje točk nad eliptičnimi krivuljami definiramo s posebnim pravilom, ki mu rečemo sekantno/tangentno pravilo. Grafično si sekantno pravilo za seštevanje točk predstavljamo takole. Točki $P_1$ in $P_2$ sta elementa eliptične krivulje $\mathcal{E}$. Skozi njiju potegnemo premico, ki seka $\mathcal{E}$ v tretji točki. To točko preslikamo čez abciso in dobljena točka je definirana kot vsota $P_1 + P_2$. Če premica skozi dani točki $P_1$ in $P_2$ ne seka krivulje v nobeni drugi točki, torej velja $P_1 = -P_2$, je njuna vsota enaka točki $\mathcal{O}$. Pravilo za seštevanje je prikazano na sliki 1.




Slika 1: Sekantno pravilo seštevanja točk na eliptični krivulji.

Kaj pa če je $P_1 = P_2$? Takrat ravnamo po tangentnem pravilu. Skozi točko $P_1$ z eliptične krivulje $\mathcal{E}$ potegnemo tangento. Ta seka krivuljo še v drugi točki, ki jo preslikamo čez abciso. Ta točka je definirana kot $2P_1$. V treh primerih pa tangenta ne seka krivulje v nobeni drugi točki. To so presečišča krivulje $\mathcal{E}$ z abcisno osjo. Dvakratnik teh točk je definiran kot točka $\mathcal{O}$.
Problem diskretnega algoritma na eliptičnih krivuljah je definiran takole. Podano imamo eliptično krivuljo $\mathcal{E}$. Za neko naravno število $x$ predstavlja produkt $xP$ večkratnik točke $P \in \mathcal{E}$ in ga lahko izračunamo z algoritmom podvoji_in_prištej v polinomskem času. Naj bo $Q \in \mathcal{E}$, tako da velja

$$Q = xP$$

za neko naravno število $x$. Ob danih $P$ in $Q$ moramo izračunati število $x$. Tudi za ECDLP ne poznamo učinkovitega algoritma, ki bi ga rešil. V resnici je ECDLP še težji problem od ostalih dveh matematičnih problemov, ki smo jih opisali. Zaradi tega velja danes kriptosistem nad eliptičnimi krivuljami za najmočnejši učinkovit

javni kriptosistem. Dolžina ključa za ECC je lahko pri enaki varnosti nekajkrat krajša kot pri sistemih s faktorizacijo števil ali pri diskretnem logaritmu po modulu $p$. Ključ dolžine 160 bitov nam pri ECC zagotavlja enako varnost kot ključ dolžine 1024 bitov pri RSA ali DSA. Najbolj znan kriptosistem na eliptičnih krivuljah je ECDSA (Elliptic Curve Digital Signature Algorithm). Je različica DSA, definiran na eliptičnih krivuljah. Najbolj zahtevna operacija pri implementaciji ECC je računanje $Q = xP$ za točko $P$ in neko naravno število $x$. Varnost ECC torej temelji na težavnosti ECDLP, učinkovitost pa je odvisna od hitrega izračuna $xP$ za neko naravno število $x$ in točko na krivulji $P$.

Pri vseh treh naštetih problemih poznamo nekatere izjeme, za katere obstaja dokaj enostavna rešitev. Alfred Menezes, Tatsuoki Okamoto in Scott Vanstone so pokazali, kako lahko ECDLP omejimo na DLP nad končnim obsegom. Temu pravimo MOV- redukcija. Vendar je ta algoritem učinkovit le za poseben razred EC, to so **supersingularne eliptične krivulje** (to so eliptične krivulje, ki imajo konstanto $b$ iz zgornje definicije enako nič). Še en poseben razred EC, v katerem je ECDLP precej enostavno rešljiv, so tako imenovane **anomalične krivulje**. To so EC, definirane nad končnim obsegom $\mathbb{F}_q$, ki imajo natanko $q$ točk. Napad nanje so neodvisno drug od drugega odkrili Igor Semaev, Nigel Smart ter Takazu Satoh in Kiyomichi Araki. Na srečo lahko pri implementaciji z enostavnim testom preverimo, če izbrana krivulja slučajno pripada kateremu od teh dveh razredov in se ji izognemo. Sicer sta danes najbolj znana algoritma za reševanje ECDLP Pollardova $\rho$-metoda in Pohlig-Hellmanova metoda.

Številne znanstvene skupine se ukvarjajo z raziskovanjem eliptičnih krivulj. Raziskave potekajo na mnogih univerzah (npr. University of Waterloo, Kanada), pa tudi v komercialnih organizacijah (npr. Certicom) in v agencijah, kot je ameriška NSA (National Security Agency). Omenili smo že nekaj matematikov, ki so prispevali pomemben delež na področju kriptografije na eliptičnih krivuljah.

Večina kriptografskih sistemov in protokolov je tudi standardiziranih. To je pomembno iz različnih vzrokov, recimo za lažji pregled nad varnostjo sistemov. Tako so tudi eliptične krivulje vsebovane v mnogih standardih, kjer so podrobno opisane. Podane so tudi priporočene konkretne eliptične krivulje in določeni obsegi, nad katerimi so definirane. Na ta način lahko pri implementaciji dobimo visoko varnost sistema, ki bi jo sami težko dosegli. S standardizacijo so lahko kriptosisteme z eliptičnimi krivuljami povzele organizacije po vsem svetu. Naštejmo nekatere standarde, ki obravnavajo eliptične krivulje.

> **ANSI X9** - ANSI (American National Standards Institute) je januarja leta 1999 vključila v svoj standard ECDSA (Elliptic Curve Digital Signature Algorithm).

**FIPS** - Ameriški narodni inštitut za standarde in tehnologijo (NIST) je februarja 2000 razširil obsoječi standard za digitalni podpis z ECDSA, kot je definiran v ANSI X9. Ta prenovljeni standard pomeni razmah kriptosistemov z eliptičnimi krivuljami v komercialne namene, saj lahko komercialni proizvodi vsebujejo ECC brez posebej pridobljenih dovoljenj, kot je bilo v veljavi do takrat.

**IEEE P1363** - Ta projekt je bil uradno odobren kot standard IEEE (Institute of Electrical and Electronics Engineers) februarja 2000. ECC je vključen v standard, ki vsebuje enkripcijo, podpise in protokole za izmenjavo ključev. Standard določa, da morajo biti eliptične krivulje definirane po modulu $p$ ali nad binarnimi končnimi obsegi.

Eliptične krivulje lahko definiramo nad različnimi obsegi. V kriptografiji se najpogosteje uporabljajo praštevilski obsegi in binarni končni obsegi. Eliptične krivulje nad praštevilskimi obsegi je v svoji diplomski nalogi preučil Srečko Maksimovič. V pričujoči nalogi pa se posvečamo eliptičnim krivuljam nad binarnimi končnimi obsegi. Pri kriptografiji z eliptičnimi krivuljami se soočimo s problemom učinkovite aritmetike s točkami na eliptični krivulji (predvsem seštevanjem in podvajanjem točk). To dosežemo tudi s čimboljšo implementacijo aritmetike z elementi iz končnega obsega. Torej moramo ločiti med aritmetiko s točkami z eliptične krivulje in aritmetiko z elementi iz končnega obsega. Mi se bomo posvetili implementaciji aritmetike z elementi iz binarnega končnega obsega. Obravnavali bomo osnovne operacije, to so seštevanje, množenje, kvadriranje in računanje inverza. Sem spada tudi reševanje kvadratne enačbe, s čemer se ne bomo ukvarjali. Podrobneje je obdelano v diplomskem delu Jerneja Barbiča.

Elemente iz končnega obsega lahko predstavimo na mnogo načinov, pri čemer lahko uporabimo različne baze. In prav s predstavitvijo elementov v določeni bazi lahko dosežemo izredno učinkovite izboljšave algoritmov. V kriptografiji z eliptičnimi krivuljami so se posebej odlikovale tri vrste baz. To so polinomske, normalne in sebidualne baze. Polinomske baze se razlikujejo glede na polinom, s katerim so definirane. S trinomi in pentonomi dosežemo najboljše implementacije algoritmov. Tudi normalne baze lahko raziskujemo še naprej. Posebna oblika normalnih baz so optimalne normalne baze, ki nas privedejo do izredno učinkovitih algoritmov za množenje. Iz posebne oblike optimalnih normalnih baz lahko razvijemo novo vrsto baz, to so umbralne baze. Z njimi lahko poenostavimo algoritem za računanje inverza, ki je v normalnih bazah precej zamudno. Umbralnim bazam pravimo tudi Čebiševe baze, zaradi sorodnosti s Čebiševimi polinomi. S sebidualnimi bazami pa dosežemo izredno ugodne rezultate pri zelo zanimivem algoritmu, to je množenje po bitih. Žal ne moremo hitro odgovoriti, katera baza je boljša od druge, saj ima vsaka svoje dobre, pa tudi slabe lastnosti. V nalogi smo skušali predstaviti najboljše algoritme za različne vrste baz, pri čemer smo izkoristili njihove ugodne lastnosti. Hkrati smo

tudi opozorili na pomankljivosti pri določenih bazah in možnosti, kako se jim izogniti (npr. pri normalnih bazah uvedemo optimalne baze, ali pri polinomskih bazah uporabimo trinome).

Poglavja v tem delu so sestavljena takole. Za razumevanje baz za končne obsege in aritmetike nad končnimi obsegi potrebujemo nekaj znanja iz algebre. Zato je prvo poglavje posvečeno algebraičnim osnovam. V naslednjem poglavju so opisane baze, s katerimi se bomo ukvarjali, in njihove lastnosti. Poglavje zaključimo z razmislekom o pretvarjanju med različnimi bazami. Tretje poglavje je najpomembnejše. V njem so najprej opisane osnove računanja nad eliptičnimi krivuljami, nato pa različni algoritmi za osnovne operacije nad binarnimi končnimi obsegi. Ločili jih bomo glede na bazo, v kateri so elementi predstavljeni. Četrto poglavje je posvečeno analizi opisanih algoritmov in njihovi primerjavi glede na količino časa in prostora, ki ga zahtevajo pri izvajanju. V zaključku bomo skušali strniti ugotovitve.

# Poglavje 1

# ALGEBRAIČNE OSNOVE

V tem poglavju bomo predstavili osnove iz teorije grup in obsegov. Teorija grup se ukvarja s sistemi, za katere ima enačba $a \circ x = b$ enolično rešitev. Ne zanima nas konkretno, kaj sta elementa $a$ in $b$ in ne, katero operacijo oznaka $\circ$ pomeni. S takšnim abstraktnim pristopom se teorija grup ukvarja z mnogimi matematičnimi sistemi naenkrat. Zahtevamo le, da matematični sistemi zadoščajo nekaterim enostavnim pravilom. Teorija nato preučuje lastnosti, ki so skupne tem sistemom. Matematiki so se do pravil oziroma aksiomov za grupe, kolobarje in obsege dokopali po več kot stotih letih trdega dela. Zasnovo zanje je leta 1771 postavil Joseph Louis Lagrange (1736-1813) s svojim izrekom, da moč poljubne podgrupe končne grupe $G$ deli število elementov grupe $G$. V teoriji grup je velikega pomena tudi delo Nielsa Henrika Abela (1802-1829). Rešitve enačb nižjih stopenj (do četrte stopnje) znamo zapisati s formulami. Abel pa je s teorijo grup pokazal, da to za enačbo pete stopnje ne velja več (zapisu rešitev s formulami pravimo radikali). Najpomembnejši mejnik na tem področju predstavlja francoski matematik Evariste Galois (1811-1832). Preučeval je rešljivost enačb in podal teorijo, ki ji danes pravimo *Galoisova teorija.* Galois je vpeljal pojma grupe in obsega. Angleški matematik Arthur Cayley (1821-1895) je leta 1854 dokazal, da lahko grupo definiramo ne glede na konkretno naravo njenih elementov. Osnove teorije grup najdemo na Internetni strani http://members.tripod.com/ dogschool/groups.html, več o teoriji grup in obsegov pa npr. v knjigi Ivana Vidava, Algebra [1].

V uvodnem razdelku bomo predstavili pojma kolobarja in obsega, definirali karakteristiko obsega ter podali pojem praobsega. V drugem razdelku se bomo posvetili razširitvam obsegov in osnovnim značilnostim razširitev. Definirali bomo različne razširitve (algebraična, končna, normalna,...). Spoznali bomo tudi osnove razpadnih obsegov. Naslednji razdelek obravnava končne obsege. Ukvarjali se bomo tudi z minimalnimi polinomi in množico konjugirank elementov. Poglavje zaključimo z razdelkom o osnovnih lastnostih sledi in norme.

## 1.1 UVOD

Spomnimo se, da je množica $G$ z neko binarno operacijo **grupa**, če za operacijo velja zaprtost in asociativnost, v $G$ obstaja enota za to operacijo in za vsak element obstaja njegov inverz. Če za operacijo velja še komutativnost, pravimo množici $G$ **Abelova grupa**. Definirajmo pojem kolobarja in obsega.

**Definicija 1.1.1. Kolobar** *je množica $K$ z dvema binarnima operacijama. Prvo operacijo imenujemo* **seštevanje**, njen kompozitum elementov $a, b \in K$ pa **vsota**, ki jo pišemo $a + b$. Drugo binarno operacijo imenujemo **množenje**, njen kompozitum elementov $a, b \in K$ imenujemo **produkt** in ga označimo z $ab$. Pri tem morajo biti izpolnjeni naslednji pogoji:

(i) za seštevanje je $K$ komutativna grupa z enoto 0,

(ii) za množenje je $K$ polgrupa (zaprtost in asociativnost množenja),

(iii) obe operaciji vežeta distributivnostna zakona:

$$(a + b)c = ac + bc,$$

$$c(a + b) = ca + cb.$$

Naj bo $K$ kolobar z enoto 1 za množenje, ki je različna od enote 0 za seštevanje. Element $u \in K$ je **obrnljiv**, če v $K$ obstaja njegov inverzni element, tj. obstaja $v \in K$, za katerega velja $uv = vu = 1$, in ga označimo z $u^{-1}$. Definirajmo še pojma obsega in karakteristike.

**Definicija 1.1.2. Obseg** *je tak kolobar z enoto 1 za množenje, v katerem je vsak od nič različen element obrnljiv.* **Podobseg** *je taka podmnožica nekega obsega, ki je za isti operaciji tudi obseg.*

**Definicija 1.1.3.** *Naj bo $K$ obseg in 1 njegova enota za množenje. Število $p$ imenujemo* **karakteristika** *obsega $K$, če je $p \in \mathbb{N}$ in je najmanjše število, za katerega velja $px = 1$ za vsak $x \in K$.*

V obsegu $K$ definirajmo zaporedje $u_0 = 0,\ u_i = u_{i-1} + 1$. Elementi zaporedja so $0,\ 1,\ 2 = 1 + 1,\ 3 = 1 + 1 + 1, \ldots$. Ti elementi so lahko vsi različni med seboj in rečemo, da ima obseg karakteristiko 0. Ali pa obstajata dva elementa, ki sta enaka. V tem primeru označimo prvi element, ki se ponovi, z $u_{k+c}$, tako da velja $u_{k+c} = u_k$. Če je $k \neq 0$, velja tudi $u_{k-1+c} = u_{k-1}$ in je prišlo do ponovitve že prej. Torej je $k = 0$ in $u_0 = u_c = 1 + 1 + \cdots + 1 = c \cdot 1 = 0$. Število $c$ je karakteristika obsega. Vidimo, da je 0 prvi element, ki se ponovi, elementi $\{u_0, u_1, \ldots, u_{c-1}\}$ pa so med seboj različni. Denimo, da je karakteristika obsega sestavljeno število, oziroma

$c = ab$ za neki naravni števili $a$ in $b$, $1 < a, b < c$. Očitno za zgornje zaporedje velja $u_{ij} = u_i u_j$ za poljubni naravni števili $i$ in $j$. Torej velja tudi $u_c = u_{ab} = u_a u_b$. Vendar je $u_c = 0$, $u_a$ in $u_b$ pa sta zaradi minimalnosti karakteristike različna od nič. Dobili smo protislovje. Torej je od nič različna karakteristika obsega vedno neko praštevilo.
Naj bo $p$ poljubno praštevilo. Množica ostankov po modulu $p$, ki jo označimo z $\mathbb{Z}_p$, tvori obseg. Sestavljajo ga števila $0, 1, 2, \ldots, p-1$. V tem obsegu velja $p \cdot 1 = p \equiv 0 \pmod{p}$, kar pomeni, da ima karakteristiko $p$. Torej obstajajo obsegi s poljubno praštevilsko karakteristiko. Obseg racionalnih števil $\mathbb{Q}$ in obsege $\mathbb{Z}_p$ ostankov po praštevilskem modulu $p$ imenujemo **praobsegi**. S $\mathbb{F}_q$ označimo obseg s $q$ elementi. Vsak končni obseg vsebuje nek praobseg. Tako vsak obseg $\mathbb{F}_{p^n}$ vsebuje vse podobsege $\mathbb{F}_p, \mathbb{F}_{p^2}, \mathbb{F}_{p^3}, \ldots, \mathbb{F}_{p^n}$.

## 1.2 RAZŠIRITVE OBSEGOV

V tem razdelku bomo definirali pojem razširitve obsegov in nekaj lastnosti razširitev. To znanje je pomembno pri končnih obsegih in s tem tudi pri naslednjem poglavju o bazah.

**Definicija 1.2.1.** *Če je $K$ podobseg obsega $F$, pravimo, da je $F$* **razširitev** *obsega $K$. Razširitev $F$ nad obsegom $K$ označimo s $F/K$.*

Množico vseh polinomov spremenljivke $x$ s koeficienti iz obsega $K$ označimo s $K[x]$. Očitno je $K[x]$ kolobar. Naj bo $f(x) \in K[x]$ poljuben polinom in $a \in F$ poljuben element. Tedaj vrednost $f(a)$ pripada obsegu $F$. Elementi $f(a)$, ki jih dobimo za vse polinome $f(x) \in K[x]$, tudi sestavljajo kolobar. Označimo ga s $K[a]$. Kolobar $K[a]$ je podkolobar obsega $F$.
Če je element $f(a)$ različen od nič, obstaja v obsegu $F$ inverzni element $(f(a))^{-1}$. Tako lahko definiramo kvociente $g(a)/f(a)$ kot produkte poljubnih elementov $g(a) \in K[a]$ z inverznim elementom $(f(a))^{-1}$. Vsi taki kvocienti sestavljajo obseg, ki ga označimo s $K(a)$. To je najmanjši podobseg obsega $F$, ki vsebuje $K$ in element $a$. Rečemo, da smo razširitev dobili s **privzemom** elementa $a$. Če je $a \in K$, je $K(a) = K$. Posplošimo ta pristop na $m$ elementov $a_1, a_2, \ldots, a_m \in F$. Tedaj vrednosti $f(a_1, a_2, \ldots, a_m)$ vseh polinomov $f(x_1, x_2, \ldots, x_m) \in K[x_1, x_2, \ldots, x_m]$ sestavljajo kolobar. Označimo ga s $K[a_1, a_2, \ldots, a_m]$. Spet za $f(a_1, a_2, \ldots, a_m) \neq 0$ obstaja inverzni element in lahko definiramo kvociente. Množica vseh kvocientov, kjer je imenovalec različen od nič, je spet nek obseg. To je najmanjša razširitev obsega $K$, ki vsebuje izbrane elemente $a_1, a_2, \ldots, a_m \in F$. Označimo jo s $K(a_1, a_2, \ldots, a_m)$ in rečemo, da smo jo dobili s privzemom elementov $a_1, a_2, \ldots, a_m$. Razširitev $K(a, b)$ dobimo tudi tako, da k obsegu $K(a)$ privzamemo element $b$. Podobno lahko pridemo do obsega $K(a_1, a_2, \ldots, a_m)$ tako, da zaporedoma privzamemo elemente $a_1, a_2, \ldots, a_m$.

**Definicija 1.2.2.** *Obseg $F$ je* **enostavna razširitev** *obsega $K$, če jo dobimo s privzemom enega samega elementa $a \in F$, tj. če je $F = K(a)$ za nek $a \in F$. V tem primeru imenujemo element $a$* **primitivni element** *razširitve.*

Upodobimo kolobar polinomov $K[x]$ v kolobar $K(a) = F$ tako, da priredimo polinomu $f(x) \in K[x]$ njegovo vrednost $f(a) \in K[a]$. Upodobitev $f(x) \to f(a)$ je homomorfizem. **Jedro** homomorfizma sestavljajo praslike enote, torej polinomi $f(x)$, za katere je $f(a) = 0$ oziroma $a$ je ničla polinoma $f$. Definirajmo algebraične elemente in njihove minimalne polinome.

**Definicija 1.2.3.** *Če vsebuje jedro homomorfizma $K(x) \to K(a)$ od nič različen polinom, pravimo elementu $a$* **algebraičen element** *nad obsegom $K$. Če je vsak element $a \in F$ algebraičen nad obsegom $K$, imenujemo obseg $F$* **algebraična razširitev** *obsega $K$.*

Obseg $F$ je Abelova grupa za seštevanje. Produkt poljubnega elementa iz $F$ s poljubnim elementom iz $K$ je tudi element obsega $F$. Torej je $F/K$ vektorski prostor nad obsegom $K$. Prostor $F$ ima lahko nad obsegom $K$ končno ali neskončno razsežnost. Razsežnost prostora $F$ nad $K$ bomo označili s $[F : K]$.

**Definicija 1.2.4.** *Če je obseg $F$ končno razsežen vektorski prostor nad obsegom $K$, pravimo, da je $F$* **končna razširitev** *obsega $K$. Število $[F : K]$ imenujemo* **stopnja razširitve** *in jo označimo z $m$.*

V preostanku razdelka bomo obravnavali samo končne razširitve obsegov. Naj bo $F$ obseg, ki je končna razširitev obsega $K$, in $m = [F : K]$. Pokažimo, da si lahko predstavljamo $F$ kot vektorski prostor dimenzije $m$ nad obsegom $K$. Po definiciji končne razširitve obstajajo v prostoru $F$ linearno neodvisni elementi $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$, ki sestavljajo njegovo bazo nad obsegom $K$. Torej lahko vsak element $a \in F$ zapišemo kot linearno kombinacijo baznih elementov:

$$a = \sum_{i=0}^{m-1} a_i \alpha_i,$$

kjer so $a_i \in K$. Zato lahko prostor $F$ enačimo s prostorom $K^m$, množico vseh $m$-teric nad $K$, in vsak element $a \in F$ zapišemo kot $a = (a_0, a_1, \ldots, a_{m-1})$. Obseg $F$ je res vektorski prostor dimenzije $m$ nad obsegom $K$. Poglejmo izrek o končnih razširitvah obsegov.

**Izrek 1.2.5.** *Vsaka končna razširitev $F/K$ je algebraična razširitev.*

**Dokaz.** Naj bo $a \in F$ poljuben element. Potence $a^0 = 1, a, a^2, \ldots$ ne morejo biti vse linearno neodvisne med seboj nad obsegom $K$, ker je vektorski prostor $F$ končno

razsežen. Naj bo $a^m$ prva potenca, ki se linearno izraža s prejšnjimi. Potem za neke $c_i \in K$ velja

$$a^m = c_0 + c_1 a + \cdots + c_{m-1} a^{m-1}.$$

Torej je $a$ koren enačbe $x^m - c_{m-1}x^{m-1} - \cdots - c_0 = 0$. Zato je element $a$ algebraičen nad obsegom $K$. □

Definirajmo še razpadne obsege in normalne razširitve.

**Definicija 1.2.6.** *Naj bo $f(x) \in K[x]$. Koreni $a_1, a_2, \ldots, a_m$ enačbe $f(x) = 0$ naj leže v obsegu $F$. Obseg $K(a_1, a_2, \ldots, a_m)$ je najmanjša razširitev obsega $K$, v kateri so vsi koreni dane enačbe in ga imenujemo* **razpadni obseg** *enačbe $f(x) = 0$ oziroma ustreznega polinoma $f(x) \in K[x]$. V razpadnem obsegu razpade polinom $f(x)$ na same linearne faktorje. Če vsak polinom $f(x) \in K[x]$, ki ima v obsegu $F$ vsaj eno ničlo, razpade v obsegu $F$ na same linearne faktorje, pravimo razširitvi $F$ obsega $K$* **normalna** *razširitev.*

Naslednji izrek govori o razpadnem obsegu polinomov. Njegov dokaz najdemo v [1, str. 282 in str. 294].

**Izrek 1.2.7.** *Naj bo $F/K$ normalna razširitev in $f(x)$ nerazcepen polinom s koeficienti iz obsega $K$. Njegov razpadni obseg je normalna razširitev obsega $K$. Razpadni obsegi polinoma $f$ so med seboj izomorfni.* □

# 1.3 KONČNI OBSEGI

Obseg s končnim številom elementov imenujemo **končen** ali **Galoisov** obseg. Naj bo $F$ končen obseg in $p > 0$ njegova karakteristika. Naj bo $q = p^n$ za nek $n \in \mathbb{N}$. Obseg $F$ vsebuje podobseg $\mathbb{F}_q$ ostankov po modulu $q$. Razsežnost vektorskega prostora $F$ nad podobsegom $\mathbb{F}_q$ je število $m = [F : \mathbb{F}_q]$. Če so elementi $\alpha_1, \alpha_2, \ldots, \alpha_m$ baza prostora $F$ nad podobsegom $\mathbb{F}_q$, lahko po izreku 1.2.5 vsak element $a \in F$ zapišemo v obliki $a = a_1\alpha_1 + a_2\alpha_2 + \cdots + a_m\alpha_m$, pri čemer so koeficienti $a_1, a_2, \ldots, a_m$ elementi obsega $\mathbb{F}_q$. Vsak od teh koeficientov je lahko katerikoli element iz $\mathbb{F}_q$. Ker obseg $\mathbb{F}_q$ vsebuje $q$ elementov, ima obseg $F$ natanko $q^m$ elemetov. Število elementov končnega obsega je vedno neka potenca njegove karakteristike. Zato bomo označevali končne obsege s $\mathbb{F}_{q^m}$, v literaturi pogosto najdemo tudi oznako $\mathrm{GF}(q^m)$ (Galoisov obseg). Prepričajmo se, da je $F = \mathbb{F}_{q^m}$ normalna razširitev obsega $\mathbb{F}_q$. S $F^*$ označimo množico obrnljivih elementov obsega $F$. Sestavljajo jo natanko vsi neničelni elementi iz obsega $F$. Množica $F^*$ je za množenje grupa. Rečemo ji tudi **multiplikativna grupa**. Njena moč je enaka $q^m - 1$. V teoriji grup obstaja zelo znan izrek, da vsak element $x$ iz končne grupe $G$ moči $m$ ustreza enačbi $x^m = e$ [1, izrek 21, str. 57]. Torej za vsak neničelen element $a \in F$ velja naslednja trditev.

**Trditev 1.3.1.** *Element $a$ je element obsega $\mathbb{F}_{q^m}$ natanko tedaj, ko je $a$ ničla enačbe $x^{q^m} - x$, oziroma, za vsak $a \in \mathbb{F}_{q^m}$ velja $a^{q^m} = a$.* □

Dobili smo $a^{q^m-1} = 1$ oziroma $a^{q^m} = a$ za $a \in F^*$. Tej enačbi zadošča tudi element $x = 0$, torej so njeni koreni ravno vsi elementi obsega $F$. Ker je stopnja enačbe $x^{q^m} - x$ enaka $q^m$, je vsak element obsega $F$ enostavna ničla te enačbe. Zato polinom $x^{q^m} - x \in F[x]$ v obsegu $F$ razpade na same linearne faktorje, oziroma

$$x^{q^m} - x = (x - a_1)(x - a_2) \cdots (x - a_{q^m}),$$

pri čemer so $a_1, a_2, \ldots, a_{q^m}$ vsi elementi obsega $F$. Koeficienti enačbe $x^{q^m} - x = 0$ so elementi obsega $\mathbb{F}_q$. Zato je $F$ normalna razširitev podobsega $\mathbb{F}_q$. Dobimo ga s privzemom korenov zgornje enačbe.
Rešitve enačbe $x^{q^m-1} = 1$ so $q^m$-ti koreni enote v obsegu $\mathbb{F}_{q^m}$. Ta enačba ima toliko rešitev, kot je njena stopnja, torej $q^m - 1$. Iz teorije o kolobarjih polinomov [Vidav, str. 163] vemo, da potem obstaja **primitivni koren** $\theta$. Vsi ostali koreni enačbe $x^{q^m-1} = 1$ so potence primitivnega korena. Koren $\theta$ je torej primitivni element razširitve $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$, tj. multiplikativna grupa obsega $F$ je ciklična. Obseg $F$ vsebuje poleg teh elementov samo še element 0. Torej sestavljajo obseg $F$ naslednji elementi

$$\{0, \theta, \theta^2, \ldots, \theta^{q^m} = 1\}.$$

Brez dokaza navedimo še Wedderburnov izrek. Dokaz najdemo v [1, str.288].

**Izrek 1.3.2 (Wedderburnov izrek).** *Vsak obseg s končnim številom elementov je komutativen.* □

Naslednja trditev je zelo uporabna pri računanju nad končnimi obsegi.

