UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO
MATEMATIKA - Uporabna smer

MIHA REŠČIČ

CHEBYSHEVE BAZE IN ODPORNOST NA NAPADE S STRANSKIM KANALOM

DIPLOMSKO DELO

Ljubljana 2006

Povzetek

V tem delu smo vpeljali Chebysheve baze za končne obsege. Le–te omogočajo precejšnje izboljšanje kvadriranja, korenjenja in invertiranja elementov končnih obsegov. Opisali smo konstrukcijo Chebyshevih baz iz optimalnih normalnih baz ter nekatere osnovne lastnosti. Izpeljali smo tudi izboljšane in hitrejše algoritme, zato je najbolj pomemben element dela implementacija celotne aritmetike Chebyshevih baz, kar nam omogoča natančno analizo osnovnih in izboljšanih algoritmov ter primerjavo med njimi. Spoznali smo tudi algebraične osnove končnih obsegov in eliptičnih krivulj. Nenazadnje smo definirali napade s stranskim kanalom na kriptosisteme z eliptičnimi krivuljami ter možne načine zaščite algoritmov.

Ključne besede: eliptične krivulje, končni obsegi, aritmetična teorija števil

Abstract

In this thesis we define Chebyshev basis for finite fields. They allow us better square, square root and inverse calculations in finite fields. We describe the construction of Chebyshev basis from the optimal normal basis and include some basic properties. We also develop faster and optimized algorithms. This makes the implementation of all algorithms in Chebyshev basis the most important part of this thesis. It allows the analysis and comparison of algorithms. We also describe some basics about finite fields and elliptic curves. At the end we define side channel attacks on elliptic curve cryptosystems and possible protection of algorithms against them.

Key words: elliptic curves, finite fields, computational number theory

Math. subj. class (2006):

PROGRAM DIPLOMSKEGA DELA

Delo naj predstavi matematične osnove, potrebne za razumevanje Chebyshevih oziroma umbralnih baz končnih obsegov karakteristike 2, ki jih uporabljamo za implementacije kriptosistemov z javnimi ključi. Poudarek naj bo na kriptosistemih z eliptičnimi krivuljami, torej na računanju večkratnika točke na eliptični krivulji. Eden izmed osrednjih problemov je implementacija učinkovitega algoritma za računanje multiplikativnega inverza. Glavni cilji so:

a) implementacija razširjenega Evklidovega algoritma v Chebyshevi bazi,

b) analiza zahtevnosti algoritmov (štetje inštrukcij) in primerjava z empiričnimi rezultati,

c) zaščita pred napadi s stranskim kanalom (angl. *side chanell attack*) pri digitalnih podpisih in usklajevanju ključa,

d) prehod med Chebyshevo in optimalno normalno bazo tipa II.

Literatura:

E. Brier and M. Joye, *Weierstrass Elliptic Curves and Side Chanell Attacks*, 4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002. Proceedings LNCS **2274**, pp. 335–345.

S. Gao and S. A. Vanstone, On orders of optimal normal bases generators, *Math. Comp.* **64** (1995) 1227–1233.

A. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone and T. Yaghoobian, *Applications of finite fields*, The International Series in Engineering and Computer Science, Springer 1993.

D. Hankerson, A. Menezes, S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer Professional Computing, 2004.

Kazalo

Poglavje 1

UVOD

Kot matematiku mi je skozi celoten študij največji izziv predstavljala prav želja, da lahko pridobljeno znanje uporabim v praktične, življenske namene. Vendar je za prestop v *resničen* predel matematike potrebna izbira ustrezne tematike. In takšna tematika se je ponudila z eliptičnimi krivuljami (in končnimi obsegi).

Varnost komunikacij oziroma varnost podatkov spada v področje *kriptografije*. Le–ta je veda, ki se je začela razvijati s trenutkom, ko je človek želel nekaj skriti pred drugimi. Lepota razvoja kriptografije, še posebej pa po 2. svetovni vojni in s hitrim razvojem interneta, je, da je dandanes dostopna vsakomur. Zadnjih 30 let postaja vse bolj popularna *kriptografija javnih ključev*. Sedaj ima vsak uporabnik svoj *javni* ključ(tega vidijo vsi udeleženci komunikacij) in svoj skriti *privatni* ključ. Na ta način se je število potrebnih ključev bistveno zmanjša.

V zadnjih 30–ih letih je na področju komunikacij prevladovala metoda *RSA*, na področju izmenjave ključev med uporabniki pa ti. *Diffie–Hellmanova* metoda izmenjave ključev. Odkar ima vsak posameznik (vsaj en) elektronski poštni naslov so postale vse bolj popularne funkcije za podpisovanje dokumentov. Zadnjih nekaj let pa so vlogo skoraj vseh omenjenih sistemov prevzele *eliptične krivulje*.

Vsi omenjeni mehanizmi imajo dve skupni lastnosti: vsi so dovzetni in ranljivi za različne tipe napadov in vsi imajo v ozadju skrite algebraične strukture, imenovane *končni obsegi*. In varnost vseh temelji na (v osnovi) eni sami predpostavki:

- *v multiplikativni grupi končnega obsega G izberemo nek element a, ki generira dovolj veliko podgrupo P. Po potenciranju na nek zelo velik eksponent e, je iz končnega rezultata $b = a^e$ izjemno težko izračunati prvotni eksponent e.*

Problem iskanja eksponenta e poznamo pod imenom *problem diskretnega logaritma - DLP*. Problem je z leti, zaradi vse hitrejših računalnikov in učinkovite implementacije *index calculus* metode, postal vse težji. K temu sta pripomogla tudi večanje

velikosti podgrupe, potrebne za zaščito eksponenta in zelo hitra metoda potenciranja *kvadriraj in zmnoži*. Iz tega razloga se je razvila kriptografija na eliptičnih krivuljah. Potegnimo analogijo prejšnjega problema:

- *na eliptični krivulji E izberimo neko točko P ki generira podgrupo F na krivulji. Če točko P pomnožimo z nekim zelo velikim številom e je iz končne točke $R = e \cdot P$ zelo težko izračunati prvoten eksponent P.*

Problem iskanja števila e poznamo pod imenom *problem diskretnega algoritma na eliptični krivulji - ECDLP*[1] . Z vse hitrejšimi računalniki in učinkovito metodo *kvadriraj in zmnoži* postaja tudi zgornji problem vse težje rešljiv in zato vse varnejši. Kot primer naj omenim, da je za primerljivo varnost 1024 bitnemu RSA ključu potrebno uporabiti 160 bitni EC ključ. Zato je hitrost kriptosistemov oziroma hitrost implementacije aritmetike izjemnega pomena.

Elemente končnih obsegov je mogoče predstaviti na mnogo načinov z uporabo mnogih različnih baz končnih obsegov. In prav z različnimi bazami dosegamo izboljšave algoritmov v aritmetiki končnih obsegov. Na začetku diplomskega dela bomo spoznali tri tipe baz: polinomske, normalne in optimalne normalne baze. Vsaka ima svoje prednosti pri določenih algoritmih. Optimalne normalne baze recimo s svojimi lastnostmi omogočajo hitro množenje. Vendar so osrednja tema diplomskega dela **Chebysheve** baze, ki so pravzaprav permutirane optimalne normalne baze, in združujejo lastnosti tako polinomskih kot optimalnih normalnih baz. Zaradi lastnosti optimalnih normalnih baz in nekaterih novih lastnosti, pridobljenih s permutacijo elementov optimalne normalne baze, se Chebysheve baze izkažejo kot baze, v katerih je mogoče hitro kvadriranje in korenjenje elementov končnih obsegov. Prav tako nam Chebysheve baze omogočijo izpeljavo hitrih algoritmov za množenje elementov končnih obsegov in za iskanje inverznih elementov v končnih obsegih.

Ker je poglaviten del diplomskega dela prav primerjava med naivnimi osnovnimi algoritmi in hitrimi izboljšanimi algoritmi, je glavni del pričujočega diplomskega dela prav implementacija končnih obsegov, predstavljenih s Chebyshevimi bazami. Vse naštete algoritme, ki jih diplomsko delo obravnava v okviru Chebyshevih baz, sem dejansko sprogramiral(implementiral) in še dodatno(računalniško) izboljšal.

Poleg prej omenjenih predpostavk o varnosti kriptosistemov pa nanjo vplivajo tudi marsikateri drugi zunanji dejavniki. Poznamo več različnih napadov na kriptosisteme: z grobo silo, algebraične,... Kriptosisteme pa se da razbiti tudi na bolj prefinjen, a zato tudi veliko cenejši način, brez neposrednega napada na sistem oziroma algoritem. Takšni načini vsebujejo prisluškovanje napravam, merjenje elektromagnetnega valovanja, merjenje toplotnih sprememb in ostale bolj fizikalne pristope.

[1] Omeniti je potrebno, da v zgornjem opisu problema uporabimo aditivno notacijo operacije grupe na eliptični krivulji, medtem ko v končnih obsegih uporabljamo multiplikativno notacijo.