**Trditev 1.3.3.** *Naj bosta $a$ in $b$ elementa končnega obsega $\mathbb{F}_{q^m}$ s karakteristiko $p$ in $q = p^n$ za neko naravno število $n$. Potem velja:*

$$(a + b)^q = a^q + b^q.$$

**Dokaz.** Po binomskem izreku velja

$$(a + b)^q = \sum_{i=0}^{q} \binom{q}{i} a^{q-i} b^i = \binom{q}{0} a^q + \sum_{i=1}^{q-1} a^{q-i} b^i + \binom{q}{q} b^q$$

Število $\binom{q}{i}$ je naravno število in za $1 \leq i \leq q - 1$ velja

$$\binom{q}{i} = \frac{q(q-1)\cdots(q-i+1)}{i(i-1)\cdots 2 \cdot 1}.$$

Ker je $q = p^n$, se ne krajša z nobenim faktorjem iz imenovalca. Torej za $q \leq i \leq q-1$ velja $\binom{q}{i} \equiv 0 \pmod{q}$ in so v zgornji enačbi vsi srednji členi enaki 0. Trditev je s

tem dokazana. □

Enakost iz zgornje trditve se da razširiti na več členov in za $a_1, a_2, \ldots, a_m \in \mathbb{F}_{q^m}$ velja:

$$(a_1 + a_2 + \cdots + a_m)^q = a_1^q + a_2^q + \cdots + a_m^q.$$

Podrobneje poglejmo še minimalne polinome elementov iz končnih obsegov.

**Definicija 1.3.4.** *Naj bo $F = \mathbb{F}_{q^m}$ končen obseg nad obsegom $\mathbb{F}_q$ s karakteristiko $p$ in $\alpha \in F$ neničelen element.* **Minimalni polinom** *za $\alpha$ glede na obseg $\mathbb{F}_q$ je moničen polinom $m(x)$ najnižje stopnje iz $\mathbb{F}_q[x]$, ki ima $\alpha$ za ničlo.*

Minimalni polinom elementa $\alpha \in F = \mathbb{F}_{q^m}$ glede na $\mathbb{F}_q$ je enolično določen in ga navadno označimo z $m_\alpha(x)$. Dokažimo, da je za $\alpha \in F^*$ minimalni polinom nerazcepen. Denimo, da je $m_\alpha(x)$ razcepen. Potem za neka polinoma $h(x), \ell(x) \in \mathbb{F}_q[x]$ s stopnjama $\deg h(x) \geq 1$ in $\deg \ell(x) \geq 1$ velja

$$m_\alpha(x) = h(x)\ell(x).$$

Ker je $\alpha$ ničla svojega minimalnega polinoma, $m_\alpha(\alpha) = h(\alpha)\ell(\alpha) = 0$, mora biti $\alpha$ ničla vsaj enega od polinomov $h(x)$ in $\ell(x)$. Torej obstaja polinom nižje stopnje od minimalnega polinoma, ki ima $\alpha$ za ničlo. To pa je v protislovju z definicijo minimalnega polinoma.

Pokažimo še, da je minimalni polinom elementa $\alpha \in \mathbb{F}_{q^m}$ glede na obseg $\mathbb{F}_q$, polinom iz $\mathbb{F}_q[x]$. Potrebovali bomo naslednjo definicijo.

**Definicija 1.3.5.** *Naj bo $\alpha \in \mathbb{F}_{q^m}$. S $t$ označimo najmanjše pozitivno število, za katerega velja $\alpha^{q^t} = \alpha$. Ker je $\alpha$ element končnega obsega, tako število zagotovo obstaja. Množica* **konjugirank** *elementa $\alpha$ je naslednja množica:*

$$C(\alpha) = \{\alpha, \alpha^q, \alpha^{q^2}, \ldots, \alpha^{q^{t-1}}\}.$$

Očitno v obsegu s karakteristiko $p$ za vsak $i$ velja $C(\alpha) = C(\alpha^{p^i})$. Sedaj lahko podamo trditev o minimalnem polinomu elementa $\alpha \in \mathbb{F}_{q^m}^*$.

**Trditev 1.3.6.** *Naj bo $\mathbb{F}_{q^m}$ končen obseg s karakteristiko $p$, $\alpha \in \mathbb{F}_{q^m}^*$ in naj bo $C(\alpha)$ množica konjugirank za $\alpha$ glede na obseg $\mathbb{F}_q$. Potem je*

$$m(x) = \prod_{\beta \in C(\alpha)} (x - \beta)$$

*polinom s koeficienti iz obsega $\mathbb{F}_q$.*

**Dokaz.** Naj bo $m(x) = \sum_{i=0}^{t} m_i x^i$. Koeficienti $m_i$ so iz obsega $\mathbb{F}_{q^m}$. Radi bi pokazali, da so v resnici iz podobsega $\mathbb{F}_q$. Velja enakost

$$\{\beta : \beta \in C(\alpha)\} = \{\beta^q : \beta \in C(\alpha)\}.$$

Izračunajmo

$$\begin{aligned} m(x)^q &= \prod_{\beta\in C(\alpha)} (x-\beta)^q = \prod_{\beta\in C(\alpha)} (x^q-\beta^q) = \\ &= \prod_{\beta\in C(\alpha)} (x^q-\beta) = m(x^q) = \sum_{i=0}^{t} m_i x^{iq}. \end{aligned}$$

Pri drugem enačaju smo uporabili trditev 1.3.3, tretji pa sledi iz zgornje enakosti. Po drugi strani pa velja

$$m(x)^q = \sum_{i=0}^{t} (m_i x^i)^q = \sum_{i=0}^{t} m_i^q x^{iq}.$$

Torej velja $m_i = m_i^q$. Po trditvi 1.3.1 to pomeni, da je $m_i \in \mathbb{F}_q$ za vsak $0 \le i \le t$, kar smo želeli dokazati. □

Naj bo $\alpha \in \mathbb{F}_{q^m}^*$ in $m_\alpha(x) = \sum_{i=0}^{t} m_i x^i$ njegov minimalni polinom. Potem so njegovi koeficienti iz $\mathbb{F}_q$. Po definiciji minimalnega polinoma je $\alpha$ njegova ničla. Izračunajmo

$$\begin{aligned} m_\alpha(\alpha^q) &= \sum_{i=0}^{t} m_i \alpha^{qi} = \sum_{i=0}^{t} m_i^q \alpha^{qi} = \\ &= \left(\sum_{i=0}^{t} m_i \alpha^i\right)^q = (m_\alpha(\alpha))^q = 0. \end{aligned}$$

Torej je tudi $\alpha^q$ ničla minimalnega polinoma $m_\alpha(x)$. Od tod sledi, da so vsi elementi množice $C(\alpha)$ ničle polinoma $m_\alpha(x)$. Dokazali smo naslednjo trditev:

**Trditev 1.3.7.** *Za element $\alpha \in \mathbb{F}_{q^m}^*$ je minimalni polinom enak*

$$m_\alpha(x) = \prod_{\beta\in C(\alpha)} (x-\beta).$$

## NERAZCEPNI POLINOMI

Spoznali smo algebraične in končne razširitve obsegov. Videli smo, da si lahko obseg $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$ predstavljamo kot vektorski prostor dimenzije $m$ nad prostorom $\mathbb{F}_q$. Če je množica $\{\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\}$ baza tega prostora, lahko vsak element $\alpha$ iz obsega $\mathbb{F}_{q^m}$ predstavimo z elementi iz baze, tj. $\alpha = a_0\alpha_0 + a_1\alpha_1 + \cdots + a_{m-1}\alpha_{m-1}$, kjer so koeficienti $a_0, a_1, \ldots, a_{m-1}$ elementi obsega $\mathbb{F}_q$. Kot bomo videli, lahko elemente obsega predstavimo tudi drugače. Poglejmo najprej trditev in njeno posledico o obstoju nerazcepnih polinomov.

**Trditev 1.3.8.** *Naj bo $\mathbb{F}_q$ končen obseg in $\mathbb{F}_r$ njegova končna razširitev. Potem je $\mathbb{F}_r$ enostavna algebraična razširitev za $\mathbb{F}_q$ in vsak primitiven element za $\mathbb{F}_r$ lahko določa obseg $\mathbb{F}_r$ nad $\mathbb{F}_q$.*

**Dokaz.** Ker je $\mathbb{F}_r^*$ ciklična grupa, v $\mathbb{F}_r$ obstaja primitiven element $\theta$. Očitno je $\mathbb{F}_q(\theta) \subseteq \mathbb{F}_r$. Po drugi strani pa $\mathbb{F}_q(\theta)$ vsebuje element 0 in vse potence elementa $\theta$, torej tudi vse ostale elemente obsega $\mathbb{F}_r$. Od tod sledi $\mathbb{F}_r = \mathbb{F}_q(\theta)$. □

**Posledica 1.3.9.** *Naj bo $\mathbb{F}_q$ končen obseg in $\mathbb{F}_{q^m}$ njegova razširitev stopnje $m$, kjer je $m$ poljubno naravno število. Potem v $\mathbb{F}_q[x]$ obstaja nerazcepen polinom $f(x)$ stopnje $m$ in je $\mathbb{F}_{q^m} \cong \mathbb{F}_q[x]/f(x)$.*

**Dokaz.** Po zgornji trditvi za nek $\theta \in \mathbb{F}_{q^m}$ velja $\mathbb{F}_{q^m} = \mathbb{F}_q(\theta)$. Potem je minimalni polinom za $\theta$ nad $\mathbb{F}_q$ nerazcepen polinom v $\mathbb{F}_q[x]$ stopnje $m$. Preostanek trditve sledi iz izreka 1.2.7 in trditve 1.3.7, saj je $\mathbb{F}_q[x]/f(x)$ razpadni obseg polinoma $f(x)$ nad $\mathbb{F}_q$. □

Torej za poljuben obseg $\mathbb{F}_{q^m}$ nad obsegom $\mathbb{F}_q$ obstaja nerazcepen polinom stopnje $m$. V poglavju o bazah bomo videli, da lahko s posebnimi oblikami nerazcepnih polinomov (npr. trinomi in pentonomi) hitreje implementiramo aritmetiko nad končnimi obsegi. Če je $f(x)$ nerazcepen polinom stopnje $m$ nad obsegom $\mathbb{F}_q$, obstaja točno $q^m$ polinomov stopnje manjše od $m$ s koeficienti iz obsega $\mathbb{F}_q$. Torej lahko vsak od teh polinomov predstavlja en element iz obsega $\mathbb{F}_{q^m}$. Polinome seštevamo po členih in je tako vsota poljubnih dveh polinomov iz obsega $\mathbb{F}_q[x]/f(x)$ spet element obsega. Pri množenju dveh polinomov moramo paziti na stopnjo. Če ima produkt stopnjo večjo od $m$, ga moramo reducirati po modulu polinoma $f(x)$. S tema dvema operacijama je obseg $\mathbb{F}_q[x]/f(x)$ res izomorfen obsegu $\mathbb{F}_{q^m}$. Naredimo enostaven primer.

**Primer:**(Predstavitev obsega s polinomi) Naj bo končen obseg določen z nerazcepnimim polinomom $f(x) = x^4 + x + 1$. Potem sestavlja obseg $\mathbb{F}_{2^4}$ naslednjih 16 elementov:

| | | | | | |
|---|---|---|---|---|---|
| $0$ | (0000) | $1$ | (0001) | $x$ | (0010) |
| $x+1$ | (0011) | $x^2$ | (0100) | $x^2+1$ | (0101) |
| $x^2+x$ | (0110) | $x^2+x+1$ | (0111) | $x^3$ | (1000) |
| $x^3+1$ | (1001) | $x^3+x$ | (1010) | $x^3+x+1$ | (1011) |
| $x^3+x^2$ | (1100) | $x^3+x^2+1$ | (1101) | $x^3+x^2+x$ | (1110) |
| $x^3+x^2+x+1$ | (1111) | | | | |

Poglejmo nekaj primerov osnovnih operacij:

- $(1101) + (1001) = (0100)$.
- $(1101) \cdot (1001) = (1111)$, saj je $(x^3+x^2+1) \cdot (x^3+1) = x^6+x^5+x^2+1$ in $(x^6+x^5+x^2+1) \mod (x^4+x+1) = x^3+x^2+x+1$.
- $(1101)^{-1} = (0100)$.

$\Diamond$

Torej vsak nerazcepni polinom stopnje $m$ nad obsegom $\mathbb{F}_q$ določa obseg $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$. Kako pa je z obstojem nerazcepnih polinomov stopnje $m$? O tem govorita naslednja trditev in njena posledica.

## 1.4 SLED IN NORMA

Vzemimo obseg $\mathbb{F}_q$ s $q$ elementi in njegovo $m$-razsežno razširitev $\mathbb{F}_{q^m}$. Obseg $\mathbb{F}_{q^m}$ si predstavljajmo kot vektorski prostor nad $\mathbb{F}_q$. Definirali bomo dve pomembni preslikavi iz $\mathbb{F}_{q^m}$ na $\mathbb{F}_q$, sled in normo. Najprej se posvetimo sledi.

**Definicija 1.4.1.** *Naj bo $\alpha \in \mathbb{F}_{q^m} = F$ in $K = \mathbb{F}_q$.* **Sled** *za element $\alpha$ nad $K$ je preslikava z $F$ v $K$, definirana kot*

$$\mathrm{Tr}_{F/K}(\alpha) = \alpha + \alpha^q + \cdots + \alpha^{q^{m-1}}.$$

Če sta obsega $F$ in $K$ razvidna iz konteksta, lahko sled za element $\alpha$ označimo kar s $\mathrm{Tr}(\alpha)$.

**Trditev 1.4.2.** *Naj bo $F = \mathbb{F}_{q^m}$ in $K = \mathbb{F}_q$. Za $\mathrm{Tr} = \mathrm{Tr}_{F/K}$ veljajo naslednje lastnosti:*

*(i)* $\mathrm{Tr}(\alpha + \beta) = \mathrm{Tr}(\alpha) + \mathrm{Tr}(\beta)$ *za vse $\alpha, \beta \in F$;*

*(ii)* $\mathrm{Tr}(c\alpha) = c\mathrm{Tr}(\alpha)$ *za vsak $c \in K$ in $\alpha \in F$;*

*(iii)* $\mathrm{Tr}(\alpha^q) = \mathrm{Tr}(\alpha)$ *za vsak $\alpha \in F$;*

*(iv)* $\mathrm{Tr}$ *je linearna surjekcija iz $F$ v $K$, kjer obravnavamo $F$ kot vektorski prostor nad $K$;*

*(v)* $\mathrm{Tr}(a) = ma$ *za vsak $a \in K$.*

**Dokaz.**

(i) Naj bosta $\alpha, \beta \in F$ in izračunajmo

$$\begin{aligned} \mathrm{Tr}(\alpha+\beta) &= \alpha + \beta + (\alpha+\beta)^q + \cdots + (\alpha+\beta)^{q^{m-1}} \\ &= \alpha + \beta + \alpha^q + \beta^q + \cdots + \alpha^{q^{m-1}} + \beta^{q^{m-1}} \\ &= \mathrm{Tr}(\alpha) + \mathrm{Tr}(\beta), \end{aligned}$$

kar smo hoteli pokazati.

(ii) Za $c \in K$ velja $c^{q^j} = c$ za vse $j \geq 0$. Zato za vsak $\alpha \in F$ dobimo

$$\begin{aligned} \mathrm{Tr}(c\alpha) &= c\alpha + c^q\alpha^q + \cdots + c^{q^{m-1}}\alpha^{q^{m-1}} \\ &= c\alpha + c\alpha^q + \cdots + c\alpha^{q^{m-1}} \\ &= c\mathrm{Tr}(\alpha). \end{aligned}$$

(iii) Za $\alpha \in F$ velja $\alpha^{q^m} = \alpha$ in zato je $\mathrm{Tr}(\alpha^q) = \alpha^q + \alpha^{q^2} + \cdots + \alpha^{q^m} = \mathrm{Tr}(\alpha)$.

(iv) Velja $(\mathrm{Tr}(\alpha))^q = \mathrm{Tr}(\alpha)$, zato je $\mathrm{Tr}(\alpha) \in K$ za vsak $\alpha \in F$. Zaradi tega in trditev (i) in (ii) je Tr linearna preslikava iz $F$ v $K$. Končni obseg $K$ je 1-dimenzionalen. Da pokažemo, da je ta preslikava surjekcija, je torej dovolj najti tak $\alpha \in F$, za katerega je $\mathrm{Tr}(\alpha) \neq 0$. Poglejmo, za katere $\alpha \in F$ velja $\mathrm{Tr}(\alpha) = 0$. To je natanko takrat, ko je $\alpha$ ničla polinoma $x^{q^{m-1}} + \cdots + x^q + x \in K[x]$. Vendar ima lahko ta polinom največ $q^{m-1}$ ničel v $F$, $F$ pa vsebuje $q^m$ elementov. Torej obstaja $\alpha \in F$ z neničelno sledjo.

(v) Za $a \in K$ velja

$$\mathrm{Tr}(a) = a + a^q + \cdots + a^{q^{m-1}} = a + a + \cdots + a = ma.$$

V dokazu smo večkrat upoštevali trditev 1.3.1. □

Dokažimo še naslednjo trditev o zvezi med linearnimi transformacijami med obsegi in sledjo. Koristila nam bo v naslednjem poglavju, pri definiciji dualnih baz.

**Trditev 1.4.3.** *Naj bosta $F$ in $K$ definirana kot v prejšnji trditvi. Potem so linearne transformacije iz $F$ v $K$ natanko preslikave $L_\beta$ za $\beta \in F$, kjer je $L_\beta(\alpha) = Tr(\beta\alpha)$ za vsak $\alpha \in F$. Za različna elementa $\beta$ in $\gamma$ iz obsega $F$ sta različni tudi preslikavi $L_\beta$ in $L_\gamma$.*

**Dokaz.** Po četrti točki prejšnje trditve je vsaka preslikava $L_\beta$ linearna preslikava iz $F$ v $K$. Ker je preslikava Tr surjekcija iz $F$ v $K$, velja za različna elementa $\beta, \gamma \in F$ $L_\beta(\alpha) - L_\gamma(\alpha) = \mathrm{Tr}(\beta\alpha) - \mathrm{Tr}(\gamma\alpha) = \mathrm{Tr}((\beta - \gamma)\alpha) \neq 0$ za primeren $\alpha \in F$. Torej sta preslikavi $L_\beta$ in $L_\gamma$ različni. Ker šteje obseg $F = \mathbb{F}_{q^m}$ $q^m$ elementov, obstaja $q^m$ različnih preslikav iz $F$ v $K$ oblike $L_\beta$. Po drugi strani pa linearno transformacijo iz $F$ v $K$ dobimo tako, da množimo katerikoli od $q$ elementov obsega $K$ s katerimkoli od $m$ elementov dane baze za $F$ nad $K$. Torej obstaja $q^m$ različnih linearnih preslikav iz $F$ v $K$. Torej preslikave $L_\beta$ res tvorijo vse možne linearne transformacije iz $F$ v $K$. □

Če imamo več zaporednih razširitev obsegov, za sled velja nekakšna tranzitivnost. O tem govori naslednja trditev.

**Trditev 1.4.4. [Tranzitivnost sledi]** *Naj bo $K$ končen obseg, $F$ njegova končna razširitev in $E$ končna razširitev obsega $F$. Potem za vsak $\alpha \in E$ velja:*

$$\mathrm{Tr}_{E/K}(\alpha) = \mathrm{Tr}_{F/K}(\mathrm{Tr}_{E/F}(\alpha)).$$

**Dokaz.** Naj bo $K = \mathbb{F}_q$, $[F : K] = m$ in $[E : F] = n$, tako da je $[E : K] = mn$ (to smo že spoznali v drugem razdelku). Potem za $\alpha \in E$ velja:

$$\mathrm{Tr}_{F/K}(\mathrm{Tr}_{E/F}(\alpha)) = \sum_{i=0}^{m-1} \mathrm{Tr}_{E/F}(\alpha)^{q^i} = \sum_{i=0}^{m-1} \Big(\sum_{j=0}^{n-1} \alpha^{q^{jm}}\Big)^{q^i},$$

oziroma

$$\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} \alpha^{q^{jm+i}} = \sum_{k=0}^{mn-1} \alpha^{q^k} = \mathrm{Tr}_{E/K}(\alpha).$$

□

Norma je naslednja pomembna preslikava iz končnega obsega na podobseg in jo dobimo z množenjem vseh $q$-tih potenc elementa iz obsega.

**Definicija 1.4.5.** *Za $\alpha \in F = \mathbb{F}_{q^m}$ in $K = \mathbb{F}_q$ je* **norma** *$N_{F/K}(\alpha)$ za element $\alpha$ nad $K$ definirana kot*

$$N_{F/K}(\alpha) = \alpha \cdot \alpha^q \cdots \alpha^{q^{m-1}} = \alpha^{(q^m-1)/(q-1)}.$$

Spet lahko indekse izpustimo, če so obsegi razvidni iz konteksta in normo elementa $\alpha$ označimo z $N(\alpha)$. Naslednja trditev nam bo podala nekaj lepih lastnosti norme.

**Trditev 1.4.6.** *Naj bo $K = \mathbb{F}_q$ in $F = \mathbb{F}_{q^m}$. Potem norma $N = N_{F/K}$ zadošča naslednjim lastnostim:*

*(i) $N(\alpha\beta) = N(\alpha)N(\beta)$, za vse $\alpha, \beta \in F$;*

*(ii) $N$ je surjekcija iz $F$ v $K$ in iz $F^*$ v $K^*$;*

*(iii) $N(a) = a^m$ za vsak $a \in K$;*

*(iv) $N(\alpha^q) = N(\alpha)$ za vsak $\alpha \in F$.*

**Dokaz.** (i) sledi direktno iz definicije norme (ravnamo podobno kot pri prvi lastnosti sledi). Opazili smo že, da $N$ preslika $F$ v $K$. Ker je $N(\alpha) = 0$ natanko tedaj, ko je $\alpha = 0$, preslika $N$ $F^*$ v $K^*$. Lastnost (i) pove, da je $N$ homomorfizem grup med tema multiplikativnima grupama. Ker so elementi jedra za $N$ natanko ničle polinoma $x^{(q^m-1)/(q-1)} - 1 \in K[x]$ v $F$, velja za red jedra $d$ naslednja neenakost: $d \leq (q^m - 1)/(q - 1)$. Slika preslikave $N$ ima red $(q^{m-1} - 1)/d$, kar je večje od $q - 1$.

Zato je $N$ surjektivna preslikava iz $F^*$ v $K^*$ in tudi iz $F$ v $K$. Lastnost (iii) sledi iz definicije norme in ker za $a \in K$ velja $a^q = a$. Velja tudi $N(\alpha^q) = N(\alpha)^q = N(\alpha)$ (zaradi (i) in $N(\alpha) \in K$). S tem smo pokazali še lastnost (iv). □

Tudi za funkcijo norme velja nekakšna tranzitivnost, o čemer govori naslednja trditev.

**Trditev 1.4.7.** [**Tranzitivnost norme**] *Naj bo $K$ končen obseg, $F$ končna razširitev za $K$ in $E$ končna razširitev za obseg $F$. Potem za vsak $\alpha \in E$ velja:*

$$N_{E/K}(\alpha) = N_{F/K}(N_{E/F}(\alpha)).$$

**Dokaz.** Ravnali bomo podobno kot pri dokazu tranzitivnosti sledi. Za vsak $\alpha \in E$ imamo

$$\begin{aligned} N_{F/K}(N_{E/F}(\alpha)) &= N_{F/K}(\alpha^{(q^{mn}-1)/(q^m-1)}) \\ &= (\alpha^{(q^{mn}-1)/(q^m-1)})^{(q^m-1)/(q-1)} \\ &= \alpha^{(q^{mn}-1)/(q-1)} \\ &= N_{E/K}(\alpha). \end{aligned}$$

□

Naj bo $f \in \mathbb{F}_q[x]$ minimalni polinom za element $\alpha \in \mathbb{F}_q$. Po definiciji minimalnega polinoma je potem $f(\alpha) = 0$ in stopnja polinoma $f$ minimalna. Potem je tudi $f(\alpha^q) = 0$ in vse ostale konjugiranke elementa $\alpha$ so tudi ničle za $f$. Zapišimo polinom $f_\alpha(x)$ za $\alpha \in \mathbb{F}_{q^m}$, definiran takole:

$$f_\alpha(x) = (x-\alpha)(x-\alpha^q)\dots(x-\alpha^{q^{m-1}}) = \sum_{i=1}^{m} f_i x^i,$$

ki je minimalni polinom za $\alpha$ nad $\mathbb{F}_q$ ali njegova potenca. Poglejmo koeficiente pri različnih potencah $x$. Pri $x^m$ je koeficient enak $f_m = 1$. Pri $x^{m-1}$ dobimo $f_{m-1} = -\sum_{i=0}^{m-1} \alpha^{q^i} = -\mathrm{Tr}(\alpha)$, pri konstantnem členu pa vidimo, da je $f_0 = N(\alpha)(-1)^m$. Oziroma, če zapišemo drugače, smo dobili $\mathrm{Tr}(\alpha) = -f_{m-1}$ in $N(\alpha) = (-1)^m f_0$.

# Poglavje 2

# BAZE KONČNIH OBSEGOV

Z razvojem kriptografije in številnih kriptosistemov, ki uporabljajo končne obsege, se je pojavila potreba po izboljšavi algotitmov za aritmetiko nad končnimi obsegi. Za računanje z elementi iz obsega moramo te elemente na nek način predstaviti v računalniku. Kot smo videli v razdelku 1.3, ima vsak končen obseg $F$, ki je razširitev podobsega $\mathbb{F}_q$ stopnje $m$, $q^m$ elementov. Označimo ga s $\mathbb{F}_{q^m}$ in ga obravnavamo kot vektorski prostor nad obsegom $\mathbb{F}_q$. Obseg $\mathbb{F}_{q^m}$ navadno predstavimo z določeno bazo vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$. Vsaka takšna baza ima natanko $m$ elementov. Torej poljuben element obsega predstavimo z $m$-terico koeficientov iz obsega po izbrani bazi. Na voljo imamo več baz. S predstavitvijo elementov v določeni bazi lahko algoritme precej izboljšamo. Vsaka baza ima svoje prednosti in slabosti. Izkazalo se je, da imajo med danes poznanimi bazami za potrebe kriptografije in kodiranja najlepše lastnosti tri vrste baz. To so polinomske, normalne in sebidualne baze. Pri polinomskih se posebej odlikujejo baze, ko za redukcijski polinom vzamemo trinom ali pentonom. Obstaja tudi posebna vrsta normalnih baz, to so optimalne normalne baze. Iz njih izpeljemo umbralne baze, ki jih lahko posebej lepo implementiramo. Žal je prehod med bazami kar zamuden problem v primerjavi s časovno zahtevnostjo algoritmov za osnovne operacije z elementi iz končnega obsega.
Torej vsak element končnega obsega predstavimo kot urejeno $m$-terico koeficientov iz množice $\{0, 1, 2, \ldots, q-1\}$. Če je $q$ majhen, s tem ni težav. Če pa sta $q$ ali $m$ veliki števili, lahko samo shranjevanje postane velik problem. V kriptografiji in teoriji kodiranja najpogosteje delamo z naslednjimi obsegi:

- obsegi oblike $\mathbb{F}_{2^m}$, kjer je $m$ veliko naravno število;
- obsegi oblike $\mathbb{F}_p$, kjer je $p$ veliko praštevilo.

V tem delu se bomo ukvarjali samo s prvim primerom, torej z obsegi $\mathbb{F}_{2^m}$ nad obsegom $\mathbb{F}_2$.

V prvem razdelku bomo pogledali osnove baz končnih obsegov, predvsem obstoj določenih baz, dualne baze in število različnih baz. Nato bomo podrobneje obdelali polinomske baze, posebej tiste, v katerih je redukcijski polinom trinom ali pentonom. V naslednjem razdelku bomo govorili o normalnih bazah. Posebna razdelka smo posvetili optimalnim normalnim bazam in umbralnim bazam. Sledijo osnove sebidualnih baz. Poglavje zaključimo s problemom prehoda med bazami.

## 2.1 OSNOVE

V tem uvodnem poglavju bomo predstavili nekaj osnovnih značilnosti baz za končne obsege. Predvsem nas zanima obstoj določenih tipov baz, njihovi duali in število takih baz.
Število različnih urejenih baz vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$ je

$$(q^m-1)(q^m-q)(q^m-q^2)\cdots(q^m-q^{m-1}) = q^{(m-1)m/2}\prod_{i=1}^{m}(q^i-1),$$

kar je tudi red grupe $\mathrm{GL}(m,q)$ vseh obrnljivih matrik velikosti $m \times m$ nad $\mathbb{F}_q$.

**Definicija 2.1.1.** *Naj bosta $\bar{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_m\}$ in $\bar{\beta} = \{\beta_1, \beta_2, ..., \beta_m\}$ bazi nekega vektorskega prostora. Rekli bomo, da sta $\bar{\alpha}$ in $\bar{\beta}$* **dualni**, *če za $1 \le i, j \le m$ velja*

$$\mathrm{Tr}(\alpha_i\beta_j) = \delta_{ij},$$

*kjer $\delta_{ij}$ pomeni Kroneckerjev delta.*

Pokažimo, da za poljubno bazo obstaja njena dualna baza.