Takšne napade imenujemo *napadi s stranskim kanalom*. V teh napadih opazujemo vse ločine, ki so stranski produkt kriptosistema. Zato sem zadnji del diplomskega dela posvetil zaščiti nekaterih algoritmov na eliptičnih krivuljah pred napadi s stranskim kanalom.

Poglavja v tem delu so sestavljena takole. Za razumevanje končnih obsegov ter njihovih baz sta namenjeni uvodni dve poglavji. Sledi eno izmed najpomembnejših poglavji tega diplomskega dela, v katerem definiramo Chebysheve baze. V istem poglavju je podan tudi način pretvarjanja med Chebyshevimi bazami in optimalnimi normalnimi bazami. Poglavje zaključimo z opisom nekaterih najosnovnejših algoritmov za osnovne računske operacije znotraj končnih obsegov. Sledi poglavje o izboljšanih algoritmih. V njem izpeljemo in opišemo možne izboljšave in pohitritve vseh opisanih algoritmov. Vsak algoritem posebej zaključimo tudi s primerom iz implementacije. V poglavju o analizi algoritmov bomo analizirali vse opisane algoritme. Poglavje je razdeljeno na dva dela, na osnovne algoritme in izboljšane algoritme. Vsak del vsebuje štetje potrebnih operacij znotraj končnega obsega za vsak algoritem posebej in štetje besednih oziroma vektorskih operacij. Vse preštete vrednosti in izmerjene časovne vrednosti so za primerjavo podane tudi v obsežnih tabelah. Diplomsko delo zaključim s poglavjem o napadih s stranskimi kanali, v katerem vzamemo pod drobnogled algoritem seštevanja točk na eliptični krivulji in in Montgomeryjev algoritem seštevanja in podvajanja točk na eliptični krivulji. Oba algoritma zaščitimo pred napadi s stranskimi kanali in na koncu poglavja strnemo ugotovitve o varnosti spremenjenih algoritmov.

Poglavje 2

Končni obsegi in eliptične krivulje

Učinkovita implementacija končnih obsegov je osnovni pogoj za postavitev kriptosistemov z eliptičnimi krivuljami. Zato je potrebno algebraične strukture prevesti v računalniški jezik. To poglavje predstavlja preprost uvod v končna obsega dveh tipov:

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

nato pa v obdelamo še osnovne definicije in lastnosti običajnih in Weierstrassovih eliptičnih krivulj.

2.1 Končni obsegi

Spoznajmo nekatere najosnovnejše lastnosti končnih obsegov ter nekatere najbolj reprezentativne primere le–teh. Več o končnih obsegih ter bazah končnih obsegov si lahko bralec prebere v [13, Vidav].

Obsegi so posplošitve znanih številskih sistemov (racionalna števila $\mathbb{Q}$, realna števila $\mathbb{R}$, kompleksna števila $\mathbb{Z}$) in njihovih pripadajočih operacij. Definirajmo najprej kolobar. Naj bo $\mathbb{F}$ takšna množica elementov, na kateri sta definirani binarni operaciji $+$ in $\cdot$, za kateri velja:

(i) $(\mathbb{F}, +)$ komutativna grupa z enoto, ki jo označimo z 0,

(ii) $(\mathbb{F} \setminus \{0\}, \cdot)$ je polgrupa. Velja zakon asociativnosti in komutativnost množenja. Pri poljubnih elementih $a,\ b,\ c \in \mathbb{F}$ je

$$(a \cdot b) \cdot c = a \cdot (b \cdot c).$$

(iii) Velja zakon distributivnosti. Za $a,\ b,\ c \in \mathbb{F}$ je

$$(a + b) \cdot c = a \cdot c + b \cdot c.$$

Sedaj definirajmo še obseg. Naj bo $\mathbb{F}$ takšna množica elementov, na kateri sta definirani binarni operaciji $+$ in $\cdot$, za kateri velja:

(i) $(\mathbb{F}, +)$ abelova grupa z enoto, ki jo označimo z 0,

(ii) $(\mathbb{F}\backslash\{0\}, \cdot)$ abelova grupa z enoto, ki jo označimo z 1. Velja zakon asociativnosti.

(iii) Velja zakon distributivnosti.

Osredotočimo se na kočne obsege, saj so pomembni v kriptografiji in v implementacijah. Imenujemo jih **Galoisovi** obsegi.

V tem diplomskem delu se uporablja predpostavka, da je z oznako *končen obseg* predstavljen *Galoisov obseg*.

Operacije v obsegu

Z operacijama seštevanja in množenja lahko sedaj definiramo še dve dodatni operaciji: odštevanje in deljenje. *Odštevanje* definiramo s pomočjo seštevanja na naslednji način : $a - b = a + (-b)$, kjer je $-b$ inverzen element za operacijo seštevanja in zanj velja $b + (-b) = 0$. *Deljenje* definiramo s pomočjo množenja na naslednji način: $a/b = a \cdot b^{-1}$, kjer je b^{-1} inverzni element za operacijo množenja, za katerega velja $b \cdot b^{-1} = 1$. Takšnemu elementu pravimo tudi *multiplikativni inverz* elementa b. V nadaljevanju bomo videli, da je iskanje multiplikativnega inverza izredno pomembno opravilo.

Obstoj in enoličnost

Število elementov obsega imenujemo **red** obsega. Obseg $\mathbb{F}$ reda q obstaja, če in samo če je q potenca nekega praštevila p, torej ko je $q = p^m$ za neko naravno število m. Število p imenujemo tudi **karakteristika** obsega $\mathbb{F}$. V primeru, da je $m = 1$ pravimo, da je $\mathbb{F}$ **praštevilski** obseg. Za vsako število q ki deli število p^m obstaja natanko en obseg reda q, določen do izomorfizma natančno. Za natančnejše definicije glej [13, Vidav].

Praštevilski obsegi

Naj bo p praštevilo. Ostanki pri deljenju s p, tj. števila $\{0,\ 1,\ 2, \ldots, p-1\}$, skupaj z operacijama seštevanje in množenje po modulu p predstavljajo praštevilski obseg reda p. Označimo ga s $\mathbb{F}_p$. Vsakemu številu a lahko priredimo vrednost $a \bmod p$, ki enolično predstavlja ostanek r pri deljenju števila a s p. Operaciji pravimo **redukcija po modulu** p.

Za lažje razumevanje praštevilskih obsegov (ki so osnova za nadaljne razširitve) podamo primer.

Primer 2.1. *Praštevilski obseg* $\mathbb{F}_{17}$
Elementi obsega so števila $\{0,\ 1, \ldots, 16\}$. *Poglejmo, kako izgledajo nekatere operacije:*

(i) Seštevanje: $12 + 15 = 10$, *ker je* $27 \bmod 17 = 10$.

(ii) Odštevanje: $5 - 13 = 9$, *ker je* $-8 \bmod 17 = 9$.

(iii) Množenje: $11 \cdot 3 = 16$, *ker je* $33 \bmod 17 = 16$.

(iv) Invertiranje: $5^{-1} = 7$, *ker je* $35 \bmod 17 = 1$.

Binarni (dvojiški) obsegi

Binarni oziroma dvojiški obsegi so obsegi reda 2^m. Pravimo jim tudi obsegi s karakteristiko 2. Eden izmed načinov za predstavitev elementov obsega $\mathbb{F}_{2^m}$ je v **polinomski bazi**. Polinomske baze so natančneje opredeljene v poglavju 3.

Predpostavimo, da že poznamo predstavitev elementov obsega s polinomom. Element binarnega obsega predstavimo kot polinom stopnje največ $m-1$ s koeficienti iz obsega $\mathbb{F}_2$:

$$\mathbb{F}_{2^m} = \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0 \mid a_i \in \{0,1\}\}.$$

Za vsak m izberemo nerazcepen polinom $f(x)$ stopnje m. Takšen polinom obstaja in poznamo načine za iskanje teh polinomov. Nerazcepnost $f(x)$ je nekakšna analogija nerazcepnosti praštevila p.

Polinom $f(x)$ je bistvenega pomena, saj se z izjemo operacij seštevanja in odštevanja, ki sta osnovni operaciji seštevanja in odštevanja polinomov po koeficientih, operaciji množenja in invertiranja izvajata po modulu $f(x)$ (za lažjo predstavo lahko tudi tukaj potegnemo vzporednice s praštevilskimi obsegi in osnovnimi operacijami po modulu p). Polinom $f(x)$ ni enolično določen, saj lahko nad danim obsegom najdemo več različnih nerazcepnih polinomov. Obsega istega reda z različnima nerazcepnim polinomom sta *izomorfna*. Za natančne izreke glej [13, X]

Primer 2.2. ***Seznam elementov obsega*** $\mathbb{F}_{2^4}$
Naj bodo $a_0,\ a_1,\ a_2,\ a_3 \in \mathbb{F}_2$. *Elementi obsega* $\mathbb{F}_{2^4}$ *so vsi možni polinomi oblike* $a_3x^3 + a_2x^2 + a_1x + a_0$. *Oglejmo si elemente obsega* $\mathbb{F}_{2^4}$.