**Trditev 2.1.2.** *Za poljubno bazo $\{\alpha_1, \alpha_2, ..., \alpha_m\}$ vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$ obstaja dualna baza $\{\beta_1, \beta_2, ..., \beta_m\}$, ki je enolično določena.*

**Dokaz.** Naj bo $\{\alpha_1, \alpha_2, ..., \alpha_m\}$ baza vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$ in $\alpha \in \mathbb{F}_{q^m}$. Potem obstajajo skalarji $c_1(\alpha), c_2(\alpha), \ldots, c_m(\alpha) \in \mathbb{F}_q$, da je

$$\alpha = c_1(\alpha)\alpha_1 + ... + c_m(\alpha)\alpha_m. \tag{2.1}$$

Preslikava $c_j$, definirana z $\alpha \mapsto c_j(\alpha)$ je linearna transformacija iz $\mathbb{F}_{q^m}$ v $\mathbb{F}_q$. Po trditvi 1.4.3 obstaja tak $\beta_i \in \mathbb{F}_{q^m}$, da za $1 \le i \le m$ velja

$$c_i(\alpha) = \mathrm{Tr}(\beta_i\alpha).$$

To pomeni

$$\alpha = \sum_{i=1}^{m} \mathrm{Tr}(\beta_i\alpha)\alpha_i$$

za vsak $\alpha \in \mathbb{F}_{q^m}$. Za $\alpha = \alpha_j$ velja

$$\alpha_j = \sum_{i=1}^{m} \mathrm{Tr}(\beta_i \alpha_j)\alpha_i.$$

Torej je

$$\mathrm{Tr}(\alpha_i \beta_j) = \begin{cases} 0, & \text{če } i \neq j, \\ 1, & \text{če } i = j. \end{cases}$$

Iz enačbe $d_1\beta_1 + \cdots + d_m\beta_m = 0$, kjer so $d_1, d_2, \ldots, d_m \in \mathbb{F}_q$, za vsak $j \in \{1, 2, \ldots, m\}$ sledi

$$\big(\sum_i d_i\beta_i\big)\alpha_j = 0 \;\text{ oziroma }\; \mathrm{Tr}\big(\sum_i d_i\beta_i\alpha_j\big) = 0.$$

Zaradi linearnosti Tr dobimo še $\sum_i d_i \mathrm{Tr}(\beta_\mathrm{i}\alpha_\mathrm{j}) = 0$. Torej za vsak $j = 1, 2, ..., m$ velja $d_j = 0$. Množica $\{\beta_1, \beta_2, ..., \beta_m\}$ je tudi baza vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$. Prepričajmo se še, da je dualna baza enolično določena. Koeficienti $c_1(\alpha), c_2(\alpha), \ldots, c_m(\alpha)$ iz enakosti (2.1) so za vsak $1 \leq j \leq m$ in za vsak $\alpha \in \mathbb{F}_{q^m}$ podani s formulo $c_j(\alpha) = \mathrm{Tr}(\beta_j\alpha)$. Elementi $\beta_j \in \mathbb{F}_{q^m}$ pa so po trditvi 1.4.3 enolično določeni z linearno transformacijo $c_j$. □

## 2.2 POLINOMSKE BAZE

Za kriptografske namene se najpogosteje uporabljajo polinomske baze. Obstajajo za vsako karakteristiko $p$ in za vsak obseg $\mathbb{F}_{q^m}$. Mi se bomo ukvarjali samo s primerom $q = 2$. Polinomsko bazo imenujemo tudi standardna ali kanonična baza.

Najprej bomo podali splošno definicijo polinomske baze. V naslednjem razdelku bomo dokazali, da polinomske baze res obstajajo za vsak končen obseg. Razdelek bomo zaključili z osnovnimi lastnostmi trinomov in pentonomov.

### OBLIKA

**Trditev 2.2.1.** *Naj bo $\alpha$ neka ničla nekega nerazcepnega polinoma $f$ stopnje $m$ iz $\mathbb{F}_q[x]$. Množica $P = \{1, \alpha, \alpha^2, ..., \alpha^{m-1}\}$ je baza vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$.*

**Dokaz.** Preveriti moramo, da so elementi množice $P$ linearno neodvisni. Minimalni polinom elementa $\alpha$ mora deliti nerazcepni polinom $f$, to pa je možno le, če je $f$ enak minimalnemu polinomu. Torej $\alpha$ ni ničla nobenega netrivialnega polinoma iz $\mathbb{F}_q[x]$, ki ima stopnjo manjšo od $m$. Zato so elementi iz množice $P$ linearno neodvisni. Ker je $|P| = m$, je množica $P$ res baza. □

Sedaj lahko definiramo polinomsko bazo.

**Definicija 2.2.2.** *Podmnožica vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$ je* **polinomska baza**, *če je oblike $\{1, \alpha, \alpha^2, ..., \alpha^{m-1}\}$, kjer je $\alpha$ neka ničla nekega nerazcepnega polinoma $f$ stopnje $m$ iz $\mathbb{F}_q[x]$.*

Torej nam vsak nerazcepen polinom stopnje $m$ iz $\mathbb{F}_q[x]$ za vsako svojo ničlo določa polinomsko bazo vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$. Za različne nerazcepne polinome dobimo različne baze za isti vektorski prostor. Za boljšo predstavo si poglejmo enostaven primer.

**Primer:** (Polinomska baza) Spet vzemimo končen obseg $\mathbb{F}_{2^4}$, določen z redukcijskim polinomom $f(x) = x^4 + x + 1$.

Element $\alpha = x = (0010)$ je generator obsega $\mathbb{F}_{2^4}^*$, saj je njegov red enak 15:

$$\begin{aligned}
\alpha^1 &= \alpha = (0010) & \alpha^2 &= (0100) & \alpha^3 &= (1000)\\
\alpha^4 &= \alpha + 1 = (0011) & \alpha^5 &= \alpha^2 + \alpha = (0110) & \alpha^6 &= \alpha^3 + \alpha^2 = (1100)\\
\alpha^7 &= \alpha^3 + \alpha + 1 = (1011) & \alpha^8 &= \alpha^2 + 1 = (0101) & \alpha^9 &= \alpha^3 + \alpha = (1010)\\
\alpha^{10} &= \alpha^2 + \alpha + 1 = (0111) & \alpha^{11} &= (1110) & \alpha^{12} &= (1111)\\
\alpha^{13} &= (1101) & \alpha^{14} &= \alpha^3 + 1 = (1001) & \alpha^{15} &= 1 = (0001)\\
\alpha^{16} &= \alpha = (0010).
\end{aligned}$$

Polinomska baza obsega $\mathbb{F}_{2^4}$ nad $\mathbb{F}_2$ je enaka $\{1, \alpha, \alpha^2, \alpha^3\}$. ◊

V prvem poglavju smo videli, da za vsako naravno število $m$ obstaja nerazcepen polinom nad obsegom $\mathbb{F}_q$ stopnje $m$. Torej za vsak končen obseg $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$ obstaja polinomska baza. Aritmetiko z elementi, predstavljenimi v polinomski bazi, si bomo podrobneje pogledali v četrtem poglavju. Aritmetiko v končnih obsegih lahko učinkoviteje implementiramo, če ima izbrani nerazcepni polinom malo neničelnih členov. Zato bomo poseben poudarek naredili na nerazcepnih *trinomih* in *pentonomih*.

## NERAZCEPNI TRINOMI IN PENTONOMI

Da bi poenostavili aritmetiko množenja s polinomi, moramo poleg hitrega množenja polinomov s koeficienti iz $\mathbb{F}_q$ znati učinkovito izvesti redukcijo danega polinoma po modulu polinoma $f$. Dobro je, če je oblika polinoma $f$ čim enostavnejša, oziroma, da je neničelnih členov malo, saj stopnje polinoma ne moremo zmanjšati. Zato so še posebej prikladni nerazcepni trinomi in pentonomi.

Če ima polinom $f(x)$ v $\mathbb{F}_q[x]$ sodo število neničelnih členov, potem je $f(1) = 0$, od tod pa sledi, da je $(x + 1)$ njegov faktor. Torej mora imeti nerazcepen polinom stopnje $\geq 2$ v $\mathbb{F}_q[x]$ najmanj tri neničelne člene. Konstanten člen mora biti enak 1, saj je sicer polinom deljiv z $x$. Nerazcepen **trinom** stopnje $m$ v $\mathbb{Z}_q[x]$ mora biti oblike $x^m + x^k + 1$, kjer je $1 \geq k \geq m - 1$. Z izbiro določenega trinoma $f(x) \in \mathbb{F}_q[x]$

stopnje $m$, ki nam predstavlja elemente končnega obsega $\mathbb{F}_{q^m} = \mathbb{F}_q[x]/(f(x))$, lahko pripeljemo do hitrejše implementacije aritmetike v obsegu. Če je trinom $x^m + x^k + 1$ nerazcepen nad $\mathbb{F}_q$, potem je nerazcepen tudi $x^m + x^{m-k} + 1$.

Za aritmetiko nad obsegom $\mathbb{F}_q$ so torej najpreprostejši trinomi, vendar na žalost za nekatere $m$ v kolobarju polinomov $\mathbb{F}_q[x]$ ni nerazcepnih polinomov stopnje $m$. Empirični rezultati kažejo, da je kar polovica vseh naravnih števil $m$ takih, s teoretičnega vidika pa vprašanje obstoja trinomov še ni razjasnjeno. Kadar za neko število $m$ ne obstaja nerazcepni trinom stopnje $m$, si lahko pomagamo z nerazcepnim pentonomom. Nerazcepen pentonom stopnje $m$ iz kolobarja $\mathbb{F}_q[x]$ je polinom z natanko petimi neničelnimi členi, od katerih je konstanten člen vedno enak 1. Teoretično še ni dokazan obstoj nerazcepnih pentonomov. Empirični rezultati so pokazali, da za $m < 1000$ v primeru, da ne obstaja nerazcepni trinom stopnje $m$, vedno obstaja nerazcepni pentonom stopnje $m$. To pa zaenkrat zadostuje za praktične potrebe.

Standard ANSI X9.62 določa naslednja pravila pri izbiri nerazcepnega polinoma za predstavitev elementov obsega $\mathbb{F}_{q^m}$:

1. Če obstaja nerazcepen trinom stopnje $m$ nad $\mathbb{F}_q$, potem naj bo redukcijski polinom $f$ nerazcepen trinom stopnje $m$ nad $\mathbb{F}_q$. Za enostavnejšo implementacijo algoritmov za osnovne operacije ANSI X9.62 predlaga, da naj bo trinom oblike $f(x) = x^m + x^k + 1$, za najmanjši možni $k$.

2. Če ne obstaja nerazcepni trinom stopnje $m$ nad $\mathbb{F}_q$, naj bo redukcijski polinom $f$ nerazcepen pentonom stopnje $m$ nad $\mathbb{F}_q$. Spet pa v korist enostavnejše implementacije algoritmov standard ANSI X9.62 predlaga, da je pentonom oblike $f(x) = x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, kjer so $k_3$, $k_2$ in $k_1$ najmanjši možni.

## 2.3 NORMALNE BAZE

Naslednja pomembna skupina baz so normalne baze. Videli bomo, da imajo nekaj velikih prednosti v primerjavi s polinomskimi bazami, pa tudi nekaj slabosti. Najprej podajmo osnovne definicije normalnih baz in nekaj lastnosti, nato pa izrek o normalni bazi. Podpoglavje bomo zaključili z zanimivo lasnostjo normalnih baz in njihovih dualnih baz.

Definirajmo pojem normalne baze in normalnega elementa.

**Definicija 2.3.1.** *Bazi vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$, ki je oblike $N = \{\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}\}$, kjer je $\alpha \in \mathbb{F}_{q^m}$, pravimo* **normalna baza**. *Ker je dimenzija prostora $m$, mora baza vsebovati $m$ elementov, oziroma, elementi $\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}$ morajo biti linearno neodvisni nad $\mathbb{F}_q$. Elementu $\alpha$ rečemo generator normalne baze $N$ oziroma*

**normalen element** *obsega* $\mathbb{F}_{q^m}$ *nad* $\mathbb{F}_q$. *Normalno bazo navadno zapišemo kot* $\{\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\}$, *kjer je* $\alpha_i = \alpha^{q^i}$ *za* $i = 0, 1, \ldots, m-1$.

Naj bo $N = \{\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\}$ normalna baza prostora $\mathbb{F}_{q^m}$, kjer je $\alpha \in \mathbb{F}_{q^m}$ in $\alpha_i = \alpha^{q^i}$. Za poljubna indeksa $i, j,\ 0 \leq i, j \leq m-1$, je produkt $\alpha_i\alpha_j$ linearna kombinacija elementov $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ s koeficienti iz $\mathbb{F}_q$. Ker velja $\alpha_i\alpha_j = (\alpha\alpha_{i-j})^{q^i}$, je dovolj če poznamo $\alpha\alpha_i$. Za $\alpha = \alpha_0$ lahko zapišemo

$$\alpha_0 \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{m-1} \end{pmatrix} = T \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{m-1} \end{pmatrix},$$

kjer je $T$ matrika velikosti $m \times m$ nad $\mathbb{F}_q$. Sedaj samo omenimo, da matriko $T$ imenujemo **multiplikacijska tabela** za normalno bazo $N$, oziroma multiplikacijska tabela za $\alpha$. O tem bomo podrobneje govorili v tretjem poglavju pri množenju v normalnih bazah.

Izkaže se, da je število normalnih elementov in s tem tudi normalnih baz precej veliko. Podrobno se z normalnimi bazami in normalnimi elementi ukvarja knjiga [4]. Tam tudi najdemo naslednji izrek in njegov dokaz (str. 79).

**Izrek 2.3.2. [IZREK O NORMALNI BAZI]** *Za vsako naravno število* $m$ *in vsako praštevilo* $p$, *da je* $q = p^n$ *za nek* $n \in \mathbb{N}$, *obstaja normalna baza obsega* $\mathbb{F}_{q^m}$ *nad* $\mathbb{F}_q$. □

Poglejmo si nekaj lastnosti, ki držijo za poljubno normalno bazo. Naj bo $N = \{\alpha_0, \alpha_1, ..., \alpha_{m-1}\}$ normalna baza vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$, kjer je $\alpha_i = \alpha^{q^i}$. Za $0 \geq i \geq m-1$ naj bo

$$\alpha\alpha_i = \sum_{j=0}^{m-1} t_{ij}\alpha_j, \tag{2.2}$$

kjer so $t_{ij} \in \mathbb{F}_q$ in $T = (t_{ij})$. Dualna baza za normalno bazo je tudi normalna baza. Naj bo $B = \{\beta_0, \beta_1, ..., \beta_{m-1}\}$ dualna baza za bazo $N$, kjer je za $0 \geq i \geq m-1$ $\beta_i = \beta^{q^i}$. Za $0 \geq i, j \geq m-1$ velja

$$\alpha\beta_i = \sum_{j=0}^{m-1} d_{ij}\beta_j\ ,$$

kjer je $d_{ij} \in \mathbb{F}_q$. Pokažimo, da za vse $0 \geq i, j \geq m-1$ velja

$$d_{ij} = t_{ji}, \tag{2.3}$$

to je, matrika $D = (d_{ij})$ je transponiranka matrike $T = (t_{ij})$. Po definiciji dualne baze velja

$$\mathrm{Tr}(\alpha_i\beta_j) = \begin{cases} 0, & \text{če } i \neq j, \\ 1, & \text{če } i = j. \end{cases}$$

Poglejmo vrednost $\mathrm{Tr}(\alpha\beta_i\alpha_k)$. Po eni strani velja

$$\mathrm{Tr}(\alpha\beta_i\alpha_k) = \mathrm{Tr}((\alpha\beta_i)\alpha_k) = \mathrm{Tr}\left(\sum_{j=0}^{m-1} d_{ij}\beta_j\alpha_k\right) = \sum_{j=0}^{m-1} d_{ij}\mathrm{Tr}(\beta_j\alpha_k) = d_{ik},$$

po drugi pa

$$\mathrm{Tr}(\alpha\beta_i\alpha_k) = \mathrm{Tr}((\alpha\alpha_k)\beta_i) = \mathrm{Tr}\left(\sum_{j=0}^{m-1} t_{kj}\alpha_j\beta_i\right) = \sum_{j=0}^{m-1} t_{kj}\mathrm{Tr}(\alpha_j\beta_i) = t_{ki}.$$

S tem smo dokazali enačbo (2.3). Matrika, ki pripada dualni bazi normalne baze, je res transponiranka matrike, ki pripada originalni normalni bazi.

## 2.4 OPTIMALNE NORMALNE BAZE

Videli bomo, da se z normalnimi bazami v primerjavi s polinomskimi bazami izredno poenostavi kvadriranje. Žal pa je množenje precej zamudno. Pri iskanju čim hitrejših algoritmov za množenje se je izkazalo, da se precej poenostavijo, če uporabimo določeno vrsto normalnih baz, to so optimalne normalne baze.

V prvem razdelku bomo opisali nekatere osnovne definicije in lastnosti optimalnih normalnih baz, v drugem razdelku podali konstrukcijo optimalnih normalnih baz in v tretjem določili vse optimalne normalne baze. Če pri določeni optimalni normalni bazi permutiramo vrstni red elementov, dobimo posebno vrsto normalnih baz, ki ji pravimo umbralne baze. Z njimi zaključujemo razdelek.

### OSNOVE

Naj bo $N = \{\alpha_0, \alpha_1, ..., \alpha_{m-1}\}$ normalna baza vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$, kjer je $\alpha_i = \alpha^{q^i}$. Za vsako naravno število $k$ velja $\alpha_i^{q^k} = \alpha_{i+k}$, kjer reduciramo indekse za $\alpha$ po modulu $m$. Potem lahko vsak element $A \in \mathbb{F}_{q^m}$ predstavimo kot $A = \sum_{i=0}^{n-1} a_i\alpha_i,\ a_i \in \mathbb{F}_q$. Za $0 \leq i \leq n-1$ naj bo

$$\alpha\alpha_i = \sum_{j=0}^{m-1} t_{ij}\alpha_j, \quad t_{ij} \in \mathbb{F}_q.$$

S $T$ označimo matriko $(t_{ij})$, dimenzije $m \times m$. Matriki $T$ pravimo **multiplikacijska tabela** za normalno bazo $N$. Število neničelnih elementov v matriki $T$ imenujemo **kompleksnost** normalne baze $N$ in jo označimo s $C_N$. Z uporabo normalnih baz z nizko kompleksnostjo lahko precej pospešimo računanje produkta dveh elementov. Naslednja trditev nam podaja spodnjo mejo za $C_N$.

**Trditev 2.4.1.** *Za poljubno normalno bazo $N$ vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$ velja $C_N \geq 2m - 1$.*

**Dokaz.** Naj bo $N = \{\alpha_0, \alpha_1, ..., \alpha_{m-1}\}$ normalna baza Za $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$. Potem je $b = \sum_{k=0}^{m-1} \alpha_k = \mathrm{Tr}(\alpha) \in \mathbb{F}_q$. Če upostevamo enačbe (2.2) in primerjamo koeficiente za $\alpha_k$, dobimo

$$\sum_{i=0}^{m-1} t_{ij} = \begin{cases} b, & j = 0, \\ 0, & 1 \leq j \leq m-1. \end{cases}$$

Ker je $\alpha$ različen od nič in ker je tudi $\{\alpha\alpha_i : 0 \leq i \leq m-1\}$ baza vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$, je matrika $T = (t_{ij})$ obrnljiva. Zato za vsak $j$ obstaja vsaj en neničeln $t_{ij}$. Ker mora biti vsota elementov vsakega stolpca nič, morata biti za vsak $j \neq 0$ vsaj dva neničelna elementa $t_{ij}$. Torej je vsaj $2m - 1$ neničelnih elementov v $T$. □

**Definicija 2.4.2.** *Normalni bazi $N$ obsega $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$ rečemo* **optimalna normalna baza***, če zanjo velja $C_N = 2m - 1$.*

Mullin, Onyszchuk, Vanstone in Wilson so podali dve konstrukciji optimalnih normalnih baz, kasneje pa sta Gao in Lenstra dokazala, da sta to dejansko edini dve obliki optimalnih normalnih baz za končne obsege. Zapišimo ti dve konstrukciji v trditvi, katere dokaz najdemo v [4, str. 96].

**Trditev 2.4.3.** *Končen obseg $\mathbb{F}_{q^m}$, kjerje $q$ praštevilo ali potenca praštevila, ima optimalno normalno bazo natanko tedaj, ko drži ena od naslednjih dveh trditev:*

*(i) Naj bo $m + 1$ praštevilo in $q$ primitiven element v $\mathbb{Z}_{m+1}$. Potem je $m$ od enote različnih $(m + 1)$-vih korenov enote linearno neodvisnih in tvorijo optimalno normalno bazo vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$.*

*(ii) Naj bo $2m+1$ praštevilo in naj bo množica $\mathbb{Z}^*_{2m+1}$ generirana z $2$ in $-1$. Potem $\beta = \gamma + \gamma^{-1}$ generira optimalno normalno bazo vekatorskega prostora $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$, kjer je $\gamma$ primitiven $(2m + 1)$-vi koren enote.* □

Optimalni normalni bazi, ki ustreza prvi točki trditve pravimo optimalna normalna baza tipa I, drugi pa optimalna normalna baza tipa II. Več o tem bomo spoznali v naslednjem razdelku.

## KONSTRUKCIJA

Privzemimo pogoje trditve 3.4.2 (i) (ONB tipa I). Naj bo $\alpha$ primitiven $(m+1)$-vi koren enote. Potem je $\alpha$ ničla polinoma $x^m + \cdots + x + 1$. Ker je $m+1$ praštevilo, $m+1$ deli $q^m - 1$ in vsi $(m+1)$-vi koreni enote so elementi obsega $\mathbb{F}_{q^m}$. Ker je $q$ primitiven v $\mathbb{Z}_{m+1}$, obstaja $m$ različnih konjugirank elementa $\alpha$, ki so tudi od enote različni $(m+1)$-vi koreni enote. To pomeni

$$N = \{\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}\} = \{\alpha, \alpha^2, \ldots, \alpha^m\}.$$

Torej je $N$ normalna baza vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$. Za $1 \leq i \leq m$ velja

$$\alpha\alpha^i = \alpha^{i+1} \in N$$

in

$$\alpha\alpha^m = 1 = -\mathrm{Tr}(\alpha) = -\sum_{i=1}^{m} \alpha^i.$$

Zato je vseh neničelnih vrednosti v mešanih produktih natanko $2m-1$ in je normalna baza $N$ optimalna. Matrika $T$, ki pripada tej bazi, ima naslednje lastnosti: v vsaki vrstici je natanko ena enica, razen v eni vrstici, kjer so vsi elementi enaki 1. Vsi ostali elementi v matriki so enaki 0. Poglejmo si ONB tipa I na primeru.

**Primer:** Vzemimo obseg $\mathbb{F}_{2^4}$. Število $m+1 = 5$ je praštevilo in 2 je primitiven element v $\mathbb{Z}_5$. Potem obstaja po trditvi 2.4.3 v obsegu $\mathbb{F}_{2^4}$ optimalna baza. Element $\alpha$ naj bo ničla polinoma $f(x) = x^4 + x^3 + x^2 + x + 1$ oziroma primitiven peti koren enote. Množica $N = \{\alpha, \alpha^2, \alpha^4, \alpha^8\}$ je normalna baza vektorskega prostora $\mathbb{F}_{2^4}$ nad $\mathbb{F}_2$. Preverimo.

$$\begin{aligned} \alpha &= (0010) & \alpha^2 &= (0100) \\ \alpha^8 = \alpha^3 &= (1000) & \alpha^4 = \alpha^3 + \alpha^2 + \alpha + 1 &= (1111) \end{aligned}$$

Elementi $\alpha, \alpha^2, \alpha^4$ in $\alpha^8$ so res linearno neodvisni.

$$\begin{aligned} f(\alpha^2) &= \alpha^8 + \alpha^6 + \alpha^4 + \alpha^2 + 1 = \alpha^3 + \alpha + \alpha^3 + \alpha^2 + \alpha + 1 + \alpha^2 + 1 = 0 \\ f(\alpha^4) &= \alpha^{16} + \alpha^{12} + \alpha^8 + \alpha^4 + 1 = \alpha + \alpha^2 + \alpha^3 + \alpha^3 + \alpha^2 + \alpha + 1 + 1 = 0 \\ f(\alpha^8) &= \alpha^{32} + \alpha^{24} + \alpha^{16} + \alpha^8 + 1 = \alpha^2 + \alpha + \alpha^3 + \alpha^2 + \alpha + 1 + \alpha^3 + 1 = 0 \end{aligned}$$

Izračunali smo, da so tudi konjugiranke k $\alpha$ peti koreni enote. Matrika, ki pripada tej bazi je enaka:

$$T = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

V matriki $T$ je res natanko $2m-1=7$ enic, torej je $N$ optimalna normalna baza. V vsaki vrstici je po ena enica, razen v tretji, kjer so same enice, torej je $N$ res ONB tipa I. $\Diamond$

Sedaj privzemimo pogoje trditve 3.4.2 (ii) (ONB tipa II). Vidimo, da je element $\beta \in \mathbb{F}_{2^m}$ in da so $\beta, \beta^2, \ldots, \beta^{2^{m-1}}$ linearno neodvisni nad $\mathbb{F}_2$. Torej je $N = \{\beta, \beta^2, \ldots, \beta^{2^{m-1}}\}$ normalna baza vektorskega prostora $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$. Velja tudi

$$N = \{\gamma + \gamma^{-1}, \gamma^2 + \gamma^{-2}, \ldots, \gamma^m + \gamma^{-m}\}.$$

Označimo $\beta_i = \gamma^i + \gamma^{-i}$. Mešani produkti ($i \neq j$) elementov iz baze so enaki

$$\begin{aligned} \beta_i\beta_j &= (\gamma^i + \gamma^{-i})(\gamma^j + \gamma^{-j}) = \\ &= \gamma^{i+j} + \gamma^{-(i+j)} + \gamma^{i-j} + \gamma^{j-i} = \beta_{i+j} + \beta_{i-j}. \end{aligned} \tag{2.4}$$

Torej se da vsak produkt zapisati z elementi iz baze. Vseh neničelnih vrednosti v mešanih produktih je $2m-1$, torej je $N$ res optimalna normalna baza vektorskega prostora $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$. Poglejmo enostaven primer za ONB tipa II.

**Primer:** Vzemimo obseg $\mathbb{F}_{2^3}$. Število $2m+1 = 7$ je praštevilo, $m = 3$ je liho in 2 generira kvadratne ostanke v $\mathbb{Z}_7$. Po trditvi 2.4.3 obstaja v $\mathbb{F}_{2^3}$ ONB tipa II, generirana z $\beta = \gamma + \gamma^{-1}$, kjer je $\gamma$ primitivni sedmi koren enote. Naj bo $\gamma$ ničla polinoma $f(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$ in $\beta = \gamma + \gamma^{-1}$. Ker je $\gamma^7 = 1$, velja $\gamma^{-1} = \gamma^6$. Torej je $\beta = \gamma + \gamma^6$. Po trditvi je množica $\{\beta, \beta^2, \beta^4\}$ normalna baza obsega $\mathbb{F}_{2^3}$. Izračunajmo.

$$\begin{aligned} \beta &= \gamma + \gamma^6 = \gamma^5 + \gamma^4 + \gamma^3 + \gamma^2 + 1 \\ \beta^2 &= \gamma^5 + \gamma^2 \\ \beta^4 &= \gamma^4 + \gamma^3 \end{aligned}$$

Po drugi strani velja:

$$\begin{aligned} \gamma + \gamma^{-1} &= \gamma^5 + \gamma^4 + \gamma^3 + \gamma^2 + 1 \\ \gamma^2 + \gamma^{-2} &= \gamma^2 + \gamma^5 \\ \gamma^4 + \gamma^{-4} &= \gamma^4 + \gamma^3 \end{aligned}$$

Torej sta množici $\{\beta, \beta^2, \beta^4\}$ in $\{\gamma + \gamma^{-1}, \gamma^2 + \gamma^{-2}, \gamma^4 + \gamma^{-4}\}$ res enaki. $\Diamond$

Zapišimo trditev, ki nam bo podala način za konstrukcijo normalnih baz z nizko kompleksnostjo. Dokaz trditve najdemo v [4, trditev 5.5.].

**Trditev 2.4.4.** *Naj bo $q$ praštevilo ali potenca praštevila, $m, k$ naravni števili, tako je $mk+1$ praštevilo, ki ne deli števila $q$. Naj bo $\beta$ primitiven $(mk+1)$-vi koren*

*enote v* $\mathbb{F}_{q^{mk}}$. *Predpostavimo še, da je* $\gcd(mk/e, m) = 1$, *kjer je e red elementa q po modulu* $mk + 1$. *Potem za vsak primitiven k-ti koren enote* $\tau$ *v* $\mathbb{Z}_{mk+1}$ *element*

$$\alpha = \sum_{i=0}^{k-1} \beta^{\tau^i}$$

*generira normalno bazo vektorskega prostora* $\mathbb{F}_{q^m}$ *nad* $\mathbb{F}_q$ *s kompleksnostjo največ* $(k+1)m - 1$ *in največ* $km - 1$, *če je* $k \equiv 1 \pmod{p}$, *kjer je p karakteristika obsega* $\mathbb{F}_q$.