0	x^2	x^3	$x^3 + x^2$
1	$x^2 + 1$	$x^3 + 1$	$x^3 + x^2 + 1$
x	$x^2 + x$	$x^3 + x$	$x^3 + x^2 + x$
$x + 1$	$x^2 + x + 1$	$x^3 + x + 1$	$x^3 + x^2 + x + 1$

Primer 2.3. ***Aritmetika obsega*** $\mathbb{F}_{2^4}$
Izberimo nerazcepen polinom $f(x) = x^4 + x + 1$ *in na nekaj primerih poglejmo, kako potekajo osnovne aritmetične operacije.*

(i) *Seštevanje:* $(x^3 + x^2 + 1) + (x^2 + x + 1) = x^3 + x$.

(ii) *Odštevanje:* $(x^3 + x^2 + 1) - (x^2 + x + 1) = x^3 + x$. *Rezultat je enak kot pri seštevanju; sledi iz dejstva, da je v binarnih obsegih* $-1 = 1$ *oziroma* $-a = a$ *za vsak* $a \in \mathbb{F}_{2^m}$.

(iii) *Množenje:* $(x^3 + x^2 + 1) \cdot (x^2 + x + 1) = x^2 + 1$, *ker je*
$(x^3 + x^2 + 1) \cdot (x^2 + x + 1) = x^5 + x + 1$ *in*
$x^5 + x + 1 \bmod (x^4 + x + 1) = z^2 + 1$.

(iv) *Invertiranje:* $(x^3 + x^2 + 1)^{-1} = x^2$,
ker je $(x^3 + x^2 + 1) \cdot x^2 \bmod (x^4 + x + 1) = 1$.

Primer 2.4. ***Primeri obsegov nad različnimi nerazcepnimi polinomi***
Vzemimo obseg $\mathbb{F}_{2^4}$. *Obstajajo trije različni nerazcepni polinomi stopnje* 4: $f_1(x) = x^4 + x + 1$, $f_2(x) = x^4 + x^3 + 1$ *in* $f_3(x) = x^4 + x^3 + x^2 + x + 1$. *Poimenujmo dane obsege s* K_1, K_2 *in* K_3. *Elementi teh obsegov so še vedno enaki kot v prejšnjem primeru, le redukcija po nerazcepnem polinomu nam vrne različne rezultate:*

(i) K_1: $x^3 \cdot x = x + 1$,

(ii) K_2: $x^3 \cdot x = x^3 + 1$,

(iii) K_3: $x^3 \cdot x = x^3 + x^2 + x + 1$.

Ker pa so zgornji trije obsegi obsegi med seboj izomorfni, lahko najdemo izomorfizem na naslednji način: poiščemo $c \in K_2$ tako, da bo $f_1(c) = 0 \bmod (f_2(x))$. Nato razširimo preslikavo $z \mapsto c$ do izomorfizma $\varphi : K_1 \to K_2$. V našem primeru bi bile možne izbire za c naslednje: $x^2 + x$, $x^2 + x + 1$, $x^3 + x^2$ in $x^3 + x^2 + 1$.
V teh primerih sem poudaril nekatere lastnosti polinomskih baz končnih obsegov, saj so ključne pri aritmetiki v teh bazah.

2.2 Eliptične krivulje

Poleg izboljšane aritmetike v določenem tipu baz končnih obsegov je vzporedna tema diplomskega dela tudi varnost kriptosistemov z eliptičnimi krivuljami pred napadom s stranskim kanalom (ang. *Side channel attack*). Zato v tem razdelku najprej definiram objekt eliptične krivulje ter osnovno aritmetiko in terminologijo, ki se uporablja na tem področju, saj nam bo na takšen način olajšano delo v nadalnjih poglavjih.

Najprej v grobem predstavim elementarne definicije in trditve o eliptičnih krivuljah, nato pa bom opisal osnovno aritmetiko, saj je poznavanje aritmetike osnova za vpogled v napade s stranskim kanalom. Preskočil bom tudi natančne definicije aritmetike na supersingularnih in supernesingularnih krivuljah. Več o tem si lahko bralec ogleda v [14] ali v [4].

2.2.1 Definicija krivulje

Definicija 2.5. *Naj bodo $a_1, a_2, a_3, a_4, a_6 \in K$ in $\triangle \neq 0$.* ***Eliptična krivulja*** *E nad obsegom K je množica točk iz $K \times K$ (ali $\overline{K} \times \overline{K}$) skupaj s točko ∞, ki zadoščajo enačbi*

$$E \ : \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{2.1}$$

$\triangle$ imenujemo diskriminanta E in je definirana kot

$$\begin{aligned} \triangle &= -d_2^3d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6, \\ d_2 &= a_1^2 + 4a^2, \\ d_4 &= 2a_4 + a_1a_3, \\ d_6 &= a_3^2 + 4a_6, \\ d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2. \end{aligned}$$

Enačbo (2.1) imenujemo **Weierstrassova enačba**. Obstajajo transformacije spremenljivk, s katerimi enačbo poenostavimo.

Definicija 2.6. *Za eliptični krivulji E_1 in E_2, definirani nad K in zapisani kot*

$$\begin{aligned} E_1 \ : \ y^2 + a_1xy + a_3y &= x^3 - a_2x^2 + a_4x + a_6 \text{ in} \\ E_1 \ : \ y^2 + \bar{a_1}xy + \bar{a_3}y &= x^3 - \bar{a_2}x^2 + \bar{a_4}x + \bar{a_6} \end{aligned}$$

pravimo, da sta ***izomorfni nad*** *K, če obstajajo $u, r, s, t \in K, u \neq 0$ tako da transformacija spremenljivk*

$$(x, y) \rightarrow (u^2x + r, u^3y + u^2sx + t) \tag{2.2}$$

transformira enačbo E_1 v enačbno E_2.

Transformacija nam ohrani krivuljo, le zapis in oblika enačbe se poenostavi. Tako lahko Weierstrassovo enačbo

$$E \ : \ y^2 + a_1xy + a_3y = x^3 - a_2x^2 + a_4x + a_6,$$

precej poenostavimo. Poenostavljena oblika enačbe je tista, ki jo bom uporabljal skozi to poglavje in tudi kasneje pri poglavju 7 o napadu s stranskim kanalom.

Ločimo tri primere:

1. K ima karakteristiko različno od 2 ali 3:
naj bosta $a, b \in K$. V tem primeru uporabimo transformacijo

$$(x, y) \to \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1 x}{216} - \frac{a_1^3 + 4a_1 a_2 - 12a_3}{24} \right),$$

ki spremeni krivuljo E v

$$y = x^3 + ax + b. \tag{2.3}$$

Diskriminanta krivulje je $\triangle = -16(4a^3 + 27b^2)$

2. K ima karakteristiko 2:
naj bosta $a, b \in K$. Tukaj je potrebno upoštevati dve možni transformaciji. V primeru, da je $a_1 \neq 0$ uporabimo transformacijo

$$(x, y) \to \left(a_1^2 x + \frac{a_3}{a_1}, a_1^3 y + \frac{a_1^2 a_4 + a_3^2}{a_1^3} \right)$$

ki spremeni krivuljo E v

$$y^2 + xy = x^3 + ax^2 + b. \tag{2.4}$$

Takšno krivuljo imenujemo **nesupersingularno** in ima diskriminanto $\triangle = b$. V drugem primeru, ko je $a_1 = 1$, pa nam transformacija

$$(x, y) \to (x + a_2, y)$$

spremeni krivuljo E v

$$y^2 + cy = x^3 + ax + b, \tag{2.5}$$

Takšno krivuljo imenujemo **supersingularna** krivulja in ima diskriminanto $\triangle = c^4$

3. K ima karakteristiko 3:
tudi v tem primeru ločimo dva primera. V prvem primeru, ko je $a_1^2 \neq -a_2$ in so $d_2 = a_1^2 + a_2$, $d_4 = a_4 - a_1 a_3$ in $a, b \in K$, nam transformacija

$$(x, y) \to \left(x + \frac{d_4}{d_2}, y + a_1 x + a_1 \frac{d_4}{d_2} + a_3 \right)$$

spremeni krivuljo E v

$$y^2 = x^3 + ax^2 + b, \tag{2.6}$$

Za takšno krivuljo pravimo, da je **nesupersingularna** in ima diskriminanto $\triangle = -a^3 b$. Če pa je $a_1^2 = -a_2$ nam transformacija

$$(x, y) \to (x, y + a_1 x + a_3)$$

transformira krivuljo E v

$$y^2 = x^3 + ax + b, \tag{2.7}$$

Takšni krivulji pravimo, da je **supersingularna** in ima diskriminanto $\triangle = -a^3$.

2.2.2 Grupne lastnosti

Naj bo E eliptična krivulja nad K. Vpeljemo **sekantno metodo** za seštevanje dveh točk na krivulji $E(K)$. Da bo eliptična krivulja $E(K)$ ustrezala grupni lastnosti skupaj z operacijo seštevanja, potrebujemo še dodatno točko, v primeru, da sekanta ne obstaja. Poimenujemo jo **točka v neskončnosti** in jo označimo z ∞. Tako tvori $E(K)$ Abelovo grupo, kjer predstavlja ∞ enoto. S to grupo nato konstruiramo kriptosisteme z eliptičnimi krivuljami.