Element $\alpha$ imenujemo **Gaussova perioda** tipa $(m, k)$. Sedaj pa bomo s pomočjo optimalnih normalnih baz tipa II definirali novo vrsto baz, to so umbralne baze.

## UMBRALNE BAZE

Z optimalnimi normalnimi bazami smo dosegli poleg enostavnega kvadriranja tudi kar učinkovito množenje dveh elementov. Še vedno pa se zaplete pri inverzu. Da bi poenostavili računanje inverza, bomo vpeljali novo vrsto baz, umbralne baze. Iz optimalnih normalnih baz tipa I lahko samo s permutiranjem elementov dobimo polinomsko bazo, definirano z nerazcepnim polinomom $f(x) = \sum_{i=0}^{i=m} x_i$. Iz optimalnih normalnih baz tipa II pa s permutiranjem elementov dobimo umbralne baze. V naslednjem poglavju bomo videli učinkovit način za računanje inverza v umbralnih bazah. Naj bo $\alpha$ normalen element, ki generira optimalno normalno bazo tipa II. Ukvarjamo se torej z obsegi $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$. Potem je $\beta = \gamma + \gamma^{-1}$, kjer je $\gamma$ primitiven $(2m+1)$-vi koren enote, in $\gamma^{2m-1} = 1$. Definirajmo

$$\beta_i = \gamma^i + \gamma^{-i} \text{ za } i \in \mathbb{Z}.$$

Potem po definiciji optimalne normalne baze tipa II velja

$$\{\alpha^{2^0}, \alpha^{2^1}, ..., \alpha^{2^{m-1}}\} = \{\beta_1, \beta_2, ..., \beta_m\},$$

torej tudi $\beta_1, \beta_2, ..., \beta_m$ sestavljajo bazo vektorskega prostora $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$.

**Definicija 2.4.5.** *Bazo* $\{\beta_1, \beta_2, ..., \beta_m\}$ *imenujemo* **umbralna optimalna normalna baza**, *ali samo umbralna baza (UB).*

Samo s permutiranjem elementov optimalne baze tipa II smo dobili novo bazo, ki se bo dala zelo učinkovito implementirati. Vidimo, da je baza iz zadnjega primera umbralna baza. Za pretvorbo vektorja $a = \sum_{i=0}^{m-1} a_i\alpha_i,\ a_i \in \mathbb{F}_2$ iz optimalne normalne baze v umbralno bazo

$$(\alpha^{2^0}, \alpha^{2^1}, ..., \alpha^{2^{m-1}}) \to (\beta_1, \beta_2, ..., \beta_m),$$

permutiramo njegove koordinate. Poglejmo nekaj lastnosti elementov, predstavljenih v umbralni bazi.

**Lema 2.4.6.** *Naj bo $\gamma$ primitiven $(2m+1)$-vi koren enote in $\beta_i = \gamma^i + \gamma^{-i}$ za $i \in \mathbb{Z}$. Potem velja:*
*(i) $\beta_0 = 0$ in $(\beta_k)^2 = \beta_{2k}$ za vsak $k \in \mathbb{Z}$.*
*(ii) $\beta_{-k} = \beta_k$ za vsak $k \in \mathbb{Z}$.*
*(iii) $\beta_k = \beta_{k+(2m+1)}$ za vsak $k \in \mathbb{Z}$.*
*(iv) $\beta_i \cdot \beta_j = \beta_{i+j} + \beta_{i-j}$ za vsak $i, j \in \mathbb{Z}$.*

**Dokaz.** Lastnosti (i) in (iii) sledita iz dejstva, da je $\gamma$ primitivni $(2m+1)$-vi koren enote in so elementi iz obsega $\mathbb{F}_2$. Lastnost (ii) sledi iz komutativnosti seštevanja v obsegu $\mathbb{F}_{2^m}$. Lastnost (iv) smo že preverili v (2.4). □

Fiksirajmo $k \in \mathbb{Z}$. Po lastnostih (ii) in (iii) iz zgornje leme obstaja enolično določen $k' \in \{1, \ldots, m\}$, za katerega velja $\beta_k = \beta_{k'}$. Torej obstaja natanko $m$ različnih neničelnih $\beta_k$. Iz tega dobimo pravilo za množenje dveh elementov $\beta_i$ in $\beta_j$ iz umbralne baze:

$$\beta_i \cdot \beta_j = \begin{cases} \beta_{i+j} + \beta_{|i-j|}, & \text{za } i+j \leq m, \\ \beta_{2m+1-(i+j)} + \beta_{|i-j|}, & \text{sicer.} \end{cases}$$

Torej lahko indekse iz zgornje leme omejimo na množico $\{1, \ldots, m\}$.

Po lastnosti (iv) iz zgornje leme dobimo pri umbralni bazi pri množenju dva člena in nato znižamo indekse. Definirajmo polinom $p \in \mathbb{F}_2[x]$ stopnje največ $m$, torej $p = \sum_{i=0}^{m} p_i x^i$, za $p_i \in \mathbb{F}_2$, ki mu pravimo **umbralni polinom** $p$ in pišemo

$$p(\beta) = \sum_{i=0}^{m} p_i \beta_i.$$

Definirajmo še **umbralno stopnjo** za $p(\beta)$ kot največji indeks neničelnega koeficienta $p_i$, za $i = 1, \ldots, m$. Potem lahko poljuben element iz končnega obsega $\mathbb{F}_{2^m}$ z optimalno normalno bazo tipa II predstavimo z umbralnim polinomom stopnje največ $m$ in konstantnim členom enakim nič; oziroma kot $m$-terico $(p_1, p_2, \ldots, p_m)$. Podajmo še trditev o umbralnih bazah.

**Trditev 2.4.7.** *Naj bo $\{\beta_1, \beta_2, \ldots, \beta_m\}$ umbralna optimalna normalna baza vektorskega prostora $\mathbb{F}_{2^m}$. Potem držijo naslednje trditve:*

*(i) $\sum_{i=1}^{m} \beta_i = 1$, enota za množenje.*

*(ii) Za umbralna polinoma $p(\beta)$ in $q(\beta)$, kjer $\deg(q) < \deg(p)$, obstaja umbralni polinom $r(\beta)$, tako da velja $p(\beta) = q(\beta) * \beta_{\deg p - \deg q} + r(\beta)$, kjer je $\deg(r) < \deg(p)$.*

*(iii) $1 + \sum_{i=1}^{m} \beta_i$ je nerazcepen umbralni polinom.*

**Dokaz.**

(i) Po definiciji $\beta_i$ in ker je $\gamma$ primitiven $(2m+1)$-vi koren enote, sledi

$$\sum_{i=1}^{m} \beta_i = \sum_{i=1}^{m} (\gamma^i + \gamma^{-i}) = 1 + \gamma^{-m} \sum_{i=1}^{2m} \gamma^i = \frac{\gamma^{2m+1} + 1}{\gamma^m(\gamma + 1)} = 1,$$

kar je enota za množenje v prostoru polinomov za $\gamma$.

(ii) Očitno.

(iii) Denimo, da je polinom $1 + \sum_{i=1}^{m} \beta_i$ razcepen. Torej obstajata dva polinoma $r$ in $s$ iz obsega $\mathbb{F}_q[x]$, s stopnjama med 2 in $m-1$, tako da velja

$$1 + \sum_{i=1}^{m} \beta_i = r(\beta)s(\beta).$$

Izrazimo elemente $\beta_i$ z $\gamma^i + \gamma^{-1}$. Dobimo:

$$1 + \gamma + \gamma^{-1} + \gamma^2 + \gamma^{-2} + \cdots + \gamma^m + \gamma^{-m} = r_1(\gamma)s_1(\gamma).$$

Obe strani enačbe pomnožimo z $\gamma^m$:

$$\gamma^m + \gamma^{m+1} + \gamma^{m-1} + \gamma^{m+2} + \gamma^{m-2} + \cdots + \gamma^{2m} + 1 = r_1(\gamma)s_1(\gamma)\gamma^m.$$

Sedaj pomnožimo obe strani enačbe s polinomom $\gamma - 1$. Na levi strani enačbe dobimo zelo enostaven polinom:

$$\gamma^{2m+1} - 1 = r_1(\gamma)s_1(\gamma)\gamma^m(\gamma - 1).$$

Element $\gamma$ je $(2m+1)$-vi primitivni koren enote, torej je ničla polinoma $x^{2m+1} - 1$, tj. je ničla polinoma na levi strani zgornje enačbe. Ker je primitivni $(2m+1)$-vi koren enote, ni enak 1, torej ni ničla polinoma $\gamma - 1$. Potem mora biti ničla vsaj enega od polinomov $r_1(x)$ in $s_1(x)$. Torej smo našli polinom stopnje manjše od $m$, ki ima $\gamma$ za ničlo, kar je v protislovju z definicijo elementa $\gamma$. Polinom iz trditve je res nerazcepen.

□

## 2.5 SEBIDUALNE BAZE

Opišimo še eno vrsto baz, to so sebidualne baze. So poseben primer dualnih baz, ki smo jih opisali v razdelku 2.1. Najprej definirajmo koncept, povezan s sebidualnostjo, nato pa sebidualne baze.

**Definicija 2.5.1.** *Baza* $\bar{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_m\}$ *je* **ortogonalna glede na sled**, *če velja* $\mathrm{Tr}(\alpha_i\alpha_j) = 0$ *za vsak* $i \neq j$. *Če velja še* $\mathrm{Tr}(\alpha_i^2) = 1$, *za* $i = 1, 2, ..., m$, *je baza* $\bar{\alpha}$ **sebidualna**. *Torej bazi, ki je enaka svoji dualni bazi, pravimo sebidualna baza.*

Poglejmo, v katerih primerih sebidualne baze sploh obstajajo. Najprej pokažimo trditev, da za $m \geq 2$ sebidualnih polinomskih baz ni.

**Trditev 2.5.2.** *Ne obstaja sebidualna polinomska baza vektorskega prostora* $\mathbb{F}_{q^m}$ *nad* $\mathbb{F}_q$, *kjer je* $m \geq 2$.

**Dokaz.** Predpostavimo, da takšna baza obstaja. Naj bo $\bar{\alpha} = \{1, \alpha, ..., \alpha^{m-1}\}$ sebidualna polinomska baza vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$. Torej za $0 \geq i, j \geq m-1$ velja $Tr(\alpha^i\alpha^j) = \delta_{ij}$. Ločimo naslednja primera:

i. Če je $q$ sod, velja
$$\mathrm{Tr}(\alpha) = \mathrm{Tr}(1 \cdot \alpha) = 0$$
in
$$\mathrm{Tr}(\alpha \cdot \alpha) = \mathrm{Tr}(\alpha^2) = (\mathrm{Tr}(\alpha))^2 = 1.$$
Prišli smo do protislovja.

ii. Najprej pokažimo, da pri lihem $q$ velja $m-1 \geq 2$. Naj bo karakteristika obsega $\mathbb{F}_q$ enaka $p > 2$. Velja $\mathrm{Tr}(1) = 1 = \sum_{i=0}^{m-1} 1$, od koder sledi, da je $m - 1 \equiv 1 \pmod{p}$. Ker je $p > 2$, mora biti $m \geq 4$. Dobili smo
$$\mathrm{Tr}(\alpha^2) = \mathrm{Tr}(\alpha \cdot \alpha) = 1 = \mathrm{Tr}(1 \cdot \alpha^2) = 0,$$
torej smo spet prišli do protislovja.

□

Torej bomo morali iskati sebidualne baze med normalnimi bazami. Na srečo nam naslednja trditev zagotavlja obstoj sebidualnih baz. Dokaz najdemo v [4].

**Trditev 2.5.3.** *Končen obseg* $\mathbb{F}_{q^m}$ *ima sebidualno bazo nad* $\mathbb{F}_q$ *natanko tedaj, ko je* $q$ *sodo število ali pa sta števili* $q$ *in* $m$ *lihi.* □

Torej v našem primeru, tj. za binarne obsege $\mathbb{F}_{2^m}$ nad obsegom $\mathbb{F}_2$, vedno obstaja sebidualna baza. Zanima nas, koliko različnih sebidualnih baz obstaja za vsak $m$. Pri tem si bomo pomagali z naslednjo trditvijo. Spomnimo se, da je matrika $A$ ortogonalna, ko velja $AA^T = I$. Videli bomo, da je ortogonalna transformacija sebidualne baze spet sebidualna. Pokazali bomo tudi, da samo z ortogonalno linearno transformacijo dobimo spet sebidualno bazo.

**Trditev 2.5.4.** *Naj bo $\bar{\beta} = \{\beta_1, \beta_2, ..., \beta_m\}$ sebidualna baza vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$, kjer je število $q$ sodo ali pa sta obe števili $q$ in $m$ lihi. Naj bo $C = (c_{ij})$ $m \times m$ razsežna obrnljiva matrika nad $\mathbb{F}_q$ in $\bar{\gamma} = \{\gamma_1, \gamma_2, ..., \gamma_m\}$ baza, za $1 \geq i \geq m$ definirana z:*

$$\gamma_i = \sum_{j=1}^{m} c_{ij}\beta_j.$$

*Potem je baza $\bar{\gamma}$ sebidualna natanko takrat, ko je $C$ ortogonalna matrika.*

**Dokaz.** Za $1 \geq i, j \geq m$ velja

$$\begin{aligned}\mathrm{Tr}\big(\gamma_i\gamma_j\big) &= \mathrm{Tr}\Big(\Big(\sum_k c_{ik}\beta_k\Big)\Big(\sum_\ell c_{j\ell}\beta_\ell\Big)\Big)\\ &= \sum_k\sum_\ell c_{ik}c_{j\ell}Tr(\beta_k\beta_\ell)\\ &= \sum_k c_{ik}c_{jk} = (CC^T)_{ij}.\end{aligned}$$

Pri drugem enačaju smo upoštevali linearnost sledi, pri tretjem pa, da je baza $\bar{\beta}$ sebidualna in je število $\mathrm{Tr}(\beta_k\beta_l)$ enako ena zgolj $k = \ell$, sicer pa enako nič. Baza $\bar{\gamma}$ je sebidualna natanko tedaj, ko je $(CC^T)_{ij} = I$. Slednje pomeni, da je $C$ ortogonalna matrika. □

Iz zgornje trditve lahko takoj izračunamo število sebidualnih baz vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$.

**Posledica 2.5.5.** *Število sebidualnih baz vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$ je enako*

$$SD(m, q) = \frac{1}{m!}|O(m, q)|,$$

*kjer smo z $O(m, q)$ označili grupo vseh ortogonalnih matrik nad $\mathbb{F}_q$ velikosti $m \times m$.*

Število takšnih matrik je znano. Zapišimo ga v naslednji trditvi, katere dokaz tudi najdemo v [4, izrek 1.12].

**Trditev 2.5.6.** *Število sebidualnih baz obsega $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$ je enako*

$$SD(m, q) = \begin{cases} \frac{1}{m!}\prod_{i=1}^{m-1}(q^i - a_i), & q \text{ sod;}\\ \frac{2}{m!}\prod_{i=1}^{m-1}(q^i - a_i), & q \text{ in } m \text{ liha;}\\ 0, & \text{sicer;}\end{cases}$$

*kjer je $a_i$ enak 1, če je $i$ sod, in 0 sicer.* □

## 2.6 PREHOD MED BAZAMI

Kot bomo videli pri analizi algoritmov, bi bilo idealno, če bi lahko eno operacijo izvajali v eni, drugo pa v drugi bazi. Ravnali bi se po najugodnejših algoritmih. Vendar to ni tako enostavno izvedljivo, saj bi nas lahko takšno prehajanje med bazami drago stalo. Vsako bazo končnega obsega $\mathbb{F}_{2^m}$ sestavlja $m$ elementov obsega, torej $m$ binarnih nizov dolžine $m$. Ker si lahko prostor $\mathbb{F}_{2^m}$ predstavljamo kot vektorski prostor, je osnovni način za pretvorbo med dvema bazama množenje z matriko. To pa je samo po sebi kar zamuden proces. Množenje vektorja dolžine $m$ z matriko velikosti $m \times m$ v splošnem zahteva $\mathcal{O}(m^3)$ operacij. To je žal za naše namene veliko preveč. Seveda je v določenih primerih prehodna matrika lepe oblike (npr. ima malo neničelnih elementov). Tedaj je prehod seveda hitrejši. V splošnem pa je to kompleksnejši problem, ki je vreden temeljitejše obravnave. To pa presega okvir te naloge. Mi bomo poskusili preučiti najhitrejše algoritme za vsak razred baz posebej. Pri vsakem razredu bomo izkoristili njene lepe lastnosti (npr. kvadriranje je izredno enostavno v normalnih bazah) ter si izbrali še kakšne dodatne lepe lastnosti (npr. pri polinomskih bazah izberemo tiste, ki so definirane z nerazcepnim trinomom, ali pri normalnih bazah optimalne normalne baze).

# Poglavje 3

# ALGORITMI ZA OSNOVNE OPERACIJE

Pri kriptografiji z eliptičnimi krivuljami se pogosto srečamo s problemom seštevanja točk ali podvajanja točk na eliptični krivulji. Za ti dve operaciji pa potrebujemo čim boljše algoritme za aritmetiko z elementi iz končnega obsega. Torej moramo ločiti med aritmetiko s točkami na eliptični krivulji in aritmetiko elementov iz obsega. Na hitro bomo predstavili eliptične krivulje nad binarnimi obsegi. Točke z eliptične krivulje lahko predstavimo na različne načine. Predstavili bomo dva, to so afine in projektivne koordinate. Predstavitev točk v različnih koordinatah spremeni tudi algoritme za seštevanje in podvajanje točk. Projektivne koordinate vpeljemo, na primer, če se hočemo izogniti sprotnemu invertiranju v končnih obsegih. Nato se bomo posvetili aritmetiki z elementi iz končnega obsega. V tem delu obravnavamo samo končne obsege oblike $\mathbb{F}_{2^m}$, tako tudi aritmetika poteka nad binarnimi obsegi. Osnovne operacije, ki jih bomo preučevali, so seštevanje (ki je zaradi linearnosti baze trivialno), množenje, kvadriranje in invertiranje. Sem spada tudi reševanje kvadratne enačbe, ki predstavlja kar kompleksen problem. Mi se z njim ne bomo ukvarjali. S predstavitvijo elementov iz obsega v različnih bazah lahko precej pridobimo na hitrosti algoritmov ali na prostoru, ki ga algoritem zahteva. Seštevanje elementov je v obeh bazah (polinomski in normalni) navadno seštevanje po komponentah, oziroma, ker gledamo na obseg $\mathbb{F}_{2^m}$ kot na vektorski prostor nad obsegom $\mathbb{F}_2$, kar ekskluzivni-ali (XOR). Ostali algoritmi pa so zahtevnejši.

Najprej bomo spoznali nekaj osnov eliptičnih krivulj nad binarnimi končnimi obsegi. Posebej bomo predstavili afine in projektivne koordinate ter predstavili seštevanje in podvajanje točk glede na različne koordinate. Nato se bomo posvetili aritmetiki nad končnimi obsegi $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$. Začeli bomo s primerom, ko so elementi predstavljeni v polinomski bazi. Najprej bomo obravnavali množenje nad $\mathbb{F}_{2^m}$, nato pa ločili algoritma za običajno množenje in za redukcijo po modulu. Sledi kvadriranje in nazadnje še računanje inverznega elementa. V drugem razdelku se bomo ukvar-

jali z aritmetiko v normalnih bazah. Najprej bomo podali algoritem za množenje, nato algoritem za inverz. Posebej bomo pogledali algoritme za osnovne operacije, ko uporabimo optimalne normalne baze, in na koncu algoritme z umbralnimi bazami. Poglavje zaključujemo še z enim algoritmom za množenje, to je množenje po bitih.

## 3.1 ELIPTIČNE KRIVULJE

V uvodu smo spoznali nekaj osnov o eliptičnih krivuljah in aritmetiki s točkami nad eliptičinimi krivuljami (seštevanje in podvajanje). Namen tega razdelka je podati osnovna dejstva na področju eliptičnih krivulj, definiranih nad binarnimi obsegi. Utemeljili bomo pomembnost hitre implementacije aritmetike nad binarnimi obsegi. Spoznali bomo dva načina za predstavitev točk na eliptični krivulji: v afinih in v projektivnih koordinatah. Videli bomo, da s tem, ko predstavimo elemente v določenih koordinatah, zelo vplivamo na algoritme za seštevanje in podvajanje točk.

### AFINE KOORDINATE

Naj bo $\mathcal{E}$ eliptična krivulja nad obsegom $\mathbb{F}_{2^m}$, podana z (afino) enačbo

$$y^2 + xy = x^3 + ax^2 + b,$$

kjer je $a \in \{0, 1\}$ in $b \neq 0$. Množico $\mathcal{E}(\mathbb{F}_{2^m})$ sestavljajo vse točke $(x, y)$, kjer sta $x, y \in \mathbb{F}_{2^m}$, ki zadoščajo zgornji enačbi, skupaj s posebno točko $\mathcal{O}$, ki jo imenujemo točka v neskončnosti. Spomnimo se, da je za $b = 0$ zgornja eliptična krivulja supersingularna. Takšnim krivuljam pa se želimo izogniti.

**Primer:** (Eliptična krivulja nad $\mathbb{F}_{2^4}$) Obseg $\mathbb{F}_{2^4}$ naj predstavlja nerazcepni trinom $f(x) = x^4 + x + 1$. Eliptična krivulja $\mathcal{E}$ naj bo oblike $y^2 + xy = x^3 + \alpha^4 x^2 + 1$ nad obsegom $\mathbb{F}_{2^4}$, torej sta elementa $a$ in $b$ iz zgornje formule enaka $a = \alpha^4$ in $b = 1$. Ker je $b \neq 0$, $\mathcal{E}$ ni supersingularna eliptična krivulja. Točke s krivulje $\mathcal{E}(\mathbb{F}_{2^4})$ so $\mathcal{O}$ in naslednje:

$$\begin{array}{lllll} (0,1) & (1,\alpha^6) & (1,\alpha^{13}) & (\alpha^3,\alpha^8) & (\alpha^3,\alpha^{13}) \\ (\alpha^5,\alpha^3) & (\alpha^5,\alpha^{11}) & (\alpha^6,\alpha^8) & (\alpha^6,\alpha^{14}) & (\alpha^9,\alpha^{10}) \\ (\alpha^9,\alpha^{13}) & (\alpha^{10},\alpha) & (\alpha^{10},\alpha^8) & (\alpha^{12},0) & (\alpha^{12},\alpha^{12}). \end{array}$$

Med vsemi pari smo poiskali tiste, ki zadoščajo zgornji enačbi. Izračunajmo še potence števila $\alpha$. Ker delamo nad binarnim obsegom $(1 = -1)$ in je redukcijski

polinom enak $f(x) = x^4 + x + 1$, velja $x^4 = x + 1$ za vsak $x \in \mathbb{F}_{2^m}$.

| | | | |
|---|---|---|---|
| $\alpha^0 = 1$ | $\alpha$ | $\alpha^2$ | $\alpha^3$ |
| $\alpha^4 = \alpha + 1$ | $\alpha^5 = \alpha^2 + \alpha$ | $\alpha^6 = \alpha^3 + \alpha^2$ | $\alpha^7 = \alpha^3 + \alpha + 1$ |
| $\alpha^8 = \alpha^2 + 1$ | $\alpha^9 = \alpha^3 + \alpha$ | $\alpha^{10} = \alpha^2 + \alpha + 1$ | $\alpha^{11} = \alpha^3 + \alpha^2 + \alpha$ |
| $\alpha^{12} = \alpha^3 + \alpha^2 + \alpha + 1$ | $\alpha^{13} = \alpha^3 + \alpha^2 + 1$ | $\alpha^{14} = \alpha^3 + 1$ | $\alpha^{15} = 1$. |

$\Diamond$

Kot smo videli v uvodu, vpeljemo na eliptičnih krivuljah posebno pravilo seštevanja in podvajanja točk, ki mu pravimo pravilo sekante/tangente. S to operacijo seštevanja tvori eliptična krivulja $\mathcal{E}(\mathbb{F}_{2^m})$ grupo. Točka $\mathcal{O}$ predstavlja v tej grupi enoto.

Dve različni točki s krivulje seštejemo tako, da ju povežemo s premico. Ta premica seka eliptično krivuljo v tretji točki, katero preslikamo čez abcisno os. Dobimo četrto točko, ki predstavlja vsoto prvih dveh. Če premica skozi dve točki ne seka krivulje v nobeni drugi točki, definiramo njuno vsoto kot točko $\mathcal{O}$. Pri podvajanju točke $P$ na eliptični krivulji pa skozi točko potegnemo tangento. Ta seka krivuljo še v eni točki, katero preslikamo čez abcisno os. Dobimo tretjo točko, ki je definirana kot dvakratnik točke $P$. V treh primerih je $P = -P$ (presečišča krivulje z abciso) in tangenta ne seka krivulje v nobeni drugi točki. Tedaj je dvakratnik te točke definiran kot točka $\mathcal{O}$. Zapišimo algebraične formule za seštevanje in podvajanje:

1. $P + \mathcal{O} = \mathcal{O} + P = P$ za vsako točko $P \in \mathcal{E}(\mathbb{F}_{2^m})$

2. Če je $P = (x, y) \in \mathcal{E}(\mathbb{F}_{2^m})$, potem velja $(x, y) + (x, x + y) = \mathcal{O}$. Točko $(x, x + y)$ označimo z $-P$ in ji rečemo **nasprotna**točka k točki $P$. Točka $-P$ je zagotovo element krivulje $\mathcal{E}(\mathbb{F}_{2^m})$.

3. (**Seštevanje točk**) Naj bosta $P = (x_1, y_1)$ in $Q = (x_2, y_2)$ dve točki na krivulji $\mathcal{E}(\mathbb{F}_{2^m})$ in $P \neq \pm Q$. Potem lahko koordinate točke $P + Q = (x_3, y_3)$ izračunamo na naslednji način:

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a, \quad \text{in} \quad y_3 = (x_1 + x_3)\lambda + x_3 + y_1,$$

kjer je

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2}.$$

4. (**Podvajanje točke**) Naj bo $P = (x_1, y_1)$ točka na krivulji $\mathcal{E}(\mathbb{F}_{2^m})$ in $P \neq -P$. Potem koordinate točke $2P = (x_3, y_3)$ izračunamo takole:

$$x_3 = \lambda^2 + \lambda + a \quad \text{in} \quad y_3 = x_1^2 + x_3\lambda + x_3,$$

kjer je v tem primeru

$$\lambda = \frac{y_1}{x_1} + x_1.$$

V obeh primerih, pri seštevanju in pri podvajanju, zahtevajo izračun rezultata 1 invertiranje nad obsegom in 2 množenji nad obsegom. Seštevanje in kvadriranje sta cenejši operaciji.

**Primer:** (Seštevanje in podvajanje točk na eliptični krivulji) Vzemimo isto krivuljo kot pri prejšnjem primeru.

1. Seštejmo točki $P = (\alpha^6, \alpha^8)$ in $Q = (\alpha^3, \alpha^{13})$. Vsoto $P + Q = (x_3, y_3)$ izračunamo tako:

$$\begin{aligned} \lambda &= \frac{\alpha^8 + \alpha^{13}}{\alpha^6 + \alpha^3} = \frac{\alpha^3}{\alpha^2} = \alpha, \\ x_3 &= \lambda^2 + \lambda + \alpha^6 + \alpha^3 + \alpha^4 = \\ &= \alpha^2 + \alpha + \alpha^6 + \alpha^3 + \alpha^4 = 1 \text{ in} \\ y_3 &= \lambda(\alpha^6 + 1) + 1 + \alpha^8 = \alpha\alpha^{13} + \alpha^2 = \alpha^{13}. \end{aligned}$$

   Pri računanju smo si pomagali z zgoraj preračunanimi vrednostmi potenc števila $\alpha$. Na primer, seštejmo $\alpha^8 + \alpha^{13} = \alpha^2 + 1 + \alpha^3 + \alpha^2 + 1 = \alpha^3$ in $\alpha^6 + \alpha^3 = \alpha^3 + \alpha^2 + \alpha^3 = \alpha^2$. Dobili smo torej vsoto $P + Q = (1, \alpha^{13})$.

2. Naj bo $P = (\alpha^6, \alpha^8)$. Potem je $2P = P + P = (x_3, y_3)$, kjer $x_3$ in $y_3$ izračunamo tako:

$$\begin{aligned} \lambda &= \frac{\alpha^8}{\alpha^6} + \alpha^6 = \alpha^3, \\ x_3 &= (\alpha^3)^2 + \alpha^3 + \alpha^4 = \alpha^2 + \alpha^4 = \alpha^{10} \text{ in} \\ y_3 &= (\alpha^6)^2 + \alpha^{10}\alpha^3 + \alpha^{10} = \alpha^{12} + \alpha^{13} + \alpha^{10} = \alpha^8. \end{aligned}$$

   Dobili smo, da je $2P = (\alpha^{10}, \alpha^8)$.

$\Diamond$

## PROJEKTIVNE KOORDINATE

V primeru, ko je invertiranje v $\mathbb{F}_{2^m}$ drago v primerjavi z množenjem, je bolje predstaviti točke v projektivnih koordinatah. Opisali bomo tri vrste projektivnih koordinat. V *standardnih* projektivnih koordinatah ustreza projektivna točka $(X, Y, Z), Z \neq 0$, afini točki $(X/Z, Y/Z)$. **Projektivna enačba** za eliptično krivuljo je enaka $Y^2Z + XYZ = X^3 + aX^2Z + bZ^3$. V *Jacobijevih* projektivnih koordinatah ustreza projektivna točka $(X, Y, Z), Z \neq 0$, afini točki $(X/Z^2, Y/Z^2)$ in projektivna enačba za krivuljo je enaka $Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6$. Poznamo še eno vrsto projektivnih koordinat, kjer projektivna točka $(X, Y, Z), Z \neq 0$ ustreza afini točki $(X/Z, Y/Z^2)$ in je projektivna enačba za krivuljo enaka

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4 \tag{3.1}$$

Formule za seštevanje in množenje točk na eliptični krivulji v projektivnih koordinatah ne zahtevajo invertiranja. Lahko jih izpeljemo tako, da točke najprej pretvorimo v afine koordinate in nato uporabimo formule za seštevanje afinih točk. Tudi metode za množenje točk, ki delujejo od leve proti desni, izvedejo seštevanje dveh točk v mešanih koordinatah; ena točka je podana v afinih, druga pa v projektivnih koordinatah. Formule za podvajanje točk za projektivno enačbo (3.1) so naslednje: $2(X_1, Y_1, Z_1) = (X_3, Y_3, Z_3)$, kjer so:

$$Z_3 = X_1^2 \cdot Z_1^2, X_3 = X_1^4 + b \cdot Z_1^4, Y_3 = bZ_1^4 \cdot Z_3 + X_3 \cdot (aZ_3 + Y_1^2 + bZ_1^4).$$

Formule za seštevanje v mešanih koordinatah so: $(X_1, Y_1, Z_1)+(X_2, Y_2, 1) = (X_3, Y_3, Z_3)$, kjer je

$$A = Y_2 \cdot Z_1^2 + Y_1, B = X_2 \cdot Z_1 + X_1, C = Z_1 \cdot B, D = B^2 \cdot (C + aZ_1^2),$$

$$Z_3 = C^2, E = A \cdot C, X_3 = A^2 + D + E, F = X_3 + X_2 \cdot Z_3,$$

$$G = X_3 + Y_2 \cdot Z_3, Y_3 = E \cdot F + Z_3 \cdot G.$$

Videli smo, da za seštevanje in podvajanje afinih točk z eliptične krivulje potrebujemo dve množenji in eno invertiranje. Pri projektivnih točkah pa se sprotnemu invertiranju izognemo na račun nekaj prostora (za točko shranjujemo po tri koordinate namesto običajnih dveh) in večjega števila množenj. Inverz računamo le na koncu, ko pretvarjamo točke nazaj v afine koordinate. Invertiranje elementov nad končnimi obsegi je, kot bomo videli v nadaljevanju, mnogo dražje od množenja.
V tem podpoglavju smo dobili nekaj občutka za to, kako pomembna je hitra implementacija algoritmov za aritmetiko z elementi nad končnimi obsegi.

## 3.2 ARITMETIKA S POLINOMSKIMI BAZAMI

Poglejmo si algoritme za osnovne operacije, ko so elementi iz končnega obsega predstavljeni v polinomski bazi. Vsi ti algoritmi so povzeti po članku [3].

### MNOŽENJE

Posvetimo se problemu množenja, ko predstavimo elemente v polinomski bazi. Naj bo $f(x) = x^m + r(x)$ nerazcepen binaren polinom stopnje $m$. Elementi obsega $\mathbb{F}_{2^m}$ so polinomi stopnje največ $m - 1$, množenje izvedemo po modulu $f(x)$. Element obsega $a(x) = a_{m-1}x^{m-1} + \cdots + a_2x^2 + a_1x + a_0$ predstavimo kot binaren vektor $a = (a_{m-1}, \ldots, a_2, a_1, a_0)$ dolžine $m$. Naj bo $t = \lceil m/w \rceil$, in naj bo $s = wt - m$, kjer predstavlja $w$ dolžino besede ($w \in \{4, 8, 16, 32, 64\}$, v softwaru je večinoma enaka 32, v zadnjem času pa tudi 64). Element $a$ bomo shranili v seznam $t$ $w$-bitnih besed: $A = (A[t-1], \ldots, A[2], A[1], A[0])$, kjer je skrajno desni bit v $A[0]$ enak $a_0$, $s$ skrajno

levih bitov v $A[t-1]$ pa je neuporabljenih in poskrbimo, da so enaki 0.
Algoritem za metodo **pomakni-in-prištej** temelji na množenju v smislu $a \cdot b = a_{m-1}x^{m-1}b + \cdots + a_2x^2b + a_1xb + a_0b$. Ko algoritem iteriramo po $i$, dobimo $x^ib \pmod{f(x)}$ in če je $a_i = 1$, to prištejemo v tekoči spomin ($c$). Element $b \cdot x \pmod{f(x)}$ enostavno izračunamo z levim pomikom vektorja $b$ in, če je $b_m = 1$, zaradi redukcije prištejemo $r(x)$ k $b$. Zapišimo ta algoritem (pomik izvedemo od desne proti levi).

**ALGORITEM 1.** *Množenje v polinomskih bazah po metodi pomakni-in-prištej*
INPUT: Binarna polinoma $a(x)$ in $b(x)$ stopnje največ $m-1$.
OUTPUT: $c(x) = a(x) \cdot b(x) \pmod{f(x)}$.

1. If $a_0 = 1$ then $c \leftarrow b$ ; else $c \leftarrow 0$.
2. For $i$ from 1 to $m-1$ do
   2.1 $b \leftarrow b \cdot x \pmod{f(x)}$.
   2.2 If $b_m = 1$ then $b \leftarrow b + r$. {Redukcija}
   2.3 If $a_i = 1$ then $c \leftarrow c + b$.
3. Return($c$).

Algoritem 1 je najbolj primeren takrat, ko lahko izvedemo vektorski pomik v enem ciklu. Ko potrebujemo veliko število besednih pomikov, pa je softwarska implementacija težja. Množenje lahko hitreje izvedemo, če najprej zmnožimo elemente obsega kot običajne polinome in nato še reduciramo po modulu $f(x)$.

**Množenje elementov obsega**
Sedaj bomo predstavili nekaj metod za množenje elementov obsega, brez upoštevanja redukcije po modulu polinoma $f(x)$. Redukcijo bomo izvedli v naslednji fazi. Izkaže se, da lahko s tem precej pospešimo celotno množenje.
**Metoda glavnika** za množenje polinomov temelji na naslednjem dejstvu. Če smo izračunali $b(x) \cdot x^k$ za nek $k \in [0, 31]$, potem lahko dobimo $b(x) \cdot x^{32j+k}$ kar s pripetjem $j$ ničelnih besed na desni strani vektorskega zapisa elementa $b(x) \cdot x^k$. Bite v besedah elementa $A$ lahko jemljemo z desne proti levi (Algoritem 2) ali z leve proti desni (Algoritem 3). Za vektor $C = (C[n], \ldots, C[2], C[1], C[0])$ naj bo $C\{j\}$ oznaka za skrajšan (angl. truncated) vektor $(C[n], \ldots, C[j+1], C[j])$.

**ALGORITEM 2.** *Metoda glavnika (od desne proti levi) za množenje v polinomskih bazah*

INPUT: Binarna polinoma $a(x)$ in $b(x)$ stopnje največ $m-1$.

OUTPUT: $c(x) = a(x) \cdot b(x)$.

1. $C \leftarrow 0$.
2. For $k$ from 0 to 31 do
   2.1 For $j$ from 0 to $t-1$ do
   If $k$-ti bit $A[j]$ enak 1 then prištej $B$ k $C\{j\}$.
   2.2 If $k \neq 31$ then $B \leftarrow B \cdot x$.
3. Return($C$).

**ALGORITEM 3.** *Metoda glavnika (od leve proti desni) za množenje v polinomskih bazah*

INPUT: Binarna polinoma $a(x)$ in $b(x)$ stopnje največ $m-1$.

OUTPUT: $c(x) = a(x) \cdot b(x)$.

1. $C \leftarrow 0$.
2. For $k$ from 31 downto 0 do
   2.1 For $j$ from 0 to $t-1$ do
   If $k$-ti bit $A[j]$ enak 1 then prištej $B$ k $C\{j\}$.
   2.2 If $k \neq 0$ then $C \leftarrow C \cdot x$.
3. Return($C$).

Videli bomo, da sta algoritma 2 in 3 mnogo hitrejša od Algoritma 1, saj imamo manj vektorskih pomikov (množenj z $x$). Algoritem 2 (od desne proti levi) je hitrejši od Algoritma 3, saj izvajamo vektorske pomike pri prvem na $t$-besednem vektorju $B$, pri drugem (od leve proti desni) pa na $2t$-besednem vektorju $C$. Vendar pa lahko Algoritem 3 občutno pospešimo na račun nekaj prostora tako, da preračunamo $u(x) \cdot b(x)$ za vse polinome $u(x)$ stopnje manjše od $v$, kjer $v$ deli dolžino besed, in z obravnavanjem $v$ bitov od $A[j]$ naenkrat. V Algoritmu 4 predstavljamo izboljšano verzijo algoritma 3 za $v = 4$ (to pomeni, da razdelimo besedo dolžine $w = 32$ na 8 delov dolžine 4).

**ALGORITEM 4.** *Množenje po metodi glavnika (od leve proti desni) z okni širine* $v = 4$, $w = 32$, $t = \lceil m/w \rceil$.

INPUT: Binarna polinoma $a(x)$ in $b(x)$ stopnje največ $m - 1$.

OUTPUT: $c(x) = a(x) \cdot b(x)$.

1. Izračunaj $B_u = u(x) \cdot b(x)$ za vse polinome $u(x)$ stopnje največ 3.
2. $C \leftarrow 0$
3. For $k$ from 7 downto 0 do ($7 = w/v - 1$)
   3.1 For $j$ from 0 to $t - 1$ do
   Naj bo $u = (u_3, u_2, u_1, u_0)$, kjer je $u_i$ $(4k + i)$-ti bit od $A[j]$.
   Prištej $B_u$ k $C\{j\}$.
   3.2 If $k \neq 0$ then $C \leftarrow C \cdot x^4$.
4. Return ($C$).

Opišimo še eno idejo za množenje polinomov. Povzeta je po Karatsubovi metodi za množenje naravnih števil. Predpostavimo, da je $m$ sodo število (če ni, ga povečamo tako, da damo na začetek ničlo). Za množenje dveh binarnih polinomov $a(x)$ in $b(x)$ stopenj največ $m - 1$, najprej razdelimo $a(x)$ in $b(x)$ na dva polinoma stopenj največ $(m/2) - 1$ takole: $a(x) = A_1(x)X + A_0(x)$, $b(x) = B_1(x)X + B_0(x)$, kjer je $X = x^{m/2}$. Potem velja:

$$\begin{aligned} a(x)b(x) &= A_1B_1X^2 + (A_1B_0 + A_0B_1)X + A_0B_0 = \\ &= A_1B_1X^2 + [(A_1 + A_0)(B_1 + B_0) + A_1B_1 + A_0B_0]X + A_0B_0, \end{aligned}$$

kar lahko izračunamo s tremi produkti polinomov stopnje $(m/2) - 1$ in ne s štirimi, kot se zdi najprej. Te produkte lahko izračunamo rekurzivno.
V primeru $m = 163$ najprej pripnemo ničelni bit k elementoma $a$ in $b$, tako da je njuna bitna dolžina 164, potem pa uporabimo Karatsubovo metodo. Tako reduciramo množenje $a$ in $b$ na množenje polinomov stopnje največ 20 (to je ravno nekaj manj od dolžine besed $w = 32$). Naslednja množenja izvedemo z uporabo različice algoritma 4. V primeru $m = 233$ najprej pripnemo 23 ničelnih bitov k $a$ in $b$, potem pa s Karatsubovo metodo zreduciramo množenje polinomov $a$ in $b$ na množenje polinomov stopenj največ 63 (ravno manj od 64). Podrobneje se s tem algoritmom ne bomo ukvarjali, ker se v praksi izkaže, da je približno enako učinkovit kot zgornji algoritem po metodi glavnika.

## REDUKCIJA

Poglejmo si dve metodi za redukcijo polinomov po modulu polinoma $f(x)$. Naj

bo $c(x)$ binaren polinom stopnje kvečjemu $2m-2$. Še vedno uporabljamo oznako $f(x) = x^m + r(x)$. Algoritem 5 reducira $c(x)$ po modulu $f(x)$ po en bit, začenši s skrajno levim bitom. Temelji na dejstvu, da za $i \geq m$ velja $x^i \equiv x^{i-m}r(x) \pmod{f(x)}$. Polinome $x^k r(x)$, $0 \leq k \leq 31$, lahko vnaprej izračunamo. Če je $r(x)$ polinom nizke stopnje ali če je $f(x)$ trinom, potem algoritem zahteva manj prostora in so seštevanja, ki zadevajo $x^k r(x)$ hitrejša.

**ALGORITEM 5.** *Redukcija po modulu (po bitih)*

INPUT: Polinom $c(x)$, stopnje največ $2m-2$, in polinom $f(x) = x^m + r(x)$, kjer je stopnja $r(x)$ manjša od $m$, $t = \lceil m/w \rceil$.

OUTPUT: $c(x) \mod f(x)$.

1. Izračunaj $u_k(x) = x^k r(x)$, $0 \leq k \leq 31$.
2. For $i$ from $2m-2$ downto $m$ do
   2.1 If $c_i = 1$ then
   Naj bo $j = \lfloor (i-m)/32 \rfloor$ in $k = (i-m) - 32j$.
   Prištej $u_k(x)$ k $C\{j\}$.
3. Return$(C[t-1], ..., C[1], C[0])$.{Vrnemo samo reducirani del.}

Če je $f(x)$ trinom ali pentonom s čim nižjimi stopnjami srednjih členov, potem lahko redukcijo polinoma $c(x)$ po modulu $f(x)$ učinkovito izvedemo po besedah. Na primer, reducirajmo deveto besedo $C[9]$ polinoma $c(x)$ po modulu polinoma $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$. Torej sta $m = 163$ in $t = 6$. Dobimo:

$$\begin{aligned} x^{288} &\equiv x^{132} + x^{131} + x^{128} + x^{125} \pmod{f(x)} \\ x^{289} &\equiv x^{133} + x^{132} + x^{129} + x^{126} \pmod{f(x)} \\ &\vdots \\ x^{319} &\equiv x^{163} + x^{162} + x^{159} + x^{156} \pmod{f(x)}. \end{aligned}$$

Iz desnih strani zgornjih kongruenc sledi, da lahko izvedemo redukcijo $C[9]$ tako, da štirikrat prištejemo pomaknjeno besedo $C[9]$ k elementu $C$, skrajno desni bit besede $C[9]$ prištejemo k bitom $132, 131, 128$ in $125$ elementa $C$. To nas privede do algoritma 6 za redukcijo po modulu, ki jo zlahka razširimo na druge nerazcepne polinome. Algoritem 6 je hitrejši od Algoritma 5 in zahteva enako prostora. Reducirati moramo besede $6, 7, 8, 9$ in $10$ ($t = 6$ in $2(t-1) = 10$, prva beseda je označena z 0).

**ALGORITEM 6.** *Redukcija po modulu (po besedah)*

INPUT: Binaren polinom $c(x)$ stopnje največ $2m-2 = 324$, polinom $f(x) = x^{163}+x^7+x^6+x^3+1$, $w = 32$, $t = \lceil m/w \rceil$.

OUTPUT: $c(x) \mod f(x)$.

1. For $i$ from 10 downto 6 do {Redukcija besede $C[i]$ po modulu $f(x)$}
   - 1.1 $T \leftarrow C[i]$.
   - 1.2 $C[i-6] \leftarrow C[i-6] \oplus (T \ll 29)$.
   - 1.3 $C[i-5] \leftarrow C[i-5] \oplus (T \ll 4) \oplus (T \ll 3) \oplus T \oplus (T \gg 3)$.
   - 1.4 $C[i-4] \leftarrow C[i-4] \oplus (T \gg 28) \oplus (T \gg 29)$.
2. $T \leftarrow C[5]$ AND $Ox$FFFFFFF8. { Počisti bite 0,1 in 2 v $C[5]$}
3. $C[0] \leftarrow C[0] \oplus (T \ll 4) \oplus (T \ll 3) \oplus T \oplus (T \gg 3)$.
4. $C[1] \leftarrow C[1] \oplus (T \gg 28) \oplus (T \gg 29)$.
5. $C[5] \leftarrow C[5]$ AND $0x00000007$. { Počisti neuporabljene bite v $C[5]$}
6. Return $((C[5], C[4], C[3], C[2], C[1], C[0]))$.

Podrobneje preučimo zgornji algoritem. Da bomo razumeli vrednosti pomikov, si preračunajmo tabelo vrednosti posameznih bitov v $i$-ti besedi. Ker smo vzeli za dolžino besede $w = 32$, tečejo indeksi od $32i$ do $32i+31$.

$$
\begin{aligned}
x^{i32} &= x^{(i-5)32+4} + x^{(i-5)32+3} + x^{(i-5)32} + x^{(i-6)32+29} \\
x^{i32+1} &= x^{(i-5)32+5} + x^{(i-5)32+4} + x^{(i-5)32+1} + x^{(i-6)32+30} \\
x^{i32+2} &= x^{(i-5)32+6} + x^{(i-5)32+5} + x^{(i-5)32+2} + x^{(i-6)32+31} \\
x^{i32+3} &= x^{(i-5)32+7} + x^{(i-5)32+6} + x^{(i-5)32+3} + x^{(i-5)32} \\
x^{i32+4} &= x^{(i-5)32+8} + x^{(i-5)32+7} + x^{(i-5)32+4} + x^{(i-5)32+1} \\
&\vdots \\
x^{i32+27} &= x^{(i-5)32+31} + x^{(i-5)32+30} + x^{(i-5)32+27} + x^{(i-5)32+24} \\
x^{i32+28} &= x^{(i-4)32} + x^{(i-5)32+31} + x^{(i-5)32+28} + x^{(i-5)32+25} \\
x^{i32+29} &= x^{(i-4)32+1} + x^{(i-4)32} + x^{(i-5)32+29} + x^{(i-5)32+26} \\
x^{i32+30} &= x^{(i-4)32+2} + x^{(i-4)32+1} + x^{(i-5)32+30} + x^{(i-5)32+27} \\
x^{i32+31} &= x^{(i-4)32+3} + x^{(i-4)32+2} + x^{(i-5)32+31} + x^{(i-5)32+28}
\end{aligned}
$$

Ker je $m = 163$ in $w = 32$, so v peti besedi porabljeni le trije biti ($163 = 5 \cdot 32 + 3$). Ostalih 29 bitov iz pete besede pa moramo reducirati. Torej reduciramo bite od 164

do 191, ki imajo naslednje vrednosti:

$$\begin{aligned} x^{164} &= x^8 + x^7 + x^4 + x \\ &\vdots \\ x^{188} &= x^{32} + x^{31} + x^{28} + x^{25} \\ x^{189} &= x^{32+1} + x^{32} + x^{29} + x^{26} \\ x^{190} &= x^{32+2} + x^{32+1} + x^{30} + x^{27} \\ x^{191} &= x^{32+3} + x^{32+2} + x^{31} + x^{28}. \end{aligned}$$

Iz tabele je razvidno, da moramo urediti še besedi $C[0]$ in $C[1]$. Na koncu počistimo odvečnih 29 bitov v besedi $C[5]$ (ko prištejemo $7 = 111_2$, se prvi trije biti ohranijo, ostali pa postanejo enaki 0).

## KVADRIRANJE

Kvadriranje polinomov je v obsegih karakteristike 2 mnogo hitrejše od množenja dveh poljubnih polinomov, saj je kvadriranje linearna operacija v $\mathbb{F}_{2^m}$. Če je $a(x) = \sum_{i=0}^{m-1} a_i x^i$, potem je njegov kvadrat enak $a(x)^2 = \sum_{i=0}^{m-1} a_i x^{2i}$. Binarno predstavitev elementa $a(x)^2$ dobimo tako, da vstavimo ničelne bite med bite binarne predstavitve elementa $a(x)$. V ta namen vnaprej pripravimo tabelo bytov za pretvorbo 8-bitnega niza v njegovo razširjeno 16-bitno verzijo.

**ALGORITEM 7.** *Kvadriranje v polinomskih bazah*

INPUT: $a \in \mathbb{F}_{2^m}$.

OUTPUT: $a^2 \mod f(x)$.

1. *Priprava.* Za vsak byte $v = (v_7, ..., v_1, v_0)$ pripravi 16-bitno vrednost $T(v) = (0, v_7, ..., 0, v_1, 0, v_0)$.
2. For $i$ from 0 to $t - 1$ do
   - 2.1 Naj bo $A[i] = (u_3, u_2, u_1, u_0)$, kjer je vsak $u_j$ byte.
   - 2.2 $C[2i] \leftarrow (T(u_1), T(u_0))$, $C[2i + 1] \leftarrow (T(u_3), T(u_2))$.
3. Izvedi redukcijo $b(x) = c(x) \mod f(x)$. (Algoritem 6)
4. Return $(b)$.

## INVERZ

Ker obravnavamo elemente iz končnega obsega $\mathbb{F}_{2^m}$, velja za poljuben element $a^{2^m} = a$ in zato bi lahko izračunali inverz po preprosti formuli $a^{2^m-2} = a^{-1}$. Za to bi v primeru $m = 163$ porabili 6-7 množenj. Z hitrejšimi algoritmi, kot je na primer razširjeni Evklidov algoritem, pa lahko dosežemo zgolj 3-4 množenja.

Evklidov algoritem za računanje največjega skupnega večkratnika je eden najstarejših poznanih algoritmov in eden najpomembnejših algoritmov v teoriji števil. Evklid ga je zapisal v svoji sedmi knjigi Elementi v tretjem stoletju pred našim štetjem. Algoritem 8 izračuna inverz neničelnega elementa iz obsega $a \in \mathbb{F}_{2^m}$ z uporabo razširjenega Evklidovega algoritma (angl. Extended Euclid Algorithm, EEA) za polinome. Podan imamo polinom $a(x)$ stopnje največ $m-1$ in iščemo polinom $b(x)$ stopnje največ $m-1$, tako da velja

$$a(x)b(x) \equiv 1 \mod f(x).$$

Ekvivalentno temu je, da velja $a(x)b(x) + d(x)f(x) = 1$ za nek polinom $d(x)$. Ker je $f(x)$ nerazcepen polinom, je največji skupni delitelj polinomov $f(x)$ in $a(x)$ enak 1. Zato lahko za iskanje inverznega elementa uporabimo Evklidov algoritem. Ker ne znamo poiskati take vrednosti $b(x)$ in $f(x)$, da bi ustrezala zgornji enačbi, si pomagamo takole. Enkrat vzamemo vrednosti $b = 0$ in $d = 1$, drugič pa obratno, $b = 1$ in $d = 0$. Dobimo naslednji dve enačbi:

$$ab_{-2} + fd_{-2} = f =: r_{-2},$$

$$ab_{-1} + fd_{-1} = a =: r_{-1}.$$

Drugo enačbo pomnožimo s $-s_0$ in jih seštejemo. Dobimo:

$$a(b_0 - s_0b_1) + f(d_0 - s_0d_1) = r_{-2} - s_0r_{-1}.$$

Torej začnemo z $r_{-2} = f$, $r_{-1} = a, b_{-2} = 0, b_{-1} = 1, d_{-2} = 1$ in $d_{-1} = 0$ in iščemo vrednosti $s_k$ in $r_k$, tako da velja

$$r_{k-2} = s_kr_{k-1} + r_k, \quad \deg r_k < \deg r_{k-1}.$$

Definirajmo

$$d_k = s_kd_{k-1} + d_{k-2} \quad \text{in} \quad b_k = s_kb_{k-1} + b_{k-2}.$$

Iteracijo ponavljamo, dokler za nek $n$ ne dobimo $r_n = 0$. Rešitev je podana z $d = d_{n-1}$ in $b = b_{n-1}$, kjer je $\deg d < \deg r$ in $\deg b < \deg f = m$. Ker nas zanima samo polinom $b$, lahko povsem izpustimo računanje vrednosti $d$-jev.
Algoritem vključuje spremenljivki $ba + df = u$ in $ca + ef = v$ za neka $d$ in $e$, ki nista posebej izračunana. Če velja $\deg(u) \geq \deg(v)$, pri vsaki iteraciji izvedemo parcialno

deljenje tako, da odštejemo $x^j v$ od $u$, kjer je $j = \deg(u) - \deg(v)$. S tem se stopnja $u$ zmanjša vsaj za 1, v povprečju pa za 2. Algoritem se ustavi, ko je $\deg(u) = 0$, torej je $u = 1$ in $ba + df = 1$, oziroma $b = a^{-1} \pmod{f(x)}$.

**ALGORITEM 8.** *Razširjen Evklidov algoritem za invertiranje v* $\mathbb{F}_{2^m}$

INPUT: $a \in \mathbb{F}_{2^m}, a \neq 0$.

OUTPUT: $a^{-1} \mod f(x)$.

1. $b \leftarrow 1,\ c \leftarrow 0,\ u \leftarrow a,\ v \leftarrow f$.
2. While $\deg(u) \neq 0$ do
   - 2.1 $j \leftarrow \deg(u) - \deg(v)$.
   - 2.2 If $j < 0$ then: $u \leftrightarrow v,\ b \leftrightarrow c,\ j \leftarrow -j$.
   - 2.3 $u \leftarrow u + x^j v,\ b \leftarrow b - x^j c$.
3. Return($b$).

## 3.3 ARITMETIKA Z NORMALNIMI BAZAMI

V tem razdelku bomo spoznali osnovne algoritme v primeru, ko elemente predstavimo v normalnih bazah. Kot pri polinomskih bazah seštejemo dva elementa iz $\mathbb{F}_{2^m}$ tako, da v $\mathbb{F}_2$ seštejemo istoležne koeficiente, oziroma izračunamo logični XOR med dvema zaporedjema ničel in enic dolžine $m$.
Poteciranje na $q$ je v normalnih bazah izredno enostavno. Naj bo $N = \{\alpha_0, \alpha_1, ..., \alpha_{m-1}\}$ normalna baza vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$, kjer je $\alpha_i = \alpha^{q^i}$. Za vsako naravno število $k$ velja $\alpha_i^{q^k} = \alpha_{i+k}$, kjer reduciramo indekse za $\alpha$ po modulu $m$. Torej lahko vsak element $a \in \mathbb{F}_{q^m}$ predstavimo kot

$$a = \sum_{i=0}^{m-1} a_i \alpha^{q^i},$$

kjer so $a_i \in \mathbb{F}_q$. Izračunajmo

$$a^q = \sum_{i=0}^{m-1} a_i^q \alpha^{q^{i+1}} = \sum_{i=0}^{n-1} a_i \alpha^{q^{i+1}} = a_{n-1}\alpha + \sum_{i=1}^{n-1} a_{i-1} \alpha^{q^i}.$$

Torej so koeficienti elementa $a^q$ enaki $(a_{m-1}, a_0, a_1, \ldots, a_{m-2})$, oziroma jih dobimo kar s cikličnim pomikom koeficientov elementa $a$ za eno mesto v desno.
Mi obravnavamo obsege $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$, torej je kvadriranje zgolj cikličen pomik koordinat vektorja $a$ in zato cena kvadriranja zanemarljiva. S tem lahko učinkovito

izpeljemo splošno potenciranje po metodi kvadriraj-in-množi. Žal pa je množenje dokaj zapleteno. V obsegih karakteristike $q \neq 2$ bi lahko množili dva elementa $a$ in $b$ enostavno po formuli

$$a \cdot b = \frac{(a+b)^2 - a^2 - b^2}{2} = \frac{(\frac{a+b}{2})^2 - (\frac{a-b}{2})^2}{2}.$$

Tako bi bila zahtevnost množenja podobnega reda kot kvadriranje. Žal pa v našem primeru, ko je karakteristika enaka dva, ta način ne pride v upoštev. Pri normalnih bazah je v splošnem množenje precej bolj zamudna operacija kot kvadriranje. Z vpeljavo optimalnih normalnih baz smo ta problem delno rešili, kar bomo videli v naslednjem razdelku. Posvetimo se torej množenju.