Pravilo je najlažje opisati geometrično. Predstavljajmo si, da smo v prostoru $\mathbb{R}^2$. Naj bosta $P = (x_1, y_1)$ in $Q = (x_2, y_2)$ dve različni točki na krivulji E. *Vsota* $R = P + Q$ je definirana na naslednji način: najprej narišemo premico skozi P in Q. Premica seče krivuljo v tretji točki. R je ta točka, prezrcaljena preko x–osi. *Podvojitev* R točke $P, R = 2P$ pa se izračuna na naslednji način: najprej narišemo tangento na krivujo v točki P. Tangenta seče krivuljo v neki drugi točki. Rešitev R je nato ta druga točka, prezrcaljena preko x–osi.

Iz geometričnih pravil lahko izpeljemo algebraične formule. Podal bom formulo za poenostavljeno Weierstrassovo enačbo (2.3).

Grupa $\mathbf{E(K)}, \boldsymbol{y^2 = x^3 + ax + b}$, $\mathbf{char(K) \neq 2, 3}$

1. Enota: $P + \infty = \infty + P = P$ za vsak $P \in E(K)$
2. Nasprotni element: če označimo $P = (x, y) \in E(K)$, potem je $(x, y) + (x, -y) = \infty$. Točko $(x, -y)$ označimo z $-P$ in jo imenujemo *nasprotni element* P. Poleg tega velja $-\infty = \infty$
3. Seštevanje točk: Naj bosta $P = (x_1, y_1), Q = (x_2, y_2) \in E(K), P \neq Q$. Potem je $R = (x_3, y_3) = P + Q$

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \quad \text{in} \quad y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1$$

4. Podvajanje točk: bodi $P = (x_1, y_1) \in E(K)$ in $P \neq -P$. Potem je $2P = (x_3, y_3)$

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \quad \text{in} \quad y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1$$

Oglejmo si primer.

Primer 2.7. *Vzemimo eliptične krivulje nad praštevilskim obsegom* $\mathbb{F}_{29}$*. Naj bodo* $p = 29, a = 4$ *in* $b = 20$ *in oglejmo si krivujo*

$$E \ : \ y^2 = x^3 + 4x + 20$$

Preverimo, da je $\triangle = -16(4a^3 + 27b^2) = -176896 \neq 0 \bmod 29$*, torej da je* E *res eliptična krivulja. Točke na* $E(\mathbb{F}_{29})$ *so naslednje:*

∞	$(2,6)$	$(4,19)$	$(8,10)$	$(13,23)$	$(16,2)$	$(19,16)$	$(27,2)$
$(0,7)$	$(2,23)$	$(5,7)$	$(8,19)$	$(14,6)$	$(16,27)$	$(20,3)$	$(27,27)$
$(0,22)$	$(3,1)$	$(5,22)$	$(10,4)$	$(14,23)$	$(17,10)$	$(20,26)$	
$(1,5)$	$(3,28)$	$(6,12)$	$(10,25)$	$(15,2)$	$(17,19)$	$(24,7)$	
$(1,24)$	$(4,10)$	$(6,17)$	$(13,6)$	$(15,27)$	$(19,3)$	$(24,22)$	

Primera operacij:

$$\begin{aligned} (5,22) + (16,27) &= (13,6) \\ 2(5,22) &= (14,6) \end{aligned}$$

2.2.3 Red grupe

Število točk na eliptični krivulji nad obsegom $K = \mathbb{F}_q$ imenujemo tudi **red** grupe $E(\mathbb{F}_q)$ in ga označimo z $\#E(\mathbb{F}_q)$. Ker ima Weierstrassova enačba (2.1) za vsak $x \in \mathbb{F}_q$ največ dve rešitvi, vemo da je $\#E(\mathbb{F}_q) \in [1, 2q + 1]$. Hassejev izrek nam ponudi natančnejše meje za red grupe.

Izrek 2.8. (Hasse) *Naj bo* E *eliptična krivulja nad* $\mathbb{F}_q$*. Potem je*

$$q + 1 - 2\sqrt{q} \leq \#E(\mathbb{F}_q) \leq q + 1 + 2\sqrt{q}.$$

Interval $[q + 1 - 2\sqrt{q}, q + 1 + 2\sqrt{q}]$ imenujemo **Hassejev** interval.

Naslednji izrek nam določi možne vrednosti $\#E(\mathbb{F}_q)$.

Izrek 2.9. *Naj bo* $q = p^m$*, kjer je* p *karakteristika* $\mathbb{F}_q$*. Eliptična krivulja* E *nad* $\mathbb{F}_q$ *z redom* $\#E(\mathbb{F}_q) = q + 1 - t$ *obstaja če in samo če držijo naslednje trditve:*

(i) $t \neq 0 \bmod p$ *in* $t^2 \leq 4q$

(ii) m *je sod in*

$$(t = 0 \text{ ali } t^2 = 2q) \text{ in } (p = 2 \text{ ali } t^2 = 3q) \text{ in } p = 3.$$

(iii) m *je lih in*

$$(t^2 = 4q \text{ ali } t^2 = q) \text{ in } (p \neq \mod 3 \text{ ali } t = 0) \text{ in } p \neq 1 \bmod 4.$$

Posledica izreka 2.9. je, da za vsako praštevilo p in naravno število t, za katerega velja $|t| \leq 2\sqrt{p}$ obstaja eliptična krivulja E nad $\mathbb{F}_q$ z $\#E(\mathbb{F}_q) = p + 1 - t$.

Primer 2.10. *Postavimo $p = 37$. V tem primeru bomo našteli vse eliptične krivulje oblike $y^2 = x^3 + ax + b$ za vsako naravno število n iz Hassejevega intervala $[37 + 1 - 2\sqrt{37}, 37 + 1 + 2\sqrt{37}]$. Naštejemo koeficiente (a, b) eliptične krivulje $E : y^2 = x^3 + ax + b$, definirane nad $\mathbb{F}_{37}$ z $\#E(\mathbb{F}_{37}) = n$.*

n	(a,b)	n	(a,b)	n	(a,b)	n	(a,b)	n	(a,b)
26	$(5,0)$	31	$(2,8)$	36	$(1,0)$	41	$(1,16)$	46	$(1,11)$
27	$(0,9)$	32	$(3,6)$	37	$(0,5)$	42	$(1,9)$	47	$(3,15)$
28	$(0,6)$	33	$(1,13)$	38	$(1,5)$	43	$(2,9)$	48	$(0,1)$
29	$(1,12)$	34	$(1,18)$	39	$(0,3)$	44	$(1,7)$	49	$(0,2)$
30	$(2,2)$	35	$(1,8)$	40	$(1,2)$	45	$(2,14)$	50	$(2,0)$

2.2.4 Struktura grupe

Izrek 2.11. *Naj bo E eliptična krivulja definirana nad $\mathbb{F}_q$. Potem je $E(\mathbb{F}_q)$ izomorfna grupi $\mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}$, kjer sta n_1 in n_2 enolično določeni naravni števili in taki, da n_2 deli n_1 in $q - 1$.*

Opazimo, da je $\#E(\mathbb{F}_q) = n_1 n_2$. Če je $n_2 = 1$, potem je $E(\mathbb{F}_q)$ ciklična grupa. Če pa je $n_1 > 1$ pravimo, da ima $E(\mathbb{F}_q)$ **rang** 2. Če je n_2 majhno število (npr. $2, 3$ ali 4) pravimo, da je $E(\mathbb{F}_q)$ **skoraj ciklična**. Ker n_2 deli tako n_1 kot tudi $q - 1$, pričakujemo, da je $E(\mathbb{F}_q)$ ciklična ali skoraj ciklična za skoraj vse eliptične krivulje E nad $\mathbb{F}_q$.

Primer 2.12. *Oglejmo si eliptično krivuljo*

$$E \ : \ y^2 = x^3 + 4x + 20,$$

definirano nad obsegom $\mathbb{F}_{29}$ (glej primer 2.7.). Krivulja ima red $\#E(\mathbb{F}_{29}) = 37$. Iz dejstva, da je 37 praštevilo sledi, da je $E(\mathbb{F}_{29})$ ciklična grupa in da je vsaka točka iz grupe z izjemo ∞ generator grupe. Oglejmo si, kako točka $P = (1, 5)$ generira grupo $E(\mathbb{F}_{29})$.

$0P = \infty$	$8P = (8,10)$	$16P = (0,22)$	$24P = (16,2)$	$32P = (6,17)$
$1P = (1,5)$	$9P = (14,23)$	$17P = (27,2)$	$25P = (19,16)$	$33P = (15,2)$
$2P = (4,19)$	$10P = (13,23)$	$18P = (2,23)$	$26P = (10,4)$	$34P = (20,26)$
$3P = (20,3)$	$11P = (10,25)$	$19P = (2,6)$	$27P = (13,6)$	$35P = (4,10)$
$4P = (15,27)$	$12P = (19,13)$	$20P = (27,27)$	$28P = (14,6)$	$36P = (1,24)$
$5P = (6,12)$	$13P = (16,27)$	$21P = (0,7)$	$29P = (8,19)$	
$6P = (17,19)$	$14P = (5,22)$	$22P = (3,28)$	$30P = (24,7)$	
$7P = (24,22)$	$15P = (3,1)$	$23P = (5,7)$	$31P = (17,10)$	

Poglavje 3

Baze končnih obsegov

Sedaj, ko smo v grobem predstavili končne obsege, se lahko podrobneje posvetimo bazam končnih obsegov oziroma možnim predstavitvam elementov obsegov.