## MNOŽENJE

Poglejmo množenje dveh poljubnih elementov $a = (a_0, a_1, ..., a_{m-1})$ in $b = (b_0, b_1, ..., b_{m-1})$ iz obsega $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$, podanih v normalni bazi $N = \{\alpha_0, \alpha_1, ..., \alpha_{m-1}\}$. Njun produkt $a \cdot b$ označimo s $c = (c_0, c_1, ..., c_{m-1})$. Radi bi izrazili $c_i$ čim enostavneje z $a_i$ in $b_i$. Naj bo

$$\alpha_i \alpha_j = \sum_{k=0}^{m-1} t_{ij}^{(k)} \alpha_k, \tag{3.2}$$

kjer so $t_{ij}^{(k)} \in \mathbb{F}_2$. Za $0 \leq k \leq m-1$ izračunajmo

$$c_k = \sum_{i,j} a_i b_j t_{ij}^{(k)} = a\ T_k b^T, \tag{3.3}$$

kjer je $T_k = (t_{ij}^{(k)})$ matrika velikosti $m \times m$ nad $\mathbb{F}_2$ in $b^T$ transponirani vektor vektorja $b$. Če obe strani enačbe (3.2) potenciramo na $2^{-\ell}$-to potenco, ugotovimo, da za vsak $0 \leq i, j, \ell \leq m-1$ velja

$$t_{ij}^{(\ell)} = t_{i-\ell, j-\ell}^{(0)}\ .$$

Matrika $T_0$ je ravno multiplikacijska matrika za normalno bazo, ki smo jo definirali v razdelku o normalnih bazah, in jo označimo kar s $T = (t_{ij})$. Formulo (3.3) lahko torej pišemo takole.

$$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j t_{i-k, j-k} = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_{i+k} b_{j+k} t_{ij}.$$

Če naredimo zanko, ki izračuna $c_0$ pri vhodnih podatkih $a$ in $b$, ista zanka pri vhodnih podatkih $a^{2^{-\ell}}$ in $b^{2^{-\ell}}$ vrne člen $c_\ell$. Spomnimo se, da sta $a^{2^{-\ell}}$ in $b^{2^{-\ell}}$ samo ciklična pomika vektorjev $a$ in $b$ v normalni bazi.

Z multiplikacijsko tabelo si torej pomagamo pri implementaciji množenja elementov $a$ in $b$ iz $\mathbb{F}_{2^m}$. Sestavimo $m$ zank v odvisnosti od matrike $T$ tako, da nam pri

vhodnih podatkih $a$ in $b$ vsaka od $m$ zank vrne eno komponento produkta $c = a \cdot b$. Če je $m$ velik, je ta postopek nepraktičen. Na srečo imamo na voljo več normalnih baz vektorskega prostora $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$. Za nekatere baze je pripadajoča multiplikacijska tabela enostavnejša od drugih, oziroma ima manj neničelnih vrednosti ali kakšne druge lepe lastnosti. Spomnimo se, da pravimu številu neničelnih vrednosti v multiplikacijski tabeli kompleksnost baze. Kompleksnost je najmanjša pri optimalnih normalnih bazah. Pri teh bazah lahko sestavimo algoritem za množenje, ki bo omogočil učinkovitejšo hardversko in softversko implementacijo tudi za velike $m$.

Za vhodne podatke pri algoritmu za množenje bomo torej potrebovali vektorja $a$ in $b$ ter multiplikacijsko tabelo $T$ normalne baze. Množenje z matriko seveda priredimo glede na obliko matrike oziroma število elementov.

**ALGORITEM 9.** *Množenje v normalnih bazah*

INPUT: $a, b \in \mathbb{F}_{2^m}$, $T = (t_{ij})$ multiplikacijska tabela normalne baze $N$.

OUTPUT: $c = a \cdot b$.

1. For $k$ from 0 to $m-1$ do

   $$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_{i+k} b_{j+k} t_{ij}.$$

2. Return (c).

## INVERZ

Tako kot množenje, je tudi računanje inverznega elementa v normalnih bazah precej zamudno. Žal tudi z vpeljavo optimalnih normalnih baz ni situacija nič boljša. Poglejmo algoritem, ki so ga predlagali Itoh, Teechai in Tsujii [**?**, str. 85]. Če je $\alpha \in \mathbb{F}_{2^m}$ in $\alpha \neq 0$, potem velja

$$\alpha^{-1} = \alpha^{2^m-2} = (\alpha^{2^{m-1}-1})^2.$$

Če je $m$ liho število, potem iz

$$2^{m-1} - 1 = (2^{(m-1)/2} - 1)(2^{(m-1)/2} + 1)$$

sledi

$$\alpha^{2^{m-1}-1} = (\alpha^{2^{(m-1)/2}-1})^{2^{(m-1)/2}+1}.$$

Torej, če poznamo vrednost $\alpha^{2^{(m-1)/2}-1}$, potrebujemo za izračun $\alpha^{2^{m-1}-1}$ zgolj eno množenje in eno kvadriranje. Če pa je $m$ sodo število, potem velja

$$\alpha^{2^{m-1}-1} = \alpha^{2(2^{(m-2)/2}-1)(2^{(m-2)/2}+1)+1}$$

in, če poznamo vrednost $\alpha^{2^{(m-2)/2}-1}$, potrebujemo za izračun $\alpha^{2^{m-1}-1}$ dve množenji in dve kvadriranji. Zapišimo algoritem 10 za računanje inverznega elementa v normalni bazi.

**ALGORITEM 10.** *Inverz v normalnih bazah*

INPUT: $a \in \mathbb{F}_{2^m}$

OUTPUT: $b = a^{-1} = a^{2^m-2}$.

1. while $m > 1$ do

   If $m$ lih do

   1.1 $m \leftarrow (m-1)/2$

   1.2 $a \leftarrow a^{2^m+1}$

   Else ($m$ je sod)

   1.3 $m \leftarrow (m-2)/2$

   1.4 $a \leftarrow a^{2(2^m+1)+1}$

2. $b \leftarrow a^2$

3. Return (b).

Za boljšo predstavo pokažimo zgornji algoritem na primeru.

**Primer:** (Inverz v normalni bazi) Vzemimo končen obseg $\mathbb{F}_{2^{163}}$. Računamo inverz elementa $a \in \mathbb{F}_{2^{163}}$. Velja $a^{-1} = a^{2^{163}-2}$. Poglejmo, kako bi razčlenili število $2^{163}-2$.

$$\begin{aligned}
2^{163}-2 &= 2(2^{81}-1)(2^{81}+1)\\
2^{81}-1 &= 2(2^{40}-1)(2^{40}+1)+1\\
2^{40}-1 &= (2^{20}-1)(2^{20}+1)\\
2^{20}-1 &= (2^{10}-1)(2^{10}+1)\\
2^{10}-1 &= (2^{5}-1)(2^{5}+1)\\
2^{5}-1 &= 2(2^{2}-1)(2^{2}+1)+1\\
2^{2}-1 &= (2^{1}-1)(2^{1}+1)
\end{aligned}$$

Torej potrebujemo devet množenj in deset kvadriranj, da izračunamo inverz elementa iz končnega obsega $\mathbb{F}_{2^{163}}$. $\Diamond$

## 3.4 ARITMETIKA Z UMBRALNIMI BAZAMI

Umbralne baze združujejo nekatere lepe lastnosti polinomskih in normalnih baz in zaradi tega lahko zelo učinkovito implementiramo aritmetiko z umbralnimi bazami. Videli smo, da smo z normalnimi bazami dosegli enostavno kvadriranje. Nato smo vpeljali optimalne normalne baze in z njimi je postalo tudi računanje produkta hitrejše. Še vedno pa je bil velik problem računanje inverza. To lahko rešimo z umbralnimi bazami. Več o tem najdemo v [5].

Primerjajmo pravilo za množenje v umbralni bazi s pravilom za množenje v polinomski bazi z nerazcepnim polinomom $f$. Po lemi 2.4.6. (iv) smo pri umbralni bazi pri množenju dveh elementov $\beta_i$ in $\beta_j$, kjer sta $i$ in $j$ poljubni celi števili, dobili dva člena:

$$\beta_i \cdot \beta_j = \beta_{i+j} + \beta_{i-j}, \tag{3.4}$$

pri polinomski bazi pa dobimo samo en člen:

$$x^i \cdot x^j = x^{i+j}. \tag{3.5}$$

Pri umbralnih bazah znižamo indekse s pomočjo leme 2.4.6, pri polinomski bazi pa z nerazcepnim polinomom znižamo stopnjo. Najprej poglejmo algoritme za kvadriranje in množenje, na koncu pa se posvetimo inverzu.

### KVADRIRANJE

Primerjajmo algoritem za kvadriranje v umbralni bazi s pripadajočo operacijo v polinomski bazi z nerazcepnim polinomom $f$. Kvadrat elementa $a(x) = (a_0, a_1, \ldots, a_{m-1})$ v polinomski bazi je enak produktu $(a'_0, a'_1, \ldots, a'_{2(m-1)})$, kjer gre za navadno množenje polinomov:

$$a(x) \cdot a(x) = (a'_0, a'_1, \ldots, a'_{2(m-1)}).$$

Ta polinom moramo še zreducirati. Navadno izračunamo produkt z vnaprej pripravljeno tabelo, za vsak bit originalnega niza postavi dodaten ničelni bit:

$$(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7) \rightarrow (b_0, 0, b_1, 0, b_2, 0, b_3, 0, b_4, 0, b_5, 0, b_6, 0, b_7, 0).$$

To izvedemo po bytih. S pomočjo razširitvene tabele spremenimo vsak byte v dvojni byte. Nato redukcijo elementa $(a'_0, a'_1, \ldots, a'_{2(m-1)})$ izvedemo z besednimi operacijami, če je stopnja polinoma $f(x) - x^m$ največ $m - w$, kjer je $w$ dolžina besede. Implementacijo kvadriranja z umbralnimi bazami izvedemo podobno. Izkaže se, da je kvadriranje v umbralnih bazah še hitrejše od kvadriranja v polinomskih bazah.

Poglejmo algoritem 11, s katerim kvadriramo element, predstavljen v umbralni bazi. Najprej vsakemu bitu iz prve polovice vektorja $(p_1, p_2, \ldots, p_m)$ dodamo ničelni bit. S

tem iz $m/2$ bitov dobimo $m$ bitov. S pomočjo Leme 2.4.6.(i) in pravila za množenje v umbralnih bazah razširimo drugo polovico $m$-terice in hkrati opravimo še redukcijo, byte za bytom (s prvo lastnostjo leme 2.4.6. dobimo $\beta_k \to \beta_{2k}$, nato pa s pravilom za množenje (3.4) reduciramo indeks). Tudi pri tem si lahko pomagamo z vnaprej pripravljenimi tabelami.

**ALGORITEM 11:** *Kvadriranje v umbralnih bazah*

INPUT: Umbralni polinom $p(\beta)$ stopnje največ $m$

OUTPUT: $p(\beta)^2$

1. $(p_1, p_2, \ldots, p_{\frac{m}{2}}, \ldots, p_m) \leftarrow (p_1, 0, p_2, 0, \ldots, 0, p_{\frac{m}{2}}, \ldots, p_m)$. {Razširi prvo polovico vektorja.}
2. $(p_1, 0, p_2, 0, p_3, \ldots, 0, p_{\frac{m}{2}}, \ldots, p_m) \leftarrow (p'_1, p'_2, \ldots, p'_m)$ {Razširi in hkrati reduciraj drugo polovico vektorja.}
3. Return $(p'(\beta))$

## MOŽENJE

Primerjajmo množenje v umbralnih bazah z množenjem v polinomskih bazah. Najprej razdelimo postopek na dva dela, kot v primeru kvadriranja. Prvi del bo navadno množenje polinomov, drugi pa redukcija. Za množenje v polinomskih bazah uporabimo enačbo (3.5), pri umbralnih bazah pa enačbo (3.4). V polinomskih bazah je produkt $2m$-terica, v umbralnih pa $3m$-terica (indeksi tečejo od $-m+1$ do $2m$). Redukcijo pri polinomskih bazah izvedemo enako kot pri kvadriranju, pri umbralnih bazah pa najprej zreduciramo $3m$-terico na $2m$-terico po Lemi 2.4.6(iii), oziroma po enačbi $\beta_{-h} = \beta_{2m+1-h}$ za $m < h \leq 2m+1$, da se iznebimo prve tretjine (indeksi od $-m+1$ do 0).

Redukcije nam ni treba narediti čisto na koncu. Kot pri Algoritmu 1 rekurzivno izračunamo $\beta_i b(\beta)$ iz $\beta_{i-1}$, ki je shranjen kot lb = levi-pomik-za-$i$, pri čemer skrajno levi bit izgine ($\beta_0 = 0$), in rb = desni-pomik-za-$i$, pri čemer skrajno desni bit prištejemo novemu skrajno desnemu bitu ($\beta_{m+1} = \beta_m$), samo z dvema pomikoma in z redukcijo dveh bitov.

**ALGORITEM 12.** *Množenje v umbralnih bazah po metodi pomakni-in-prištej*
INPUT: Umbralna polinoma $a(\beta)$ in $b(\beta)$ stopnje največ $m$.
OUTPUT: Umbralni polinom $c(\beta) = a(\beta) \cdot b(\beta)$ stopnje največ $m$.

1. $c \leftarrow 0, \texttt{lb} \leftarrow b, \texttt{rb} \leftarrow b$.
2. For $i$ from 1 to $m$ do
   2.1 $\texttt{lb} \leftarrow$ levi-pomik($\texttt{lb}$) in $\texttt{rb} \leftarrow$ desni-pomik($\texttt{rb}$) mod $(\beta_{m+1} = \beta_m)$.
   2.2 If $a_i = 1$ then $c \leftarrow c + \text{lb} + \texttt{rb}$.
3. Return($c$).

Tako kot zgornji algoritem, lahko tudi ostale algoritme za množenje v polinomskih bazah prevedemo na umbralne baze. Najbolj učinkovit je bil algoritem 4.

## INVERZ

Algoritem za izračun multiplikativnega inverza za umbralne baze bomo izpeljali iz razširjenega Evklidovega algoritma, kot pri računanju inverza v polinomskih bazah. Za izvedbo Evklidovega algoritma potrebujemo naslednje:

- moramo znati zmanjšati stopnjo,
- imeti moramo enoto za množenje in
- imeti moramo nerazcepni polinom.

Za implementacijo inverza v umbralnih bazah si bomo pomagali s trditvijo 2.4.7. Naj bo $a(\beta)$ poljuben element iz $\mathbb{F}_{2^m}$, predstavljen v umbralni bazi. Preden začnemo izvajati razširjen binaren Evklidov algoritem v umbralnih bazah, zreduciramo umbralno stopnjo $a(\beta)$ na največ $m-1$, z nerazcepnim polinomom iz zgornje trditve $(f(\beta) = \beta^m + \beta^{m-1} + \cdots + \beta + 1)$. Potem izvedemo običajni razširjeni binarni Evklidov algoritem kot pri polinomski bazi, vendar s pravilom množenja v umbralnih bazah (formula (3.4)). Ko dobimo inverzni element, preverimo konstantni člen. Če je različen od nič, je enak enoti za množenje in ga po trditvi 2.4.7(ii) nadomestimo z $\sum_{i=1}^{m} \beta_i$. Sedaj je rezultat umbralni polinom stopnje največ $m$ in brez konstantnega člena, kot smo želeli.

**ALGORITEM 13.** *Inverz v umbralnih bazah*

INPUT: Umbralni polinom $a(\beta) \in \mathbb{F}_{2^m}$.

OUTPUT: Inverz $a(\beta)^{-1}$.

1. Redukcija $a(\beta)$ s polinomom $1 + \sum_{i=1}^{m} \beta_i$.
2. Razširjeni Evklidov algoritem (kot algoritem 8, le s pravilom za množenje 3.4).
3. If $b_0 = 1$ then $b(\beta) \leftarrow r(\beta)$. ($b(\beta)$ nadomestimo s polinomom $r(\beta)$ iz Trditve 2.4.7 (ii).
4. Return($b(\beta)$).

Naredimo primer za računanje inverza v umbralnih bazah.

**Primer:** Naj bo $m = 11$. Potem je nerazcepni polinom iz zgornje trditve enak:

$$f(\beta) = \beta_{11} + \beta_{10} + \beta_9 + \beta_8 + \beta_7 + \beta_6 + \beta_5 + \beta_4 + \beta_3 + \beta_2 + \beta_1 + 1.$$

Poiščimo inverz elementa $a(\beta) = \beta_{11} + \beta_{10} + \beta_7 + \beta_6 + \beta_3 + \beta_2$. Stopnja $a$ je 11, torej jo moramo zreducirati s polinomom $f(\beta)$. Nadomestimo člen $\beta_{11}$ v elementu $a(\beta)$ s polinomom $\beta_{10} + \beta_9 + \beta_8 + \beta_7 + \beta_6 + \beta_5 + \beta_4 + \beta_3 + \beta_2 + \beta_1 + 1$. Sedaj je umbralna stopnja elementa $a(\beta)$ manjša od umbralne stopnje polinoma $f(\beta)$:

$$a(\beta) = \beta_9 + \beta_8 + \beta_5 + \beta_4 + \beta_1 + 1.$$

Lahko začnemo izvajati Evklidov algoritem, kot pri polinomskih bazah, le pravilo za množenje nadomestimo s pravilom za množenje v umbralnih bazah (tj. $\beta_i\beta_j = \beta_{i+j} + \beta_{|i-j|}$):

$$\begin{aligned}
r_{-2} &= \beta_{11} + \cdots + \beta_1 + 1 = \\
&= \underline{(\beta_2 + 1)}(\beta_9 + \beta_8 + \beta_5 + \beta_4 + \beta_1 + 1) + \beta_7 + \beta_6 + \beta_3 + \beta_2 + \beta_1 = s_0 r_{-1} + r_0 \\
r_{-1} &= \beta_9 + \beta_8 + \beta_5 + \beta_4 + \beta_1 + 1 = \\
&= \underline{\beta_2}(\beta_7 + \beta_6 + \beta_3 + \beta_2 + \beta_1) + \beta_5 + \beta_4 + \beta_3 + \beta_1 + 1 = s_1 r_0 + r_1 \\
r_0 &= \beta_7 + \beta_6 + \beta_3 + \beta_2 + \beta_1 = \\
&= \underline{(\beta_2 + 1)}(\beta_5 + \beta_4 + \beta_+ 3 + \beta_1 + 1) + \beta_4 + \beta_2 + 1 = s_2 r_1 + r_2 \\
r_1 &= \beta_5 + \beta_4 + \beta_3 + \beta_1 + 1 = \\
&= \underline{(\beta_1 + 1)}(\beta_4 + \beta_2 + 1) + \beta_3 + \beta_2 + 1 = s_3 r_2 + r_3 \\
r_2 &= \beta_4 + \beta_2 + 1 = \\
&= \underline{(\beta_1 + 1)}(\beta_3 + \beta_2 + 1) + 1 = s_4 r_3 + r_2
\end{aligned}$$

Razširjeni del je enak:

$$\begin{aligned}
s_0b_{-1}+b_{-2} &= \underline{(\beta_2+1)}\cdot 1+0=\beta_2+1=b_0(\beta)\\
s_1b_0+b_{-1} &= \underline{\beta_2}(\beta_2+1)+1=\beta_4+\beta_2+1=b_1(\beta)\\
s_2b_1+b_0 &= \underline{(\beta_2+1)}(\beta_4+\beta_2+1)+\beta_2+1=\beta_6=b_2(\beta)\\
s_3b_2+b_1 &= \underline{(\beta_1+1)}\beta_6+\beta_4+\beta_2+1=\beta_7+\beta_6+\beta_5+\beta_4+\beta_2+1=b_3(\beta)\\
s_4b_3+b_2 &= \underline{(\beta_1+1)}(\beta_7+\beta_6+\beta_5+\beta_4+\beta_2+1)+\beta_6=\beta_8+\beta_5+\beta_2+1=b_4(\beta)
\end{aligned}$$

Dobili smo

$$b_4(\beta)=\beta_8+\beta_5+\beta_2+1 \;=\; \beta_{11}+\beta_{10}+\beta_9+\beta_7+\beta_6+\beta_4+\beta_3+\beta_1=a(\beta)^{-1}.$$

Za preiskus zmnožimo $a(\beta)$ in $a(\beta)^{-1}$. Upoštevamo pravilo za množenje v umbralnih bazah in lemo 2.4.6, po kateri velja:

$$\beta_{12}=\beta_{11},\ \beta_{13}=\beta_{10},\ \beta_{14}=\beta_9,\ \beta_{15}=\beta_8,\ \beta_{16}=\beta_7,\ \beta_{17}=\beta_6,$$

$$\beta_{18}=\beta_5,\ \beta_{19}=\beta_4,\ \beta_{20}=\beta_3,\ \beta_{21}=\beta_2,\ \beta_{22}=\beta_1.$$

Dobimo:

$$(\beta_9+\beta_8+\beta_5+\beta_4+\beta_1+1)(\beta_8+\beta_5+\beta_2+1)=1,$$

kot smo želeli. ◊

## 3.5 MNOŽENJE PO BITIH

Za konec poglejmo idejo množenja po bitih. Elwyn Berlekamp je razvil algoritem za množenje po bitih nad $\mathbb{F}_{2^m}$ za dekodiranje Reed-Solomonovih kod. Ta algoritem je izredno učinkovit, saj zahteva malo korakov. Njegova slabost pa je, da zahteva kar dve bazni transformaciji, saj je eden od faktorjev v primarni, drugi pa v dualni bazi in rezultat je spet v dualni bazi. V nekaterih primerih je transformacija med bazami enostavna, v splošnem pa ne. Seveda pa se s sebidualnimi bazami izognemo pretvorbam med bazami in v tem primeru je množenje po bitih izredno uporabno. Mi se ne bomo podrobneje ukvarjali s tem algoritmom, saj bi potrebovali dosti časa za natančnejše preučevanje. To pa presega okvir te naloge.

Poglejmo si strukturo Berlekampovega algoritma za množenje po bitih nad obsegom $\mathbb{F}_{2^m}$. Naj bo $\bar{\alpha}=\{\alpha_0,\alpha_1,\ldots,\alpha_{m-1}\}$ baza za obseg $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$. Vemo že, da je pripadajoča dualna baza množica $\bar{\beta}=\{\beta_0,\beta_1,\ldots,\beta_{m-1}\}\subset\mathbb{F}_{2^m}$, tako da velja

$$\mathrm{Tr}(\alpha_i\beta_j)=\begin{cases}1 & \text{za } i=j\\ 0 & \text{za } i\neq j.\end{cases}$$

Dokazali smo že, da dualna baza obstaja in je enolično določena, spomnimo se le, kako jo poiščemo. Definirajmo matriko $A = (a_{ij})_{i,j=0}^{m-1}$ dimenzije $m \times m$ nad $\mathbb{F}_2$ takole:

$$a_{ij} = \mathrm{Tr}(\alpha_i \alpha_j),$$

in naj bo $B = A^{-1}$. Označimo $(k, j)$-to vrednost v matriki $B$ z $b_{kj}$. Potem je za $j = 0, \ldots, m-1$ dualna baza $\bar{\beta}$ podana z

$$\beta_j = \sum_{k=0}^{m-1} b_{kj} \alpha_k. \tag{3.6}$$

Matriki $A$ in $B$ lahko uporabimo za transformacijo koordinat iz primarne baze $\bar{\alpha}$ v dualno bazo $\bar{\beta}$. Naj bo $x \in \mathbb{F}_{2^m}$ in naj bo

$$x = \sum_{i=0}^{m-1} x_i \alpha_i,$$

zapis elementa $x$ v primarni bazi,

$$x = \sum_{i=0}^{m-1} x'_j \beta_j, \tag{3.7}$$

pa njegov zapis v dualni bazi. Elementi $x_i$ in $x'_j$ zavzemajo vrednosti 0 ali 1. Zapišimo binarna vektorja $x$ in $x'$ takole:

$$x = (x_0, x_1, \ldots, x_{m-1})^T$$

$$x' = (x'_0, x'_1, \ldots, x'_{m-1})^T.$$

Hitro lahko preverimo, da velja

$$x' = Ax, \;\; \text{kjer} \;\; x'_j = \mathrm{Tr}(x\alpha_j) \tag{3.8}$$

in

$$x = Bx', \;\; \text{kjer} \;\; x_i = \mathrm{Tr}(x'\beta_i). \tag{3.9}$$

V nadaljevanju poglavja bomo obravnavali le baze oblike $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$, kjer je $\alpha$ primitiven element v $\mathbb{F}_{2^m}$. Pripadajočo dualno bazo še naprej označujmo z $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$. Glavna motivacija za nas je, da je množenje $x$ z $\alpha$ enostavno v dualnem koordinatnem sistemu. Poglejmo si, zakaj je tako. Izrazimo $x$ v dualnih koordinatah:

$$x = x'_{m-1}\beta_{m-1} + x'_{m-2}\beta_{m-2} + \cdots + x'_0\beta_0.$$

Potem za $j = 0, 1, \ldots, m-1$ dobimo

$$x'_j = \mathrm{Tr}(x \cdot \alpha^j).$$

Kaj so komponente za $\alpha x$ v dualnem koordinatnem sistemu? Iz enačbe (3.6) dobimo

$$(\alpha x)'_j = \mathrm{Tr}(\alpha x \cdot \alpha^j) = \mathrm{Tr}(x \cdot \alpha^{j+1}).$$

Če primerjamo enačbi (3.7) in (3.8), vidimo naslednje:

$$(\alpha x)'_j = \begin{cases} x'_{j+1}; & j = 0, 1, \ldots, m-2; \\ \mathrm{Tr}(x \cdot \alpha^m); & j = m-1. \end{cases}$$

Torej lahko množimo spremenljivko $x$ in konstanto $\alpha$ samo s pomičnim registrom. Izračunati moramo le $\mathrm{Tr}(x\alpha^m)$. Rezultat je seveda v dualnih koordinatah. Spomnimo se še, da seštevanja izvajamo po modulu 2.
Sedaj znamo množiti spremenljivko $x$ s konstanto $\alpha$. Kako pa bi množili ostale elemente iz obsega? Podrobno si poglejmo vsebino $i$-tega registra pri računanju produkta $x\alpha$. Označimo jo z $\mathbf{vsebina}_i(t)$. Za $t = 0$ vsebuje $i$-ti register $i$-ti bit elementa $x$, kar je po enačbi (3.6) enako $\mathrm{Tr}(\alpha^i x)$:

$$\mathbf{vsebina}_i(0) = \mathrm{Tr}(\alpha^i x).$$

Pri $t = 1$, vsebuje pomični register $\alpha x$, torej

$$\mathbf{vsebina}_i(1) = \mathrm{Tr}(\alpha^i \cdot \alpha x) = \mathrm{Tr}(\alpha \cdot \alpha^i x).$$

V splošnem velja za $0 \leq t \leq m-1$:

$$\mathbf{vsebina}_i(t) = \mathrm{Tr}(\alpha^i \cdot \alpha^t x) = \mathrm{Tr}(\alpha^t \cdot \alpha^i x).$$

Oziroma, če zapišemo z besedo:

$$\mathbf{vsebina}_i(t) = t\text{-ta komponenta za } \alpha^i x. \tag{3.10}$$

Zato, če gledamo $i$-ti register, bomo videli bite v $\alpha^i \cdot x$ enega za drugim; to pomeni, da bo po $t$ ciklih vseboval $i$-ti register $t$-to komponento od $\alpha^i \cdot x$ (v dualnih koordinatah). Ker lahko vsak element iz $\mathbb{F}_{2^m}$ zapišemo kot linearno kombinacijo elementov $1, \alpha, \ldots, \alpha^{m-1}$, nam enačba (3.9) pove, kako množimo $x$ s poljubno konstanto $\mu$. Sedaj imamo dve zanki. Ena je enaka kot v primeru množenja $x\alpha$ (določena je z elementom $\alpha$), druga pa je odvisna od konstante $\mu$. Kako pa množimo spremenljivko $x$ s spremenljivko $y$? Če je $y$ izražen v primarnih koordinatah kot

$$y = y_{m-1}\alpha^{m-1} + y_{m-2}\alpha^{m-2} + \cdots + y_0 \cdot 1,$$

potem velja naslednje:

$$\sum_{i=0}^{m-1} y_i \mathbf{vsebina}_i(t) = \sum_{i=0}^{m-1} y_i \cdot (\alpha^i x)'_t =$$

$$= (x \cdot \sum y_i \alpha^i)'_t = (x \cdot y)'_t.$$

Pomagali smo si z enačbama (3.9) in (3.10). Če hranimo primarne koordinate za $y$ v $m$ stopenjskem pomičnem registru, bo torej po $t$ ciklih produkt registrov za $x$ in $y$ enak $t$-temu bitu produkta v dualnih koordinatah. V register za $x$ shranimo $m$ bitov $x'_{m-1}, \ldots, x'_0$ spremenljivke $x$ v dualnih koordinatah in v register za $y$ shranimo $m$ bitov $y_{m-1}, \ldots, y_0$ spremenljivke $y$ v primarnih koordinatah. Po $t$ ciklih bo na izhodu $t$-ti bit produkta $xy$, izražen v dualnih koordinatah.

Žal imamo elemente izražene v dveh različnih bazah, kar je lahko kar velik problem, saj bi pri večjih sistemih potrebovali poseben algoritem za pretvarjanje baz. Spet se spomnimo, da se problemu pretvarjanja med bazami izognemo, če uporabimo sebidualne baze. Obstajajo tudi nekateri drugi primeri, ko je pretvorba med bazami lahko izvedljiva.