V tem poglavju bomo obdelali tri najbolj razširjene tipe baz končnih obsegov: polinomske, normalne in optimalne normalne. Poznavanje teh predstavlja osnovo za razumevanje Chebyshevih baz. Poleg tega bomo pri vsakem tipu baz povzel osnovno aritmetiko in nekatere algoritme za lažjo kasnejšo primerjavo.

Končni obseg $\mathbb{F}_{q^m}$, kjer je q praštevilo, razumemo kot vektorski prostor nad podobsegom $\mathbb{F}_q$. Vektorji v tem pogledu predstavljajo elemente obsega $\mathbb{F}_{q^m}$, skalarji pa elemente iz $\mathbb{F}_q$. Vektorski prostor ima dimenzijo m in precej baz.

V primeru, da je $B = \{b_m, b_{m-1}, \ldots, b_1\}$ baza, potem lahko poljuben element $a \in \mathbb{F}_{q^m}$ enolično predstavimo z m–terico $(a_m, a_{m-1}, \ldots, a_1)\}$ elementov iz $\mathbb{F}_q$. Uporabljamo sledeč zapis: $a = a_m b_m + a_{m-1} b_{m-1} + \cdots + a_1 b_1$. V prejšnjih primerih 2.3. in 2.4. so je bil element polinomske baze m–terica(četverica) $(a_m, a_{m-1}, \ldots, a_1), a_i \in \{0, 1\}$, polinomsko bazo obsega $\mathbb{F}_{2^m}$ pa so sestavljali elementi $\{x^{m-1}, x^{m-2}, \ldots, x, 1\}$.

3.1 Polinomske baze

Polinomska predstavitev obsega $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$ je oblike $\mathbb{F}_{q^m} = \mathbb{F}_q[x]/(f)$, kjer je $f \in \mathbb{F}_q[x]$ nerazcepen polinom stopnje m in vsak element iz $\mathbb{F}_{q^m}$ je predstavljen kot polinom stopnje manjše od m. Pripadajočo bazo $\{1, x, x^2, \ldots, x^{m-1}\}$ imenujemo **polinomska baza**.

Obstoj in definicija

Najprej natančneje definirajmo obliko in izbiro polinomske baze.

Trditev 3.1. *Naj bo α neka ničla nekega nerazcepnega polinoma f stopnje m iz*

$\mathbb{F}_q[x]$. *Množica* $B = \{\alpha^{m-1}, \alpha^{m-2}, \ldots, \alpha, 1\}$ *je baza vektorskega prostora* $\mathbb{F}_{q^m}$ *nad* $\mathbb{F}_q$.

Dokaz. Preveriti moramo, da so elementi množice B linearno neodvisni. Minimalni polinom m elementa α mora deliti nerazcepni polinom f. Iz tega in iz nerazcepnosti polinoma f sledi, da je f enak minimalnemu polinomu za α oziroma $c \cdot m(x)$ za $c \in \mathbb{F}_q$. Torej α ni ničla nobenega drugega netrivialnega polinoma iz $\mathbb{F}_q[x]$stopnje manjše od m. Iz tega sledi linearna neodvisnost elementov iz množice B. Dejstvo, da je B baza sledi iz enakosti $|B| = m$. ■

Sedaj lahko definiramo polinomske baze.

Definicija 3.2. *Podmnožica vektorskega prostora* $\mathbb{F}_{q^m}$ *nad* $\mathbb{F}_q$ *je* ***polinomska baza****, če je oblike* $\{\alpha^{m-1}, \alpha^{m-2}, \ldots, \alpha, 1\}$*, kjer je* α *neka ničla nekega nerazcepnega polinoma* p *stopnje* $m \in \mathbb{F}_q[x]$.

Kot smo videli v primerih v prejšnjem poglavju je takšnih nerazcepnih polinomov lahko več. Naj podam primer, za predstavitev zgornje trditve in definicije:

Primer 3.3. ***Primer polinomske baze za*** $\mathbb{F}_{2^4}$
Za nerazcepen polinom izberimo polinom $f(x) = x^4 + x + 1$*. Sedaj poiščimo bazo tako, da najdemo element, čigar red je enak* 15*. Tak element je generator obsega. V našem primeru je to element* $x = \alpha$*:*

α	$\alpha^9 = \alpha^3 + \alpha$
α^2	$\alpha^{10} = \alpha^2 + \alpha + 1$
α^3	$\alpha^{11} = \alpha^3 + \alpha^2 + \alpha$
$\alpha^4 = \alpha + 1$	$\alpha^{12} = \alpha^3 + \alpha^2 + \alpha + 1$
$\alpha^5 = \alpha^2 + \alpha$	$\alpha^{13} = \alpha^3 + \alpha^2 + 1$
$\alpha^6 = \alpha^3 + \alpha^2$	$\alpha^{14} = \alpha^3 + 1$
$\alpha^7 = \alpha^3 + \alpha + 1$	$\alpha^{15} = 1$
$\alpha^8 = \alpha^2 + 1$	$\alpha^{16} = \alpha$

Polinomska baza obsega $\mathbb{F}_{2^4}$ *z izbranim nerazcepnim polinomom* $f(x) = x^4 + x + 1$ *je* $B = \{\alpha^3, \alpha^2, \alpha, 1\}$.

3.2 Normalne baze

Definicija 3.4. *Naj bo* $\alpha \in \mathbb{F}_{q^m}$*. Bazi vektorskega prostora* $\mathbb{F}_{q^m}$ *nad* $\mathbb{F}_q$*, oblike* $N = \{\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}\}$*, pravimo* ***normalna baza****.*

Ker je razsežnost prostora enaka m, mora baza N vsebovati m elementov oziroma, elementi $\alpha, \alpha^q, \ldots \alpha^{q^{m-1}}$ morajo biti linearno neodvisni nad $\mathbb{F}_q$. Elementu α pravimo **generator normalne baze** N oziroma **normalen element** obsega $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$. Normalno bazo običajno zapišemo kot $N = \{\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\}$, kjer je $\alpha_i = \alpha^{q^i}$ za $i = 0, 1, \ldots m-1$.

Izrek 3.5. *Za vsako naravno število m in vsako tako praštevilo p, da je $q = p^n$ za nek $n \in \mathbb{N}$, obstaja normalna baza obsega $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$.* ■

Pred nadaljevanjem poglavja bomo najprej spoznali preslikavo, ki jo imenujemo **sled** in jo označimo s Tr(). Sledila pa bo še definicija dualnih baz.

Definicija 3.6. *Naj bo $\alpha \in \mathbb{F}_{q^m} = F$ in $K = \mathbb{F}_q$. **Sled** za element α nad K je preslikava iz F v K, definirana kot*

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

Definicija 3.7. *Naj bosta $\overline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ in $\overline{\beta} = \{\beta_1, \beta_2, \ldots, \beta_m\}$ bazi nekega vektorskega prostora. Rekli bomo, da sta $\overline{\alpha}$ in $\overline{\beta}$ **dualni**, če za $1 \leq i, j \leq m$ velja*

$$\mathrm{Tr}(\alpha_i \beta_j) = \delta_{ij}, \tag{3.1}$$

kjer predstavlja δ_{ij} Kroneckerjev delta.

Opišimo sedaj eno izmed lastnosti normalnih baz, ki nas bo zanimala tudi v nadaljevanju poglavja. Naj bo $N = \{\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\}$ normalna baza vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$. Označimo $\alpha_i = \alpha^{q^i}$. Za $0 \leq i \leq m-1$ naj bo

$$\alpha\alpha_i = \sum_{j=0}^{m-1} t_{ij}\alpha_j, \tag{3.2}$$

kjer so $t_{ij} \in \mathbb{F}_q$ in je $T = (t_{ij})_{i,j=1,\ldots,m}$ matrika velikosti $m \times m$ nad $\mathbb{F}_q$. Matriko T imenujemo **multiplikacijska tabela** za normalno bazo N.

Naj bo $B = \{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ dualna baza N, kjer prav tako velja $\beta_i = \beta^{q^i}$. Za $0 \leq i, j \leq m-1$ velja

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

kjer je $d_{ij} \in \mathbb{F}_q$ in $D = (d_{ij})_{i,j=1,\ldots,m}$ matrika.

Pokažimo, da za vse $0 \leq i, j \leq m-1$ velja

$$d_{ij} = t_{ji} \quad \text{oziroma} \quad D = T^T \tag{3.3}$$

Po definiciji dualne baze velja $\mathrm{Tr}(\alpha_i\beta_j) = \delta_{ij}$. 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 (3.3).

Pri normalnih bazah nas predvsem zanima lastnost normalne baze N, ki jo imenujemo **kompleksnost** baze N.

Definicija 3.8. ***Kompleksnost baze*** *N označena s c_N je definirana kot število neničelnih t_{ij} v n izrazih oblike*

$$\alpha \cdot \alpha^{q^i} = \sum_{j=0}^{n-1} t_{ij}\alpha^{q^j}, \;\; 0 \le i \le n-1.$$

Komleksnost baze nam pove tudi število neničelnih elementov multiplikacijske tabele T.

Izrek 3.9. *Za poljubno normalno bazo N vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$ velja*

$$c_N \ge 2n - 1.$$

Dokaz. Naj bo $N = \{\alpha^{m-1}, \alpha^{m-2}, \ldots, \alpha, 1\}$ normalna baza za $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$. Potem je $b = \sum_{k=0}^{m-1} \alpha^{q^k} = \mathrm{Tr}(\alpha)$ element obsega $\mathbb{F}_q$. Če upoštevamo zgornjo enačbo (3.2) in primerjamo koeficiente za α_k dobimo

$$\sum_{i=0}^{m-1} \alpha^{q^k} = \begin{cases} b, & i = 0 \\ 0, & 1 \le l \le m-1 \end{cases}$$

Ker je $\alpha \neq 0$ in ker je tudi $\{\alpha\alpha_i \mid 0 \le i \le m-1\}$ 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 enaka nič(zaradi enačbe (3.1), morata biti za vsak $j \neq 0$ vsaj dva neničelna elementa t_{ij}. Torej je vsaj $2m-1$ neničelnih elementov v T. ■

3.3 Optimalne normalne baze

Nas predvsem zanimajo normalne baze z najmanjšo kompleksnostjo $c_N = 2m - 1$ saj je zaradi tega aritmetika precej lažja.

Oblika in definicija

Pred formalno definicijo optimalnih normalnih baz si podrobneje poglejmo kompleksnost normalnih baz.

Vzemimo normalno bazo $N = \{\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\}$ prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$, kjer je $\alpha_i = \alpha^{q^i}$ in $\alpha \in \mathbb{F}_{q^m}$. Za poljubna dva indeksa i, j je produkt $\alpha_i\alpha_j$ linearna kombinacija elementov iz baze $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ in koeficientov t_{ij} iz $\mathbb{F}_q$. Ker velja zveza

$$\alpha_i\alpha_j = (\alpha\alpha_{i-j})^{q^i},$$

je dovolj poznati $\alpha\alpha_i$.

Če v enačbi (3.2) upoštevamo, da je $\alpha_0 = \alpha$, lahko zapišemo

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

Definicija 3.10. *Normalni bazi N obsega $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$ pravimo **optimalna normalna baza**, če zanjo velja $c_N = 2m - 1$.*

Konstrukcija

Gao in Lenstra sta v [7] pokazala dva načina za konstrukcijo optimalne normalne baze. Opisana sta v spodnji trditvi. Dokaz prav tako najdemo v [7].

Trditev 3.11. *Končen obseg $\mathbb{F}_{q^m}$ ima optimalno normalno bazo natanko tedaj, ko drži ena izmed naslednjih dveh trditev:*

(i) Naj bo $m + 1$ praševilo 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 N 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. Naj bo γ primitiven $(2m + 1)$–vi koren enote. Potem $\beta = \gamma + \gamma^{-1}$ generira optimalno normalno bazo N vektorskega prostora $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$,

Optimalnim normalnim bazam, ki ustrezajo prvemu oziroma drugemu pogoju pravimo optimalne normalne baze **tipa I** oziroma **tipa II**. Minimalni polinom za tip I je očitno nerazcepen polinom $g(x) = x^n + \cdots + x + 1$. Minimalni polinom za tip II pa ustreza naslednji rekurzivni zvezi iz [2]:

$$f_0(x) = 1,\ f_1(x) = x + 1,\ f_n(x) = xf_{n-1}(x) - f_{n-2}(x) \text{ za } n \geq 2.$$

Naj naštejem nekaj vrednosti m, za katere obstaja optimalna normalna baza tipa I:

$$2, 4, 10, 12, 18, 28, 36, 52, 58, 60, 66, 82$$
$$100, 106, 130, 138, 148, 162, 172, 178, 180, 196$$
$$210, 226, 268, 292, 316, 346, 348, 372, 378, 388$$

in tipa II (le najpomembnejše) [1]:

$$2, 3, 5, 11, 23, 29, 41, 89, 113, 191, 233, 239, 281, 359.$$

Poglejmo si, kako dejansko zgornjo trditev uporabimo za konstrukcijo optimalne normalne baze.

Baze tipa I

Uporabimo predpostavke iz točke (i) trditve (3.11.). Naj bo α primitiven $(m+1)$–vi koren enote in $m+1$ praštevilo. Ker je $x^m + \cdots + x + 1$ minimalni polinom za baze tipa I po točki (i) trditve (3.11.) velja, da je α 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 iz $\mathbb{F}_{q^m}$. q je primitiven v $\mathbb{Z}_{m+1}$, zato obstaja m različnih konjugirank elementa α, ki so seveda tudi $m+1$–vi koreni enote. Sledi torej:

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

Vidimo, da je N normalna baza vektorskega prostora $\mathbb{F}_{q^m}$ nad $\mathbb{F}_q$. Poglejmo sedaj, kako poteka množenje. Za $1 \leq i \leq m$ velja

$$\alpha\alpha^i = \alpha^{i+1} \in N \tag{3.4}$$

oziroma

$$\alpha\alpha^m = 1 = -\mathrm{Tr}(\alpha) = -\sum_{i=1}^{m} \alpha^i. \tag{3.5}$$

Zato je neničelnih vrednosti v mešanih produktih $\alpha^i\alpha^j$ za $i \neq j$ natanko $2m-1$ in zato je po definiciji optimalne normalne baze tudi N optimalna normalna baza. Sedaj si oglejmo še lastnosti multiplikacijske matrike T, ki pripada bazi N. V vsaki vrstici je natanko ena enica z izjemo vrstice, kjer so same enice. Vsi ostali elementi so očitno enaki 0 zaradi enačb (3.4) in (3.5).

Za lažje razumevanje si oglejmo konkreten primer.

[1] Naštel sem le praštevilske vrednosti, saj so pomembne za kasnejšo implementacijo. Za celoten seznam glej [2]

Primer 3.12. ***Primer optimalne normalne baze za obseg*** $\mathbb{F}_{2^4}$
Vzemimo obseg $\mathbb{F}_{2^4}$*. Število* $m+1=5$ *je praštevilo in primitiven peti koren enote v* $\mathbb{Z}_5$ *je število* 2*. Torej po trditvi (3.11.) obstaja optimalna normalna baza. Naj bo* α *ničla minimalnega polinoma* $f(z)=x^4+x^3+x^2+x+1$ *oziroma primitivni* 5*–i koren enote. Vidimo, da* $\alpha=2$*. Množica* $\{\alpha,\alpha^2,\alpha^4,\alpha^8\}$ *mora seveda sestavljati optimalno normalno bazo za* $\mathbb{F}_{2^4}$ *nad* $\mathbb{F}_2$*. Preverimo, če slednje drži!*

$$\begin{aligned}&\alpha\\&\alpha^2\\&\alpha^4=\alpha^3+\alpha^2+\alpha+1\\&\alpha^8=\alpha^3\end{aligned}$$

Preveriti je potrebno še, ali so vsi elementi baze tudi ničle minimalnega polinoma $f(x)$*.*

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

Vidimo, da so elementi α,α^2,α^4 *in* α^8 *res linearno neodvisni. Dobimo multiplikacijsko matriko za našo bazo:*

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

V matriki T *je natanko* $7=2m-1=2\cdot 4-1$ *enic. S tem zaključujemo primer o optimalnih bazah tipa I.*

Baze tipa II

Sedaj si podrobneje poglejmo še optimalne normalne baze tipa II. Privzemimo predpostavke iz točke (ii) trditve (3.11.). Naj bo $2m+1$ praštevilo in naj bo množica $\mathbb{Z}^*_{2m+1}$ generirana z 2 in -1. Naj bo γ primitiven $(2m+1)$–vi koren enote.Vzemimo $\beta\in\mathbb{F}_{2^m}$. Iz trditve sledi, da so $\beta,\beta^2,\ldots,\beta^{2^{m-1}}$ linearno neodvisni nad $\mathbb{F}_2$. Normalno bazo N vektorskega prostora $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$ sestavlja množica $N=\{\beta,\beta^2,\ldots,\beta^{2^{m-1}}\}$. Sedaj vpeljimo oznako

$$\beta^i=\gamma^i+\gamma^{-i}. \tag{3.6}$$

Seveda lahko bazo zapišemo tudi kot

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

Množici sta enaki, vendar se vrstni red elementov razlikuje. Več o tem bomo spoznali v poglavju 4.

Mešani produkti elementov baze se izražajo na naslednji način:

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

Na tak način se da vsak element iz obsega izraziti z elementi iz baze. Vseh neničelnih mešanih produktov je natanko $2m-1$, torej je N optimalna normalna baza tipa II za vektorski prostor $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$.