# Poglavje 4

# ANALIZA IN PRIMERJAVA ALGORITMOV

Kot smo videli, se algoritmi precej razlikujejo glede na to, katero bazo smo uporabili za predstavitev elementov iz končnega obsega. Pri vsakem algoritmu smo skušali čimbolj izkoristiti lastnosti in obliko baze. Žal prinaša vsaka baza tudi nekatere slabe strani in zaradi tega je težko takoj odgovoriti na vprašanje, v kateri bazi je računanje najbolj učinkovito. To je odvisno od same aplikacije, oziroma koliko katerih operacij z elementi končnih obsegov bomo pri implementaciji algoritmov potrebovali. V našem primeru bo seveda poudarek na algoritmih za aritmetiko nad eliptičnimi krivuljami. V tem poglavju bomo naredili primerjavo algoritmov glede na število osnovnih operacij, ki jih zahtevajo algoritmi za seštevanje, množenje, kvadriranje in računanje inverza. Mi bomo šteli operacije nad besedami, torej nas bo zanimala implementacija algoritmov v softwaru.

Izvajanje osnovnih operacij lahko obravnavamo s povsem temeljnega vidika, to je kako se izvajajo nad biti. Bite lahko strnemo v besede in preučujemo izvajanje operacij s samimi besedami. Več besed pa sestavlja vektor. Osnovni dve operaciji z vektorji sta vektorski pomik in seštevanje dveh vektorjev. Kot bomo videli, potrebujemo za ti dve operaciji nekaj več osnovnih operacij na besednem nivoju.

Poglavje je organizirano takole. V prvem razdelku bomo preučili delovanje osnovnih operacij nad besedami. S temi izsledki si bomo pomagali v nadaljevanju poglavja pri analizi algoritmov, ki smo jih spoznali v tretjem poglavju. Držali se bomo enakega zaporedja algoritmov, torej bomo v drugem razdelku prešteli osnovne operacije pri algoritmih za polinomske baze, v tretjem za normalne baze in nato še za umbralne baze. Na koncu bomo glede na rezultate analize naredili primerjavo med algoritmi in skušali potegnili osnovne zaključke.

# 4.1 OSNOVNE BESEDNE OPERACIJE

Pri vseh algoritmih se kot osnovni operaciji pojavljata vektorski pomik in seštevanje dveh vektorjev. Zato je smiselno ti operaciji posebej preučiti. Tako se bomo lahko pri analizi algoritmov za osnovne operacije opirali na te ugotovitve. Vektorski pomik izvedemo z osnovno besedno operacijo za pomik za $i < w$ mest, kjer je $w$ dolžina besede. Ker se ukvarjamo z binarnimi obsegi, dva vektorja seštejemo z osnovno besedno operacijo ekskluzivnim-ali (XOR). Spet bomo dolžino besed označili z $w$, število besed v vektorju dolžine $m$ pa bomo označili s $t = \lceil m/w \rceil$.

Preden se lotimo štetja osnovnih besednih operacij, ki jih zahtevata ciklični pomik vektorja (shift) in seštevanje dveh nizov (XOR), se malo pomudimo pri tabelah, ki jih bomo uporabljali. Gre predvsem za maske, ki nam omogočajo priti do posameznih bitov ali delov besed. Maskirno besedo bomo prvič potrebovali že pri pomiku vektorja. Na primer, videli bomo, da moramo pri desnem pomiku za eno mesto preveriti vrednost skrajno desnega bita v besedi, saj ga moramo prenesti v naslednjo besedo. Do skrajno desnega bita pridemo z AND besedno operacijo z maskirno besedo oblike $00\ldots01$. Pri desnem pomiku za dve mesti potrebujemo besedo $00\cdots011$ in tako dalje. Dolžina maskirnih besed je enaka dolžini ostalih besed, torej $w$. Pri levih pomikih seveda potrebujemo maskirne besede, kjer se enice širijo z leve. Zato si že vnaprej pripravimo tabeli `dmaska` in `lmaska`. Zapisali smo še tabelo `bmaska`, ki nam pomaga v besedi poiskati točno določen bit.

| | | | | | |
|---|---|---|---|---|---|
| | $w$ | | $w$ | | |
| `bmaska`$[1]$ | $= \overbrace{1000\ldots000}$ | `lmaska`$[1]$ | $= \overbrace{1000\ldots000}$ | `dmaska`$[1]$ | $= 0111\ldots111$ |
| `bmaska`$[2]$ | $= 0100\ldots000$ | `lmaska`$[2]$ | $= 1100\ldots000$ | `dmaska`$[2]$ | $= 0011\ldots111$ |
| `bmaska`$[3]$ | $= 0010\ldots000$ | `lmaska`$[3]$ | $= 1110\ldots000$ | `dmaska`$[3]$ | $= 0001\ldots111$ |
| | $\vdots$ | | $\vdots$ | | $\vdots$ |
| `bmaska`$[w-1]$ | $= 0000\ldots010$ | `lmaska`$[w-1]$ | $= 1111\ldots110$ | `dmaska`$[w-1]$ | $= 0000\ldots001$ |
| `bmaska`$[w]$ | $= 0000\ldots001$ | `lmaska`$[w]$ | $= 1111\ldots111$ | `dmaska`$[w]$ | $= 0000\ldots000$ |

Tabela 1. Maskirne tabele za iskanje bita (`bmaska` ) ter za levi (`lmaska` ) in desni (`dmaska`) pomik.

Podobno si tudi pri obračanju vektorjev lahko pomagamo z vnaprej pripravljeno tabelo. Za vse različne byte (skupno jih je $2^8$) izračunamo njihovo obrnjeno verzijo. Rezultat je zapisan v spodnji tabeli pod imenom `obrni_byte`. Pri kvadriranju moramo byte spremeniti v dvojne byte tako, da vsakemu bitu dodamo en ničelni bit. V ta namen si pripravimo tabelo `kvadr_byte`.

| | BYTE | | BYTE |
|---|---|---|---|
| `obrni_byte`[00000000] | = | 00000000 | |
| `obrni_byte`[00000001] | = | 10000000 | |
| `obrni_byte`[00000010] | = | 01000000 | |
| `obrni_byte`[00000011] | = | 11000000 | |
| `obrni_byte`[00000100] | = | 00100000 | |
| | | ⋮ | |
| `obrni_byte`[11111101] | = | 10111111 | |
| `obrni_byte`[11111110] | = | 01111111 | |
| `obrni_byte`[11111111] | = | 11111111 | |

| | BYTE | | DVOJNI BYTE |
|---|---|---|---|
| `kvadr_byte`[00000000] | = | 0000000000000000 | |
| `kvadr_byte`[00000001] | = | 0000000000000001 | |
| `kvadr_byte`[00000010] | = | 0000000000000100 | |
| `kvadr_byte`[00000011] | = | 0000000000000101 | |
| `kvadr_byte`[00000100] | = | 0000000000001000 | |
| | | ⋮ | |
| `kvadr_byte`[11111101] | = | 0101010101010001 | |
| `kvadr_byte`[11111110] | = | 0101010101010100 | |
| `kvadr_byte`[11111111] | = | 0101010101010101 | |

Tabela 2. Tabeli za obračanje bytov `obrni_byte` in razširjanje bytov `kvadr_byte` .

Za pripravo zgornjih tabel seveda potrebujemo nekaj časa, vendar si jih pripravimo vnaprej in njihova priprava ne vpliva na čas pri izvajanju posameznih algoritmov. Zgornje tabele nam tudi poberejo nekaj prostora. V prvem primeru imamo tabele velikosti $w \times w$. Ker je dolžina besed navadno enaka 32, imamo tabele velikosti $2^{10} \approx 10^3$. V drugem primeru so tabele velikosti $2^8 \times 8 = 2^{11} \approx 2 \cdot 10^3$ oziroma $2^8 \times 16 \approx 4 \cdot 10^3$. Podatki reda nekaj kilo bitov so sprejemljivi glede na zmogljivosti današnjih računalnikov. Če pa bi, na primer, hoteli pripraviti tabele za dvojne byte, bi dobili tabelo velikosti $2^{20} \approx 10^6$. To je že reda velikosti mega bitov. Takšni podatki pa že predstavljajo večji problem. Zato se omejimo na tabele za byte.
Mi delamo z besedami velikosti 32 bitov, torej so sestavljene iz štirih bytov. Do posameznega byta v besedi si pomagamo z besednimi pomiki. Na primer, če hočemo priti do drugega byta v besedi, najprej pomaknemo besedo za 8 bitov v levo, da se iznebimo prvega byta. Nato pomaknemo besedo za 24 bitov (3 byte) v desno, da se iznebimo še desnih dveh bytov. Ko pomaknemo besedo za 16 bitov (2 byta) nazaj v levo, smo dobili besedo, kjer so prvi, tretji in četrti byte enaki nič, drugi byte pa je ostal nespremenjen. Za to smo porabili tri besedne pomike. Drugi način je, da si vnaprej pripravimo besede, kjer bi bile pri posameznih bytih same enice ali same ničle. Na primer, če spet pogledamo primer, ko hočemo doseči drugi byte. Prištejemo maskirno besedo (AND), ki ima v prvem, tretjem in četrtem bytu same ničle v drugem bytu pa same enice ($00000001111111100 \ldots 00$). S tem dobimo ničle v prvem, tretjem in četrtem bytu, v drugem pa so vrednosti ostale nespremenjene.

Pomik vektorja dolžine $m$ za eno mesto oziroma računanje produkta $x \cdot a(x)$, kjer je $a$ vektor dolžine $m$, poteka takole. Pomik izvedemo z besednimi operacijami. Najprej pomaknemo besedo za eno mesto, kar nas stane 1 besedni pomik. Pri tem si moramo zapomniti skrajno desni bit, saj ga moramo prenesti v naslednjo besedo. Skrajni bit preverimo z AND operacijo z maskirno besedo `dmaska`$[\mathrm{w}-1] = 00 \ldots 01$, saj 0 AND karkoli vrne 0, medtem ko 1 AND karkoli vrne vrednost drugega. Za to porabimo torej 1 AND besedno operacijo. Če je skrajni bit enak nič, smo

končali s to besedo. Sicer pa moramo z naslednjo besedo izvesti XOR operacijo z maskirno besedo $10\ldots00$ (na prvo mesto vpišemo 1). Ker imamo maskirne besede že pripravljene, nam jih ni treba na novo generirati in ne tratimo dodatnega časa. Ciklični pomik zahteva za vsako besedo 1 pomik in 1 AND operacijo, v primeru da je skrajni bit enak 1 pa še 1 XOR operacijo. Skupaj torej zahteva en pomik vektorja dolžine $m$, oziroma $t$ besednega vektorja, $t$ pomikov besed, $t$ besednih AND operacij in v najslabšem primeru $t$ XOR besednih operacij (skrajni bit je pri vsaki besedi enak ena), v pričakovanem pa $t/2$ XOR operacij ( skrajni bit je ena v vsaki drugi besedi). Pri pomikih za $i < w$ mest v desno ravnamo po enakem postopku. Vsakič si moramo zapomniti skrajnih $i$ bitov, ki jih preverimo z maskirno besedo `dmaska`[i] (1 AND). V pričakovanem primeru bo vedno potrebno izvesti še 1 XOR operacijo (samo v primeru, da je skrajnih $i$ bitov enakih 0, ni potrebno). Skupno torej vektorski pomik za $i$ mest v desno zahteva $t$ besednih pomikov, $t$ besednih AND operacij in $t$ besednih XOR operacij. Z maskirnimi besedami izvajamo AND in XOR operacije. Ker obe zahtevata enako časa, jih bomo navadno šteli kar skupaj.

Dva vektorja dolžine največ $m$ seštejemo na naslednji način. Spet sta vektorja razdeljena na besede dolžine $w$. Dve besedi seštejemo z 1 besedno XOR operacijo. Ker je skupno $t$ besed, potrebujemo za vsoto dveh vektorjev dolžine $m$ skupno $t$ XOR besednih operacij.

K osnovnim operacijam nad besedami spada tudi prirejanje vrednosti vektorju dolžine $m$. Vsaki besedi posebej moramo dodeliti neko vrednost, torej porabimo po eno operacijo za vsako besedo. Za cel vektor potrebujemo $t$ besednih operacij. To je malo v primerjavi z ostalimi operacijami, pa tudi v vseh algoritmih imamo približno enako število prirejanj, zato jih bomo običajno zanemarili.

Od osnovnih operacij nad vektorji sta torej vektorsko seštevanje in vektorsko prirejanje dokaj poceni, medtem ko vektorski pomik zahteva nekoliko več časa (več kot dvakrat toliko kot ostali dve operaciji). Pri algoritmih, ki jih bomo analizirali, bomo posebej prešteli vektorske pomike in vektorska seštevanja. Vektorska prirejanja bomo v glavnem izpuščali. Glede na rezultat in na zgornje ugotovitve bomo lahko naredili primerjavo med algoritmi. Strnimo ugotovitve v tabeli.

| BESEDNE OP. | SHIFT | AND | XOR | prir. | skupaj |
|---|---|---|---|---|---|
| POMIK za 1 | $t$ | $t$ | $t$ | 2t | $5t$ |
| POMIK za $i$ | $2t$ | 0 | $2t$ | $2t$ | $6t$ |
| SEŠTEVANJE | 0 | 0 | $t$ | $3t$ | $4t$ |
| PRIREJANJE | 0 | 0 | 0 | $2t$ | $2t$ |

Tabela 3. Število operacij za osnovne besedne operacije, kjer $t = \lceil m/w \rceil$ pomeni število besed dolžine $w$ v vektorju dolžine $m$.

Tudi for stavki niso tako trivialni, kot se zdi. Na splošno bomo pri for stavkih

šteli le število zank, čeprav izvajanje for stavka potrebuje nekaj dodatnega časa. Pri vsaki zanki se mora namreč indeks, ki šteje število ponovitev, povečati in nato primerjati s pogojem iz for stavka. Vse to bomo zanemarili, saj obstajajo načini, pri katerih računalnik izvaja for zanke drugače. Na primer pri metodi, imenovani 'loop unrolling', si vnaprej pripravi registre tako, da mu dejansko ni treba pri vsaki zanki brati in večati indeksov ter preverjati pogojev. Dodaten čas pri izvajanju for stavka lahko zanemarimo tudi zato, ker imamo pri vseh algoritmih približno enako število for stavkov.

## 4.2 ALGORITMI ZA POLINOMSKE BAZE

S polinomskimi bazami smo dobili izredno enostavno implementacijo, zlasti ko za razcepni polinom izberemo trinom ali pentonom. Algoritmi za seštevanje, redukcijo in kvadriranje so precej enostavni, medtem ko sta algoritma za množenje in inverz bolj zapletena. V tem razdelku bomo naredili podrobno analizo algoritmov. Ugotovili bomo, katere operacije se splača šteti in katere se sme zanemariti. Kot se bo izkazalo, bodo določene operacije zahtevale občutno več časa kot ostale. V tem razdelku bomo upoštevali vse operacije, v naslednjih pa bomo občutno cenejše operacije zanemarili.

Ker obravnavamo obsege $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$, so elementi vektorji iz ničel in enic dolžine $m$. Omenili smo že, da dva vektorja, predstavljena v polinomski bazi, pa tudi ostalih obravnavanih bazah, seštejemo tako, da po besedah izvedemo navadno XOR operacijo (ekskluzivni-ali). O tej operaciji smo posebej govorili v prejšnjem razdelku. Ugotovili smo, da za vsoto dveh vektorjev porabimo $t$ besednih XOR operacij. Torej je seštevanje v primerjavi z drugimi operacijami izredno poceni.

### MNOŽENJE

V tretjem poglavju smo najprej spoznali ALGORITEM 1, s katerim zmnožimo dva elementa po metodi pomakni-in-prištej, kjer jemljemo bite z desne proti levi. Vhodna podatka sta dva binarna polinoma $a(x)$ in $b(x)$ stopnje največ $m-1$ oziroma vektorja dolžine $m$, rezultat pa je njun produkt $c(x)$ po modulu polinoma $f(x)$. Po korakih preštejmo operacije.

1. Primerjava $a_0 = 1$ v if stavku se nanaša zgolj na en bit v nizu $a$, torej zahteva zgolj eno bitno operacijo. To je seveda zanemarljivo v primerjavi z drugimi operacijami. Vektorju $c$ priredimo vrednost ničelnega vektorja z enim vektorskim prirejanjem.

2. For zanko izvedemo $(m-1)$-krat.

2.1 Vektorju $b$ priredimo vrednost produkta $b(x) \cdot x$. Množenje polinoma $b(x)$ z $x$ zahteva 1 vektorski pomik. Sledi redukcija po modulu redukcijskega polinoma. Vsakič moramo zreducirati polinom samo za eno stopnjo, saj je po vektorskem pomiku polinom $b(x)$ stopnje največ $m$. Problem je torej samo pri členu $x^m$. Najprej poglejmo primer, ko je redukcijski polinom nerazcepni trinom $f(x) = x^m + x^k + 1$. Člen $x^m$ nadomestimo z $x^m = x^k + 1$. Popravljati moramo torej dve besedi. Besedi na sredini niza (za člen $x^k$) prištejemo maskirni niz (1 XOR). V skrajni besedi potrebujemo še 1 XOR operacijo z maskirnim nizom (zadnji bit postane enak 1). Skupno potrebujemo torej 2 XOR besedni operaciji. Če je redukcijski polinom nek poljuben nerazcepen polinom $f(x) = x^m + x^{k_1} + x^{k_2} + \cdots + x^{k_i} + 1$, spet upoštevamo $x^m = x^{k_1} + x^{k_2} + \cdots + x^{k_i} + 1$. Vektor $b$ moramo popravljati na $i + 1$ mestih. Potrebujemo približno $i + 1$ XOR besednih operacij. To je v splošnem primeru kar zahteven problem, saj je lahko število $i$ relativno veliko ($1 \leq i \leq m - 1$). Seveda lahko bita, ki sta v isti besedi, popravljamo hkrati in prihranimo na času.

2.2 Primerjava v if stavku ($a_i = 1$) se spet nanaša zgolj na en bit, kar je zanemarljivo v primerjavi z ostalimi operacijami. V najslabšem primeru bo pogoj v if stavku vedno resničen in bomo morali vsakič prirediti vektorju $c$ vrednost vsote $b + c$ in sešteti vektorja $b$ in $c$ (1 vektorsko seštevanje). V pričakovanem pa bo pogoj resničen v vsaki drugi zanki in bomo vektorja $a$ in $b$ sešteli vsakič drugič.

3. Return $C$ sicer zahteva 1 operacijo nad vektorjem, a je v skupni oceni algoritma ne bomo upoštevali, saj imamo enako operacijo prav v vseh algoritmih in se jo zdi nesmiselno dodatno omenjati. Pri analizah ostalih algoritmov jo bomo izpustili.

Sedaj seštejmo vse operacije po korakih. Skupno smo v primeru, da je redukcijski polinom trinom naredili $m - 1$ vektorskih pomikov. V najslabšem primeru imamo $2(m - 1) + 1 = 2m - 1$ vektorskih prirejanj in $(m - 1)(2 + t)$ besednih XOR, v pričakovanem pa $3(m - 1)/2 + 1$ vektorskih prirejanj in $(m - 1)(2 + t/2)$ besednih XOR operacij. Vektorskih seštevanj je v najslabšem primeru približno $m$, v pričakovanem pa približno $m/2$. Pri naslednjih algoritmih bomo prirejanja pri skupni oceni izpuščali, saj jih je dosti manj in so cenejša od ostalih dveh operacij. V vseh algoritmih jih je približno enako. Algoritem 1 ne zahteva dosti prostora. Za shranjevanje vseh treh polinomov, $a, b$ in $c$, potrebujemo po $m$ bitov.

V algoritmih 2, 3 in 4 smo izvajali samo množenje vektorjev, brez redukcije. V ALGORITMU 2 smo zapisali metodo glavnika, kjer smo jemali besede od desne proti levi. Vhodna podatka sta spet vektorja dolžine $m$, rezultat pa je vektor dolžine

največ $2m$. Poglejmo algoritem po korakih.

1. Vektorju $C$ priredimo ničelni vektor, za kar porabimo 1 vektorsko prirejanje na vektorju dolžine $2m$. To je toliko, kot 2 prirejanji na vektorju dolžine $m$.

2. For zanka se izvede $w$-krat (to je v našem primeru 32-krat).

   2.1 For zanka se izvede $t$-krat (za vsako besedo enkrat). Ceno primerjave $A[j](k) = 1$ lahko spet zanemarimo, saj se nanaša zgolj na en bit. Če je pogoj v if stavku resničen, seštejemo dva vektorja. V najslabšem primeru se bo to zgodilo vsakič, v pričakovanem pa vsakič drugič.

   2.2 Ceno primerjave $k \neq 31$ spet zanemarimo. Ta pogoj bo resničen v vseh primerih, razen v zadnji zanki. Torej imamo v vseh zankah 1 vektorski pomik $B \cdot x$, razen v eni (vektor $B$ je dolg $m$ bitov).

3. Return $C$ je v tem primeru nekaj večji, saj je vektor $C$ dolg $2m$ bitov. Vendar ta operacija zahteva malo dodatnega časa in jo bomo v prihodnje izpuščali.

Seštejmo glavni dve operaciji algoritma 2. Dobili smo $w - 1$ vektorskih pomikov. V najslabšem primeru imamo še $m$ vektorskih seštevanj, v pričakovanem pa $m/2$ vektorskih seštevanj. Algoritem 2 zahteva nekoliko več prostora od algoritma 1, saj je bil prej vektor $c$ dolg $m$, sedaj pa je dolg $2m$ bitov. Za vektorja $a$ in $b$ potrebujemo enako prostora, obakrat po $m$ bitov.

ALGORITEM 3 je podoben algoritmu 2, edina razlika je, da v tem primeru jemljemo besede z leve proti desni. Pojdimo po korakih.

1. Vektorju $C$ priredimo ničelni vektor, za kar porabimo 1 vektorsko prirejanje na vektorju dolžine $2m$. To je toliko, kot 2 prirejanji na vektorju dolžine $m$.

2. For zanka se izvede $w$-krat (navadno je $w = 32$).

   2.1 For zanka se izvede $t$-krat (za vsako besedo enkrat). Ceno primerjave $A[j](k) = 1$ lahko spet zanemarimo, saj se nanaša zgolj na en bit. Če je pogoj v if stavku resničen, seštejemo dva vektorja. V najslabšem primeru se bo to zgodilo vsakič, v pričakovanem pa vsakič drugič.

   2.2 Ceno primerjave $k \neq 31$ spet zanemarimo. Ta pogoj bo resničen v vseh primerih, razen v zadnji zanki. Torej imamo v vseh zankah 1 vektorski pomik $C \cdot x$, razen v eni. Za razliko od algoritma 2, ko se je vektorski pomik v koraku 2.2 nanašal na $m$ bitni vektor, se sedaj nanaša na $2m$ bitni vektor $C$. To lahko štejemo kot dva pomika vektorja dolžine $m$.

Skupaj smo dobili enako število seštevanj kot pri algoritmu 2, le vektorskih pomikov je dvakrat več, torej $2(w - 1)$. Potrebujemo tudi enako količino prostora, torej po $m$ bitov za vektorja $a$ in $b$ ter $2m$ bitov za vektor $C$.

Analizirajmo še časovno zahtevnost ALGORITMA 4, ki je izboljšava algoritma 3. Za množenje polinomov $a(x)$ in $b(x)$ stopnje največ $m-1$ smo uporabili metodo glavnika (od leve proti desni) z okni širine $v = 4$ (širina okna mora deliti dolžino besed $w$). Za hitrejše izvajanje vnaprej preračunamo produkte $u(x) \cdot b(x)$, za vse polinome $u(x)$ stopnje največ $v-1$ (ker jemljemo okna širine $v$). Spet po vrsti preštejmo operacije.

1. Vseh polinomov $u(x)$ stopnje največ $v-1$ je $2^v-1$ ( v našem primeru je $v = 4$, torej je vseh polinomov stopnje največ 3 $2^4 - 1 = 15$). Za njihovo generiranje porabimo $v-1$ besednih pomikov in $2^{v-1}$ seštevanj. Privzeli bomo, da imamo te polinome že vnaprej pripravljene in nas njihovo generiranje ne stane dodatnega časa. Za vse produkte $u(x) \cdot b(x)$ potrebujemo $v-1$ vektorskih pomikov in $3v-1$ vektorskih seštevanj. V našem primeru je $v = 4$, torej potrebujemo 3 vektorske pomike in 11 vektorskih seštevanj.

2. Vektorju $C$ priredimo ničelni vektor. Za to potrebujemo $2t$ besednih prirejanj, oziroma toliko kot za 2 prirejanji pri vektorju dolžine $m$.

3. For zanko izvedemo $\frac{w}{v}$-krat.

   3.1 For zanko izvedemo $t$-krat. Za generiranje vektorja $u$ potrebujemo $v$ bitnih operacij, kar bomo zanemarili. Vektor $B_u$ prištejemo k $C\{j\}$ z enim vekorskim seštevanjem.

   3.2 Pogoj v if stavku bo resničen v vsakem primeru, razen pri zadnjem izvajanju for zanke. Zato se vsakič, razen zadnjič, izvede še pomik vektorja $C$ za $v$ mest. Ker je vektor $C$ dolg $2m$ bitov, je to enako kot dva pomika vektorjev dolžine $m$.

Algoritem 4 zahteva $2\frac{w}{v} + v - 3$ vektorskih pomikov in $\frac{w}{v}\frac{m}{w} + 3v - 1 = \frac{m}{v} + 3v - 1$ vektorskih seštevanj. To pomeni v našem primeru, za $v = 4$, $\frac{w}{2} + 1$ vektorskih pomikov in $\frac{m}{4} + 11$ vektorskih seštevanj. Za vektorja $a$ in $b$ potrebujemo po $m$ bitov in za vektor $C$ $2m$ bitov prostora. Za vse produkte $B_u = u(x) \cdot b(x)$ potrebujemo dodatnih $2^v(m+v)$ bitov prostora, saj moramo shraniti $2^v$ različnih polinomov stopnje največ $m+v-1$. V našem primeru je $v = 4$, torej potrebujemo dodatnih $16(m+4)$ bitov.

Za lepši pregled zapišimo časovne in prostorske zahtevnosti algoritmov 2,3 in 4 v tabeli. Štejemo operacije z vektorji. Pomiki so približno trikrat dražji od seštevanja, kar upoštevamo v skupnem številu operacij. V primeru smo vzeli $m = 160$, $w = 32$, torej je $t = 5$, ter $v = 4$.

| Algoritmi za množenje | | ALGORITEM 2 | ALGORITEM 3 | ALGORITEM 4 |
|---|---|---|---|---|
| POMIK | najslab. | $w-1$ | $2(w-1)$ | $2\frac{w}{v}+v-3$ |
| | pričak. | $w-1$ | $2(w-1)$ | $2\frac{w}{v}+v-3$ |
| SEŠTEVANJE | najslab. | $m$ | $m$ | $\frac{m}{v}+3v-1$ |
| | pričak. | $m/2$ | $m/2$ | $\frac{m}{v}+3v-1$ |
| SKUPAJ | pričak. | $\frac{m}{2}+3w-3$ | $\frac{m}{2}+6w-6$ | $\frac{m+6w}{v}+6v-10$ |
| PRIMER | pričak. | 173 | 266 | 108 |
| PROSTORSKA ZAHT. | | $4m$ | $4m$ | $4m+2^v(m+v)$ |

Tabela 4. Časovna in prostorska zahtevnost algoritmov za množenje brez redukcije (algoritmi 2,3 in 4). Štejemo vektorske operacije, v prostorski zahtevnosti pa bite.

## REDUKCIJA

Z algoritmi 2,3 in 4 smo zmnožili polinoma $a(x)$ in $b(x)$, produkt $c(x)$ pa moramo še reducirati po modulu polinoma $f(x) = x^m + r(x)$. Polinom $c(x)$ je stopnje največ $2m-2$, saj sta polinoma $a(x)$ in $b(x)$ stopnje največ $m-1$. Redukcijski polinom $f(x)$ ima stopnjo $m$. ALGORITEM 5 izvaja redukcijo po en bit.

1. Za preračunanje vseh produktov $u_k(x) = x^k r(x)$, kjer je $0 \leq k \leq w-1$ potrebujemo $w$ vektorskih pomikov. Če ima vektor $r(x)$ malo členov, oziroma če je trinom, lahko izvedemo množenje $x^k r(x)$ samo z nekaj besednimi pomiki (pri trinomu zhteva vsak $x^k r(x)$ samo dva besedna pomika).

2. For zanka se izvrši $m-1$-krat.

   2.1 Spet imamo primerjavo $c_i = 1$, ki se nanaša samo na en bit, zato lahko to zanemarimo. V najslabšem primeru je na vsakem koraku $c_i = 1$ in se if zanka vsakokrat izvrši. Za izračun spremenljivk $j$ in $k$ potrebujemo 5 operacij s števili. To zahteva dosti manj časa kot ostale operacije, zato smemo to zanemariti. Za prištetje vektorja $u_k(x)$ k skrajšanemu vektorju $C\{j\}$ potrebujemo eno vektorsko seštevanje. V primeru, ko je redukcijski polinom trinom, imamo zgolj dve besedni seštevanji. V pričakovanem primeru se if zanka izvrši za vsaki drugi $i$.