Primer 3.13. *Vzemimo obseg $\mathbb{F}_{2^3}$. Število $2m+1 = 7$ je res praštevilo, $m = 3$ je liho in 2 generira kvadratne ostanke v $\mathbb{Z}_7$. Po trditvi (3.11.) obstaja v $\mathbb{F}_{2^3}$ ONB tipa II, generirana z $\beta = \gamma + \gamma^{-1}$, kjer je γ primitivni 7–i koren enote oziroma ničla minimalnega polinoma $f_3(x) = x^3 + x^2 + 1$.*

Ker je iskanje ničel minimalnih polinomov lahko zahtevno, nam naslednja trditev pokaže, kako konstruirati normalno bazo z nizko kompleksnostjo.

Trditev 3.14. *Naj bo q praštevilo ali potenca praštevila ter m in k takšni naravni števili, da je $(mk+1)$ praštevilo, ki ne deli števila q. Naj bo β primitiven $(mk+1)$–vi koren enote v $\mathbb{F}_{q^{mk}}$ in p karakteristika obsega $\mathbb{F}_q$. Predpostavimo še, da je $\mathrm{GCD}(mx/e, m) = 1$, kjer je e red elementa q po modulu $(mk+1)$. Potem za vsak primitiven k–ti element enote τ 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 $((k+1)m-1)$ oziroma največ $(km-1)$, če je $k = 1 \bmod p$.

Dokaz trditve najdemo v [7]. Element α imenujemo **Gaussova perioda** tipa (m, k).

Poglavje 4

Chebysheve baze

V prejšnjih poglavjih smo spoznali osnovne lastnosti najpogostejših dveh tipov baz končnih obsegov. Predstavljeno znanje nam omogoča definicijo **Chebyshevih** oziroma **umbralnih** baz, ki so osrednja tema te diploma.

V polinomskih bazah se izkaže, da je operacija seštevanja hitrejša kot v ostalih bazah. Optimalne normalne baze so primerne za množenje, saj se izvaja hitrejše kot v ostalih bazah. Chebysheve baze pa ponujajo kompromis med slednjima dvema tipoma baz, saj omogočajo tako hitro seštevanje in množenje kot tudi hitro kvadriranje, korenjenje in invertiranje.

Izkaže se, da so Chebysheve baze normalne optimalne baze tipa II, vendar s permutiranim vrstnim redom elementov. Zato bomo spoznali konstrukcijo Chebyshevih baz s pomočjo permutacije elementov in definirali permutacijo samo. Iz lastnosti optimalnih normalnih baz tipa II bomo izpeljali osnovne lastnosti Chebyshevih baz ter nadaljevali z nekaterimi bolj pomembnimi lastnostmi. Te nam bodo omogočile izpeljavo nekaterih naivnih algoritmov aritmetike v Chebyshevih bazah, npr. kvadriranje. Pojasnili bomo tudi ozadje za dvema bolj zapletenima algoritmoma: Evklidovim algoritmom za izračun inverznega elementa in množenjem.

4.1 Definicija in konstrukcija

Naj bo $\mathbb{F}_{2^m}$ obseg nad $\mathbb{F}_2$. Iz prejšnjega poglavja(izrek (3.11.)) vzemimo optimalno normalno bazo N tipa II. Vemo, da je generator te baze element $\beta = \gamma + \gamma^{-1}$, kjer je $\gamma \in \mathbb{F}_{2^m}$ primitiven $(2m+1)$–vi koren enote.

Definirajmo

$$\beta_i = \beta^i = \gamma^i + \gamma^{-i} \quad \text{za} \quad i \in \mathbb{Z}. \tag{4.1}$$

Definicija 4.1. *Bazo* $B = \{\beta_1, \beta_2, \ldots, \beta_m\}$ *imenujemo* ***Chebysheva baza*** *oziroma* ***umbralna(optimalna normalna) baza****.*

Oglejmo si nekaj osnovnih lastnosti elementov β_i Chebysheve baze.

Lema 4.2. *Naj bo* $\mathbb{F}_{2^m}$ *obseg nad* $\mathbb{F}_2$ *in* γ *primitiven* $(m+1)$*–vi koren enote v* $\mathbb{F}_{2^m}$*. Naj bo* $\beta_i = \gamma^i + \gamma^{-i}$ *element Chebysheve baze za* $i \in \mathbb{Z}$*. Potem velja:*

(i) $\beta_0 = 0$,

(ii) $(\beta_k)^2 = \beta_{2k} \quad za \quad \forall k \in \mathbb{Z}$,

(iii) $\beta_{-k} = \beta_k \quad za \quad \forall k \in \mathbb{Z}$,

(iv) $\beta_{k+(2m+1)} = \beta_k \quad za \quad \forall k \in \mathbb{Z}$,

(v) $\beta_i \cdot \beta_j = \beta_{i+j} + \beta_{i-j} \quad za \quad \forall i, j \in \mathbb{Z}$.

Dokaz.

(i) $\beta_0 = \gamma^0 + \gamma^0 = 2\gamma^0 = 0$ v $\mathbb{F}_2$.

(ii) $(\beta_k)^2 = (\gamma^k + \gamma^{-k})^2 = \gamma^{2k} + 2\gamma^{k-k} + \gamma^{-2k} = \gamma^{2k} + \gamma^{-2k} = \beta_{2k}$.

(iii) $\beta_{-k} = \gamma^{-k} + \gamma^k = \gamma^k + \gamma^{-k} = \beta_k$.
Predzadnja enakost velja zaradi komutativnosti seštevanja v obsegu.

(iv) $\beta_{k+(2m+1)} = \gamma^{k+(2m+1)} + \gamma^{-k-(2m+1)} = \gamma^k\gamma^{(2m+1)} + \gamma^{-k}\gamma^{-(2m+1)} = \gamma^k + \gamma^{-k}$,
ker je γ $(2m+1)$–vi koren enote.

(v) $\beta_i \cdot \beta_j = (\gamma^i + \gamma^{-i})(\gamma^j + \gamma^{-j}) = \gamma^{i+j} + \gamma^{i-j} + \gamma^{-(i-j)} + \gamma^{-(i+j)} = \beta_{i+j} + \beta_{i-j}$.

■

Primer 4.3. *Z zgornjo lemo lahko na preprost(lažji kot z množenjem) način izvedemo operacijo kvadriranja v obsegu. Naj bo a element obsega* $\mathbb{F}_{2^5}$ *nad* $\mathbb{F}_2$*, predstavljen kot polinom* $a(\beta) = \beta_4 + \beta_2 + \beta_1$*. Potem je*

$$\begin{aligned}
a(\beta)^2 &= a(\beta) \cdot a(\beta) = (\beta_4 + \beta_2 + \beta_1) \cdot (\beta_4 + \beta_2 + \beta_1) \\
&= \beta_4\beta_4 + \beta_4\beta_2 + \beta_4\beta_1 + \beta_2\beta_4 + \beta_2\beta_2 + \beta_2\beta_1 + \beta_1\beta_4 + \beta_1\beta_2 + \beta_1\beta_1 \\
&= \beta_8 + \beta_4\beta_2 + \beta_4\beta_1 + \beta_2\beta_4 + \beta_4 + \beta_2\beta_1 + \beta_1\beta_4 + \beta_1\beta_2 + \beta_2 \\
&= \beta_8 + \beta_6 + \beta_2 + \beta_5 + \beta_3 + \beta_6 + \beta_{-2} + \beta_4 + \beta_3 + \beta_1 + \beta_5 + \beta_{-3} + \beta_3 + \beta_{-1} + \beta_2 \\
&= \beta_8 + \beta_6 + \beta_2 + \beta_5 + \beta_3 + \beta_6 + \beta_2 + \beta_4 + \beta_3 + \beta_1 + \beta_5 + \beta_3 + \beta_3 + \beta_1 + \beta_2 \\
&= \beta_8 + \beta_4 + \beta_2 \\
&= (\beta_4)^2 + (\beta_2)^2 + (\beta_1)^2.
\end{aligned}$$

Pri tretjem enačaju smo uporabili lastnost (ii), pri četrtem lastnost (v) in pri petem lastnost (iii) zgornje leme. Za boljšo predstavo aritmetike smo izpustili dejstvo, da v obsegih s karakteristiko 2 *velja enakost*

$$(x+y)^2 = x^2 + y^2.$$

Naj bo

$$\{\alpha_0, \alpha_1, \ldots \alpha_{m-1}\} = \{\alpha^{2^0}, a^{2^1}, \ldots, \alpha^{2^{m-1}}\}$$

optimalna normalna baza tipa II. Po definiciji optimalne normalne baze tipa II velja

$$\{\alpha_0, \alpha_1, \ldots \alpha_{m-1}\} = \{\beta_{2^0}, \beta_{2^1}, \ldots, \beta_{2^{(m-1)}}\}.$$

Ker je red elementa β enak $2m+1$, je preprosto preveriti, da je $\beta_i = \beta_j$ če in samo če je $i = \pm j \bmod (2m+1)$. Poleg tega za vsak i, $1 \leq i \leq m$, velja

$$\alpha_i = \alpha^{2^i} = \gamma^{2^i} + \gamma^{2^i} = \beta_{2^i}.$$

Po drugi strani pa, ker je multiplikativna grupa $\mathbb{Z}^*_{2m+1}$ generirana z 2 in -1 obstaja takšno število k, da je $i \equiv \pm 2^k \bmod (2m+1)$ in zato $\beta_i = \alpha^{2^k}$. Tako tudi množica $\{\beta_1, \beta_2, \ldots, \beta_m\}$ sestavlja bazo $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$.