V spolšnem smo dobili $w$ vektorskih pomikov. Ko je redukcijski polinom trinom, pa imamo samo $2w$ besednih pomikov. Vektorskih seštevanj je v najslabšem primeru $m-1$, v pričakovanem pa $\frac{m-1}{2}$. V primeru trinoma imamo zgolj $2(m-1)$ besednih seštevanj, kar je enako kot dve vektorski seštevanji. Algoritem 5 zahteva $2m$ bitov za vektor $c$, $m$ bitov za vektor $f$, za shranjevanje vseh produktov $u_k(x) = x^k r(x)$ pa potrebujemo približno $w \cdot m$ bitov.

Ta algoritem bi seveda pospešili, če bi izvajali redukcijo nad besedami, ne nad biti. V ALGORITMU 6 smo za redukcijski polinom vzeli točno določen pentonom, operacije pa izvajali po besedah. Pretvorba algoritma na splošen polinom je enostavna. Poglejmo časovno zahtevnost algoritma 6 za fiksen pentonom.

1. Dani polinom moramo zreducirati s stopnje največ 324 na stopnjo manjšo od 163. Redukcijo izvajamo po besedah, torej moramo reducirati pet besed. For zanka se tako izvrši 5-krat.

   1.1 Vrednost $i$-te besede priredimo vektorju $T$. Torej imamo samo eno besedno prirejanje.

   1.2 Vsi nadaljni koraki do 2. koraka zahtevajo besedne pomike in seštevanja (XOR). Število seštevanj in pomikov je odvisno od števila členov v redukcijskem polinomu. Korak 1.2 zahteva en besedni pomik in eno besedno XOR operacijo.

   1.3 Trije besedni pomiki in štiri XOR besedne operacije.

   1.4 Dva besedna pomika in dve XOR besedni operaciji.

2. Besedi $T$ priredimo vrednost besede $C[5]$, ki smo ji z enim besednim seštevanjem počistili začetne tri bite.

3. Urediti moramo še začetni dve besedi. Za prvo besedo, torej za $C[0]$ porabimo tri besedne pomike in štiri besedne XOR operacije.

4. Za urejanje besede $C[1]$ potrebujemo dva besedna pomika in dve besedni XOR operaciji.

5. V zadnji besedi, torej v $C[5]$ moramo počistiti neuporabljene bite, za kar potrebujemo eno besedno seštevanje.

Skupno smo dobili 35 besednih pomikov in 43 besednih XOR operacij. Ker dani vektor sestavlja pet besed, zahteva algoritem 6 približno 7 vektorskih pomikov in približno 9 vektorskih seštevanj. V tem primeru smo imeli fiksen redukcijski polinom in s tem fiksno stopnjo polinoma, ki ga želimo reducirati. Poglejmo, kako bi bilo v primeru poljubnega pentonoma poljubne stopnje. Torej reduciramo polinom stopnje največ $2m - 2$ s pentonomom stopnje največ $m$.

1. For zanka se ponovi tolikokrat, kolikor besed moramo reducirati, torej $t$-krat.

   1.1 do 1.4 Teh korakov znotraj for zanke je pri vseh pentonomih približno enako. Imamo eno besedno prirejanje, 6 besednih pomikov in 7 besednih XOR operacij.

2. V skrajni, $t$-ti besedi moramo počistiti nekaj bitov, kar storimo z enim besednim XOR.

3. do 5. Urediti moramo skrajne besede, za kar potrebujemo približno toliko operacij kot v zgornjem primeru. Torej dodatnih 5 besednih pomikov in 7 besednih XOR.

Skupaj smo spet dobili približno 7 vektorskih pomikov in približno 9 vektorskih seštevanj. Podobno bi lahko izračunali za trinom. Dobili bi približno za polovico manj operacij kot v primeru pentonoma, torej 4 vektorske pomike in 4 vektorska seštevanja. Kaj pa če je redukcijski polinom poljuben nerazcepen polinom stopnje $m$? Poglejmo si spet algoritem po korakih.

1. Reducirati moramo $t$ besed, torej se for zanka ponovi $t$-krat.

   1.1 do 1.4 V najslabšem primeru ima izbrani redukcijski polinom $m$ členov. Zaradi tega se število korakov znatno poveča. Vsak člen polinoma moramo posebej upoštevati. Dobimo približno $m$ besednih pomikov in $m$ besednih XOR (v resnici je to število za neko konstanto večje). V pričakovanem primeru bo imel redukcijski polinom $m/2$ členov. Tako bomo dobili za približno polovico manj pomikov in XOR operacij.

2. Spet moramo urediti še skrajne besede. V $t$-ti besedi počistimo bite z enim besednim XOR.

3. do 5. Še dve besedi moramo urediti. Za to potrebujemo v najslabšem primeru več kot $m$ pomikov in še nekaj več XOR operacij, v pričakovanem pa približno dvakrat manj.

Skupaj smo v najslabšem primeru dobili približno $2m$ vektorskih pomikov in nekaj več kot $2m$ vektorskih seštevanj. V pričakovanem primeru dobimo približno dvakrat manj operacij, to je nekaj več kot $m$ vektorskih pomikov in nekaj več kot $m$ vektorskih seštevanj. Prostorska zahtevnost je približno $2m + m$ bitov ($2m$ za vektor $c$ in $m$ za vektor $f$).

## KVADRIRANJE

Za kvadriranje smo uporabili zelo enostaven postopek, opisan v ALGORITMU 7. Vsak byte moramo najprej pretvoriti v njegovo 2-bytno verzijo. V ta namen že imamo preračunane tabele, o katerih smo govorili na začetku tega poglavja. Tako razširimo dani polinom, ki ga na koncu reduciramo. Analizirajmo algoritem po korakih.

1. Tabele za razširjene verzije bytov imamo vnaprej pripravljene.
2. For zanka se izvede $t$-krat.
   2.1 K vsaki besedi vektorja $A$ poiščemo ustrezni niz. Torej imamo eno vektorsko prirejanje.
   2.2 Vektorju $C$ priredimo pripadajoče dvojne byte. Spet samo prirejanje, vendar za $2m$-bitni vektor, kar stane toliko kot dve prirejanji za $m$-bitni vektor.
3. Dobljeni niz moramo reducirati. Uporabimo verzijo algoritma 6, katerega časovno zahtevnost že poznamo.

Skupno smo dobili 3 vektorska prirejanja in še toliko operacij, kot jih zahteva redukcija. Če zanemarimo prirejanja, lahko zaključimo, da nas stane kvadriranje v polinomskih bazah zgolj toliko, kot nas stane redukcija. Shraniti moramo 3 vektorje dolžine $m$ ($a$, $f$ in rezultat $b$) in en vektor dolžine $2m$ (vektor $c$).

## INVERZ

Inverz elementa, predstavljenega v polinomski bazi, smo izračunali z različico razširjenega Evklidovega algoritma za polinome. Vhodni podatek je vektor $a$ dolžine $m$, izhodni pa njegov inverz, prav tako vektor dolžine $m$. Poglejmo ALGORITEM 8 po korakih.

1. V prvem koraku imamo 4 vektorska prirejanja.
2. Stopnja vektorja $u$ se na vsakem koraku zmanjša vsaj za 1. Zato se bo while zanka v najslabšem primeru izvedla $m$-krat, saj se bo stopnja vektorja $u$ na vsakem koraku zmanjšala samo za 1. V pričakovanem primeru se bo while zanka izvedla $m/2$-krat, ker se bo stopnja vektorja $u$ zmanjševala za 2.
   2.1 Imamo 1 odštevanje števil, kar je poceni v primerjavi z drugimi operacijami in smemo zanemariti.
   2.2 V najslabšem primeru se bo if zanka vsakič izvedla (vsakič bo $j < 0$). V zanki imamo še 3 prirejanja. V pričakovanem primeru bomo izvedli ta tri prirejanja v vsaki drugi zanki.
   2.3 Imamo 2 vektorska pomika in 2 vektorski seštevanji.

Dobili smo, da v najslabšem primeru algoritem 8 zahteva $2m$ vektorskih pomikov in prav toliko vektorskih seštevanj. V pričakovanem primeru dobimo za polovico manj vektorskih pomikov in seštevanj, torej po $m$. Algoritem potrebuje prostora za šest

vektorjev dolžine $m$, torej $6m$ bitov.

Strnimo rezultate analize algoritmov za polinomske baze v naslednji tabeli. Oznaka $t$ pomeni število besed v vektorju, torej $t = \lceil m/w \rceil$. V algoritmu 4 smo za širino oken vzeli $v = 4$. Za redukcijo smo v obeh primerih, algoritma 5 in 6, vzeli primer, ko je redukcijski polinom trinom. Na začetku poglavja smo videli, da je vektorski pomik približno trikrat dražji od vektorskega seštevanja, zato smo v skupnem rezultatu to upoštevali tako, da smo število pomikov pomnožili s tri in jih nato sešteli s številom seštevanj. V primeru smo za dolžino vektorja vzeli $m = 160$, za dolžino besed $w = 32$ in je zato $t = 5$.

| Algoritmi za PB | Pomiki | | Seštevanja | | Skupaj | Primer | Prostorska |
|---|---|---|---|---|---|---|---|
| | Najsl. | Pričak. | Najsl. | Pričak. | pričak. | pričak. | zahtevnost |
| Mnž. in red. (Alg1) | $m-1$ | | $m$ | $m/2$ | $\frac{7m}{2}-3$ | 557 | $3m$ |
| Množenje (Alg2) | $w-1$ | | $m$ | $m/2$ | $\frac{m}{2}+3w-3$ | 173 | $4m$ |
| Množenje (Alg3) | $2(w-1)$ | | $m$ | $m/2$ | $\frac{m}{2}+6w-6$ | 266 | $4m$ |
| Množenje (Alg4) | $\frac{w}{2}+1$ | $\frac{w}{2}+1$ | $\frac{m}{4}+11$ | $\frac{m}{4}+11$ | $\frac{m+6w}{4}+14$ | 108 | $4m+2^v(m+v)$ |
| Redukcija (Alg 5) | $2w/t$ | | 2 | | $\frac{6w}{t}+2$ | 41 | $(w+3)m$ |
| Redukcija (Alg 6) | 4 | | 4 | | 4 | 4 | $3m$ |
| Kvadriranje (Alg 7) | 4 | | 4 | | 4 | 4 | $5m$ |
| Inverz (Alg 8) | $2m$ | $m$ | $2m$ | $m$ | $4m$ | 640 | $6m$ |

Tabela 5. Število operacij pri algoritmih v polinomski bazi.

## 4.3 ALGORITMI ZA NORMALNE BAZE

Normalne baze so se izkazale za izredno učinkovite pri kvadriranju. Kvadrat elementa, predstavljenega v normalni bazi, izračunamo zgolj z enim cikličnim pomikom koordinat (torej 1 vektorski pomik). Žal sta se množenje in inverz v normalnih bazah izkazala kot težja problema. Množenje smo delno rešili z uporabo optimalnih normalnih baz. Analize algoritmov z normalnimi bazami bomo naredili za najslabši primer, za pričakovani primer in za primer optimalnih normalnih baz.

Elementi obsega $\mathbb{F}_{2^m}$, predstavljeni v normalni bazi, so tudi binarna zaporedja dolžine $m$. Kot pri polinomskih bazah, lahko tudi v normalnih bazah seštejemo dva elementa z logičnim-ali (XOR) po besedah, torej potrebujemo zgolj 1 vektorsko seštevanje. Algoritma za množenje in inverz sta zapletenejša. Poglejmo njuno časovno zahtevnost.

## MNOŽENJE

Hitrost algoritma za množenje je precej odvisna od normalne baze, v kateri so elementi predstavljeni. Če vsebuje multiplikacijska tabela normalne baze mnogo neničelnih členov, potrebujemo zelo veliko časa za množenje matrike z vektorji. Z uporabo optimalnih normalnih baz pa se stvari precej pospešijo. Žal tega algoritma ne moremo izvajati z besednimi operacijami. Spustiti se moramo na nivo bitov. S tem postanejo operacije nekoliko dražje. Do posameznega bita v besedi pridemo z maskirno besedo in za to potrebujemo 1 besedni AND. Vhodna podatka sta spet vektorja $a$ in $b$ dolžine $m$, rezultat pa je vektor $c$ dolžine $m$. Analizirajmo ALGORITEM 9.

1. For zanka se izvede $m$-krat.
   Matrika $T$ je shranjena tako, da čimbolj izkoristimo njeno obliko. Tri bite zmnožimo z 2 AND bitnima operacijama. Nato moramo dobljeni bit prišteti k ostalim, oziroma izvesti 1 XOR operacijo. Število teh množenj in seštevanj v posamezni zanki je odvisno od kompleksnosti normalne baze. Seveda je najmanjše, ko je izbrana baza optimalna normalna baza. V najslabšem primeru je kompleksnost $m^2$, v pričakovanem $m^2/2$, za optimalne normalne baze pa $2m - 1$.

Sesštejmo vse operacije. Množenje v normalnih bazah v najslabšem primeru stane $m \cdot 2m^2 = 2m^3$ bitnih AND operacij in $m \cdot m^2 = m^3$ bitnih XOR operacij, kar ustreza $3m^2$ vektorskim seštevanjem (spet smo privzeli, da zahtevata operaciji AND in XOR približno enako časa). V pričakovanem primeru je operacij dvakrat manj, torej $3m^2/2$ vektorskih seštevanj. Pri optimalnih normalnih bazah smo precej na boljšem, saj nas množenje stane $3(2m - 1) = 6m - 3$ vektorskih seštevanj. Vhodna podatka sta niza dolžine $m$, rezultat pa je prav tako niz dolžine $m$. Za shranjevanje multiplikacijske tabele potrebujemo v splošnem primeru dosti prostora (do $m^2$ bitov). Pri optimalnih normalnih bazah pa seveda manj (shranimo samo neničelne elemente).

## INVERZ

Pri algoritmu za inverz v normalnih bazah smo izkoristili lastnost normalne baze, da je kvadriranje izredno poceni. Še vedno pa je računanje inverza precej zamudno. Analizirajmo časovno zahtevnost (ALGORITMA 10) po korakih.

1. Število $m$ se na vsakem koraku prepolovi. Zato se bo while zanka ponovila približno $\lfloor \log_2(m-1) \rfloor$-krat. Nato ločimo dva primera, na lihe $m$ in sode $m$. Če je $m$ lih, poteka algoritem takole:

1.1 Ceno izračuna $\frac{m-1}{2}$ bomo zanemarili.

1.2 Kvadriranje je zgolj ciklični pomik vektorja $a$ za eno mesto. Torej je potenciranje $a^{2^m}$ samo vektorski pomik za $m$ mest. Izračun $a^{2^m}$ nas stane zgolj en vektorski pomik. Za izračun produkta $a^{2^m} \cdot a$ potrebujemo eno množenje v normalnih bazah.

Za sode $m$ pa imamo:

1.3 Spet bomo ceno izračuna $\frac{m-2}{2}$ zanemarili.

1.4 V tem primeru pa imamo dvakrat potenciranje, torej dva vektorska pomika, ter še dve množenji v normalnih bazah.

2. Še eno kvadriranje, torej še en vektorski pomik.

V najslabšem primeru bo $m$ vedno sodo število, to bo takrat, ko bo imelo število $m$ v binarni predstavitvi same enice. Tedaj bo algoritem 10 zahteval $2\lfloor \log_2(m-1) \rfloor$ množenj v normalni bazi. Če uporabimo optimalne normalne baze, nas stane posamezno množenje manj. V najslabšem primeru potrebujemo še $2\lfloor \log_2(m-1) \rfloor + 1$ vektorskih pomikov. V pričakovanem primeru bo v binarni predstavitvi števila $m$ približno $m/2$ enic in nas bo računanje inverza stalo približno $\frac{3}{2}\lfloor \log_2(m-1) \rfloor$ množenj v normalni bazi in prav toliko vektorskih pomikov. Prostorska zahtevnost je skromna, zgolj $m$ bitov.

V spodnji tabeli so rezultati analize algoritmov za normalne baze. Posebej smo navedli rezultate za optimalne normalne baze. V primeru smo vzeli $m = 160$.

| Algoritmi za NB | primer | Množenje (Alg 9) | Kvadriranje | Inverz (Alg 10) |
|---|---|---|---|---|
| Pomiki | najsl. | 0 | 1 | $2\lfloor \log_2(m-1) \rfloor + 1$ |
| | pričak. | 0 | 1 | $\frac{3}{2}\lfloor \log_2(m-1) \rfloor$ |
| | ONB | 0 | 1 | - |
| Seštevanja | najsl. | $3m^2$ | 0 | $6m^2\lfloor \log_2(m-1) \rfloor$ |
| | pričak. | $3m^2/2$ | 0 | $\frac{9m^2}{4}\lfloor \log_2(m-1) \rfloor$ |
| | ONB | $6m-3$ | 0 | $\frac{9}{2}(2m-1)\lfloor \log_2(m-1) \rfloor$ |
| Skupaj | pričak. | $3m^2/2$ | 3 | $\frac{3}{2}(3+\frac{3m^2}{2})\lfloor \log_2(m-1) \rfloor$ |
| Primer | pričak. | 957 | 3 | 10800 |
| Prostorska zahtevnost | | $m^2+3m$ | $m$ | $m$ |

Tabela 6. Število operacij v algoritmih za normalne baze.

## 4.4 ALGORITMI ZA UMBRALNE BAZE

Umbralne baze združujejo lepe lastnosti polinomskih in normalnih baz. Tako so tudi algoritmi za aritmetiko v umbralnih bazah mešanica algoritmov za polinomske

in normalne baze. Pojdimo po vrsti.

## KVADRIRANJE

Idejo za kvadriranje v umbralnih bazah smo povzeli po algoritmu za kvadriranje v polinomskih bazah. Videli bomo, da je v umbralnih bazah kvadriranje še cenejše. Poglejmo ALGORITEM 11, ki izračuna kvadrat elementa, predstavljenega v umbralni bazi. Kot pri polinomskih in normalnih bazah najprej razširimo niz, pri čemer uporabimo vnaprej pripravljene tabele (`kvadr_byte`). Nato z upoštevanjem lastnosti umbralnih baz zreduciramo niz nazaj do dolžine $m$. Poglejmo po korakih.

1. Razširjanje bytov v njihovo dvojno verzijo izvedemo s pomočjo vnaprej pripravljenih tabel `kvadr_byte`, kar nas stane zanemarljivo časa.

2. Drugo polovico niza hkrati razširimo in nato skrčimo. Pri tem si spet pomagamo s pripravljenimi tabelami `kvadr_byte` in `obrni_byte`. V ta namen bi lahko imeli pripravljeno kar eno samo tabelo, ki bi združila zgornji dve, torej bi hkrati razširila in obrnila byte.

Algoritem 11 sestavlja zgolj pretvarjanje bytov v razširjeno verzijo in reduciranje bytov s pomočjo vnaprej pripravljenih tabel. To so v primerjavi z ostalimi zelo poceni operacije. Zato bomo rekli, da nas kvadriranje v umbralnih bazah stane približno toliko, kot kvadriranje v normalnih bazah. Prostorska zahtevnost je $2m$ bitov.

## MNOŽENJE

Kot pri kvadriranju, smo tudi pri množenju (ALGORITEM 12) razdelili postopek na dva dela. Najprej vektor razširimo s pomočjo pravil iz leme 2.4.6. Nato indekse reduciramo.

1. V prvem koraku imamo tri prirejanja.

2. For zanka se izvede $m$-krat.

   2.1 Dva vektorska pomika.

   2.2 If zanka se bo v najslabšem primeru izvedla vsakič, v pričakovanem pa vsakič drugič. Za vsoto treh vektorjev potrebujemo 2 vektorski seštevanji.

Algoritem 12 torej zahteva skupno $2m$ vektorskih pomikov. V najslabšem primeru zahteva $2m$, v pričakovanem pa $m$ vektorskih seštevanj. Za shranjevanje porabimo dvakrat po $m$ bitov (za vektorja $a$ in $b$), dodatnih $m$ bitov pa za shranjevanje produkta $c$.

### INVERZ

Tudi za inverz smo si pomagali s podobnim algoritmom kot pri polinomskih bazah. Uporabili smo različico razširjenega Evklidovega algoritma. Postopek je enak kot pri polinomskih bazah, le v algoritmu upoštevamo pravilo množenja v umbralnih bazah.

1. Polinom $a(\beta)$ reduciramo z enim vektorskim seštevanjem.
2. Za razširjeni Evklidov algoritem pri umbralnih bazah potrebujemo enako časa kot pri polinomskih bazah. V najslabšem primeru imamo tako $2m$ vektorskih pomikov in $2m$ vektorskih seštevanj. V pričakovanem pa za polovico manj, torej $m$ pomikov in $m$ seštevanj.
3. V najslabšem primeru bo konstanten člen vedno enak 1 in bomo morali zamenjati. Ceno prirejanj pa bomo zanemarili.

Za inverz v umbralnih bazah smo torej porabili podobno časa kot za inverz v polinomskih bazah, torej v najslabšem primeru $2m$ vektorskih pomikov in $2m + 1$ vektorskih seštevanj. V pričakovanem primeru imamo $m$ vektorskih pomikov in $m + 1$ vektorskih seštevanj. Tudi prostorska zahtevnost je podobna, torej $6m$ bitov.

Zapišimo rezultate analize algoritmov za umbralne baze v tabeli 4. Spet smo za primer vzeli $m = 160$.

| Algoritmi za UB | Pomiki | | Seštevanja | | Skupaj | Primer | Prostorska |
|---|---|---|---|---|---|---|---|
| | Najsl. | Pričak. | Najsl. | Pričak. | pričak. | pričak. | zahtevnost |
| Kvadriranje (Alg 11) | 1 | 1 | 1 | 1 | 4 | 4 | $2m$ |
| Množenje (Alg 12) | $2m$ | | $2m$ | $m$ | $7m$ | 1120 | $3m$ |
| Inverz (Alg 13) | $2m$ | $m$ | $2m+1$ | $m+1$ | $4m+1$ | 641 | $6m$ |

Tabela 7. Število operacij v algoritmih za umbralne baze.

## 4.5 REZULTATI

Strnimo rezultate v tabeli. Pri polinomskih bazah smo posameznim operacijam kar prišteli ceno redukcije s trinomom. Pri množenju smo navedli najučinkovitejši algoritem, to je ALGORITEM 4. Pri normalnih bazah smo tudi posebej navedli rezultate v pričakovanem primeru in za optimalne normalne baze. Pri umbralnih bazah smo navedli število operacij za pričakovani primer.

| Število operacij | Seštevanje | Množenje | Kvadriranje | Inverz |
|---|---|---|---|---|
| Polinomske baze | 1 | $\frac{m+6w}{4} + 14$ | 4 | $4m$ |
| Normalne baze | 1 | $3m^2/2$ | 1 | $\frac{3}{2}(3 + \frac{3m^2}{2})\lfloor \log_2(m-1) \rfloor$ |
| Optimalne normalne baze | 1 | $6m - 3$ | 1 | $(9m-3)\lfloor \log_2(m-1) \rfloor$ |
| Umbralne baze | 1 | $7m$ | 4 | $4m + 1$ |

Tabela 8. Primerjava med številom vektorskih operacij pri algoritmih za različne baze.

Iz tabele je razvidno, da nobena od obravnavanih baz ni dobra za vse operacije. Izstopa operacija kvadriranja v normalnih bazah, ki je daleč najhitrejša operacija (če ne štejemo seštevanja, ki je enako hitro v vseh naštetih bazah). Vendar se zaplete pri ostalih dveh operacijah. V polinomskih bazah so vse operacije približno enako hitre. Umbralne baze so kar ugodne, saj združujejo dobre lastnosti polinomskih in normalnih baz. Tako je kvadriranje v njih skoraj zastonj, pa tudi množenje in računanje inverza ni preveč zamudno. Žal ne moremo poljubno prehajati med bazami, saj bi to pomenilo mnogo dodatnega časa. Zato se moramo glede na implementacijo aritmetike na eliptičnih krivuljah odločiti za določeno bazo. Primerjajmo še zgornje algoritme glede na prostorsko zahtevnost.

| Prostorska zahtevnost | Seštevanje | Množenje | Kvadriranje | Inverz |
|---|---|---|---|---|
| Polinomske baze | $3m$ | $20m + 64$ | $5m$ | $6m$ |
| Normalne baze | $3m$ | $m^2 + 3m$ | $m$ | $m$ |
| Optimalne normalne baze | $3m$ | $5m - 1$ | $m$ | $m$ |
| Umbralne baze | $3m$ | $3m$ | $2m$ | $6m$ |

Tabela 9. Primerjava med algoritmi glede na prostorsko zahtevnost.

Prostorska zahtevnost je pri vseh algoritmih linearno odvisna od števila $m$, razen pri množenju v normalnih bazah, ko multiplikacijska tabela zahteva $m^2$ bitov. Vendar lahko pri implementaciji to tabelo shranimo drugače (samo v najslabšem primeru je res $m^2$ neničelnih elementov). Pri množenju v polinomskih bazah smo shranjevali $2^v = 16$ produktov, zato je prostorska zahtevnost nekoliko večja.

# Poglavje 5

# ZAKLJUČEK

V tem delu smo predstavili pomembnejše baze binarnih končnih obsegov, ki se uporabljajo v kriptografiji z eliptičnimi krivuljami. Za te baze smo skušali poiskati najhitrejše algoritme za osnovne operacije, to so seštevanje, množenje, kvadriranje in računanje inverza. Naredili smo analizo opisanih algoritmov in jih primerjali med seboj. Težko je povedati, katera baza je v splošnem najboljša. Videli smo, da ima vsaka baza svoje dobre in slabe lastnosti. Ko implementiramo aritmetiko s točkami z eliptične krivulje, imamo na razpolago več možnosti. Točke lahko predstavimo v različnih koordinatah. Če izberemo afine koordinate, potrebujemo za seštevanje in podvajanje točk dve množenji in eno invertiranje elementov iz končnega obsega. Pri projektivnih koordinatah pa imamo več množenj, vendar nobenega sprotnega invertiranja (invertiramo le na koncu, ko moramo pretvoriti točko nazaj v afine koordinate). Ker je invertiranje v normalnih bazah drago, pri implementaciji aritmetike nad eliptičnimi krivuljami, kjer imamo točke predstavljene v afinih koordinatah, raje predstavimo elemente iz končnega obsega v katerih drugih bazah. In ko so točke z eliptične krivulje predstavljene v projektivnih koordinatah, izberemo tisto bazo končnih obsegov, v kateri je množenje najcenejše. Skratka, imamo ogromno možnosti. Izberemo tisto, ki je najboljša v konkretnem primeru.

Vendar je kriptografija nad eliptičnimi krivuljami področje ki se intenzivno razvija in mnogo strokovnjakov išče nove, boljše algoritme. Omenili smo že nekatere matematike, ki se ukvarjajo s kriptografijo na eliptičnih krivuljah. S hitro implementacijo aritmetike s polinomskimi bazami se ukvarjajo Christof Paar (npr. članek 'Some remarks on efficient inversion on finite fields', Crypto 1995), Richard Schroeppel (npr. članek 'Faster elliptic calculations in GF($2^n$)', Crypto 1995), in drugi. Pomemben članek o učinkoviti implementaciji aritmetike z normalnimi bazami je članek 'A fast software implementation for aritmetic operations in GF($2^n$)', avtorjev E.De Win, A. Bosselaers, C. Vanderberghe, P. De Gersem in J. Nadewalle (ASIACRYPT, 1996). Michael Rosing je napisal knjigo 'Implementing elliptic curve cryptography' (Manning Publications Company, 1999). Veliko raziskav poteka o optimalnih normalnih

bazah. Pomembni članki o tem so: Shuhong Gao, Joachim von zur Gathen in Daniel Panario, 'Gauss periods: orders and cryptographical applications' (Mathematics of Computation, 1998), Shuhong Gao in Scott A. Vanstone, 'On orders of optimal basis generators' (Mathematics of Computation, 1995) in Aleksandar Jurišić, 'Umbral optimal normal bases'. Pomembno delo na področju kriptografije z eliptičnimi krivuljami je opravil Alfred Menezes (npr. knjiga 'Handbook of applied cryptography').

# Literatura


[1] I. Vidav, *Algebra*, Mladinska knjiga, Ljubljana, 1989.

[2] Lidl, Niederreiter, *Finite fields*, Cambridge University Press, 1987.

[3] D. Hankerson, J.Lopez Hernandez, A. Menezes, *Software Implementation of Elliptic Curve Cryptography Over Binary Fields*, University of Waterloo, Faculty of Mathematics,CORR 2000-42 (Avgust 2000).

[4] A. Menezes, I. A. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, T. Yaghoobian, *Applications of Finite Fields*, Kluwer Academic Publishers, 1992.

[5] A. Jurišic, *Umbral optimal normal bases*, Politehnika Nova Gorica in IMFM (Avgust 2001).

[6] A. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.