Prehod med bazama

Za lažjo konstrukcijo si poglejmo permutacijo, ki omogoča prehod med optimalno normalno bazo tipa II in Chebyshevo bazo. Oglejmo si zvezo

$$\beta_{2^i} = \gamma^{2^i} + \gamma^{-2^i} = \alpha^{2^i}$$

in elemente β_{2^i} za $i = 1, 2, \ldots, m$.

$$\begin{aligned}
\beta_{2^0} &= \gamma^{2^0} + \gamma^{-2^0} = \alpha^{2^0} = \alpha_0 \\
\beta_{2^1} &= \gamma^{2^1} + \gamma^{-2^1} = \alpha^{2^1} = \alpha_1 \\
\beta_{2^2} &= \gamma^{2^2} + \gamma^{-2^2} = \alpha^{2^2} = \alpha_2 \\
&\ldots \\
\beta_{2^m} &= \gamma^{2^m} + \gamma^{-2^m} = \alpha^{2^m} = \alpha_m
\end{aligned}$$

Potenca 2^i bo očitno presegla vrednost m za nek $i = 1, 2, \ldots, m$. Zato si oglejmo naslednjo zvezo. Iz enakosti $\gamma^{2m+1} = 1$ sledi, da je

$$\gamma^{m+1} = \gamma^{-m}$$

oziroma

$$\beta_{m+1} = \beta_m$$

in še

$$\beta_{m+k} = \beta_{m+1-k}.$$

Natančneje si oglejmo gornjo enakost in opišimo postopek za izračun permutacije elementov β_{2^i}, ko je indeks 2^i večji od m za nek indeks $i = 1, 2, \ldots, m$. Naj bo $I = 2^i$.

$$\beta_I \;=\; \beta_{2^i} = \alpha_i$$

V primeru, da je $2^i > m$ upoštevamo gornje lastnosti in izračunamo

$$k = 2^i - m$$

in

$$I = m + 1 - k = m + 1 - 2^i + m = 2m + 1 - 2^i.$$

Tako dobimo zvezo

$$\beta_I = \beta_{2m+1-2^i} = \alpha_i. \tag{4.2}$$

Elementi umbralne baze se torej permutirajo iz optimalne normalne baze tipa II na naslednji način:

$$\alpha_i = \begin{cases} \beta_{2^i} & \text{če } 2^i < m, \\ \beta_{2m+1-2^i} & \text{sicer} \end{cases} \tag{4.3}$$

Sedaj lahko izpeljemo algoritem za prehod med optimalno normalno bazo tipa II in Chebyshevo bazo. Algoritem za prehod med bazama je opisan v algoritmu 1.

ALGORITEM 1: ***Prehod iz ONB tipa II v Chebysheve baze***

INPUT: ONB tipa II $N = \{\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{m-1}\}$
OUTPUT: Chebysheva baza $B = \{\beta_1, \beta_2, \ldots, \beta_m\}$

---

1. `for` i `=` 0 `to` $m-1$ `do`

   1.1. `if` 2^i `<` m `then` β_{2^i} `=` α_i

   `else` β_{2m+1-2^i} `=` α_i

2. `Return` $BB = \{\beta_1, \beta_2, \ldots, \beta_m\}$;

Množenje v Chebyshevih bazah

Če fiksiramo $k \in \mathbb{Z}$, potem po lastnostih (iii) in (iv) iz leme 4.2. izpeljemo, da obstaja za vsak $k \in \mathbb{Z}$ enolično določen $k' \in \{1, 2, \ldots, m\}$, za katerega velja $\beta_k = \beta_{k'}$. Torej obstaja natanko m različnih neničelnih β_k. Oglejmo si, kako dobimo k'.

Brez škode za splošnost predpostavimo, da je $k \in [-m, 2m+1]$. V primeru, da leži k izven intervala, ga lahko s pravili (iii) in (iv) leme 4.2. reduciramo. Ločimo tri primere:

- $0 \leq k \leq m$: $k' = k$.
- $-m \leq k < 0$: $k' = -k$.
- $m+1 \leq k \leq 2m+1$: potem je $k = m+1+\ell$ za nek $0 \leq \ell \leq m$. Velja tudi

$$\beta_{m+1+\ell} = \beta_{2m+1-m+\ell} = \beta_{2m+1-(m-\ell)} \overset{(iv)}{=} \beta_{-(m-\ell)} \overset{(iii)}{=} \beta_{m-\ell}$$

  Torej, če je $k = m+1+\ell$ je $k' = m-\ell$.

Sedaj lahko izpeljemo pravilo za množenje elementov iz Chebysheve baze:

$$\beta_i \cdot \beta_j = \begin{cases} \beta_{i+j} + \beta_{|i-j|} & \text{za} \quad i+j \leq m, \\ \beta_{2m+1-(i+j)} + \beta_{|i-j|}, & \text{sicer.} \end{cases} \tag{4.4}$$

Oziroma, še natančneje:

$$\beta_i \cdot \beta_j = \begin{cases} \beta_{i+j} + \beta_{|i-j|} & \text{za} \quad i+j \leq m, \\ \beta_{m-(i+j)} + \beta_{|i-j|}, & \text{za} \quad m+1 \leq i+j \leq 2m+1, \\ \beta_{2m+1-(i+j)} + \beta_{|i-j|} & \text{sicer.} \end{cases} \tag{4.5}$$

S formulama (4.4) oziroma (4.5) omejimo indekse na željeno množico $\{1, 2, \ldots, m\}$.

4.2 Evklidov algoritem v Chebyshevih bazah

V tem razdelku bomo spoznali nekaj lastnosti Chebyshevih baz, ki nam omogočijo iskanje inverznega elementa s pomočjo razširjenega Evklidovega algoritma.

Če se vrnemo k točki (v) leme (4.2.) vidimo, da pri množenju elementov iz Chebysheve baze kaj hitro dobimo elemente, katerih stopnja presega največjo stopnjo elementov baze. Definirajmo polinom $p(x) \in \mathbb{F}_m[x]$ stopnje največ m, torej $p(x) = \sum_{i=1}^{m} p_i x^i$, za $p_i \in \mathbb{F}_2$. Ker smo v enačbi (4.1) definirali zvezo med β^i in β_i, lahko sedaj definiramo **umbralni polinom $p(\beta)$**, ki ga zapišemo kot

$$p(\beta) = \sum_{i=1}^{m} p_i \beta^i = \sum_{i=1}^{m} p_i \beta_i. \tag{4.6}$$

Definirajmo **umbralno stopnjo** polinoma $p(\beta)$ kot največji indeks neničelnega koeficienta p_i za $i = 1, 2, \ldots m$. Potem lahko poljuben element iz končnega obsega $\mathbb{F}_{2^m}$ z optimalno normalno bazo predstavimo z umbralnim polinomom stopnje največ m oziroma kot m–terico $(p_1, p_2, \ldots, p_m)$.

Po točki (i) leme (4.2.) vemo, da je $\beta_0 = 0$, tako da lahko vsoto iz enačbe (4.6) razširimo še za ničelen člen $p_0\beta_0 = 0$.

Definicija 4.4. *Naj bo $p(x) \in \mathbb{F}_m[x]$ polinom stopnje največ m.* ***Umbralni polinom*** *imenujemo polinom $p(\beta)$ in ga zapišemo kot*

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

Z definicijo umbralnega polinoma (4.4.) lahko sedaj vsak element obsega $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$ zapišemo kot umbralni polinom stopnje največ m oziroma z $(m+1)$–terico ali vektorjem $(p_0, p_1, p_2, \ldots, p_m)$.

Na dodatno definirano mesto v umbralnem polinomu (samo za potrebe Evklidovega algoritma in kasnejše implementacije) shranimo konstantni člen polinomov oziroma multiplikativno enoto (ki kljub intuitivnemu razmisleku o oznakah ni enaka β_0), ki jo označimo s simbolom 1. Za razjasnitev morebitne zmede poskrbi naslednja trditev.

Trditev 4.5. *Naj bo $\{\beta_1, \beta_2, \ldots \beta_m\}$ Chebysheva baza obsega $\mathbb{F}_{2^m}$. Potem držijo naslednje trditve:*

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

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

(iii) $1 + \sum_{i=1}^{m} \beta_i$ *je nerazcepni umbralni polinom nad $\mathbb{F}_{2^m}$.*

Dokaz.

(i) Naj bo $2m+1$ praštevilo in naj bo množica in γ primitiven $(2m+1)$–vi koren enote. Iz definicije β_i 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 $\mathbb{F}_2[x]$.

(ii) Za dokaz glej [13, IX].

(iii) Predpostavimo, da je polinom $1 + \sum_{i=1}^{m} \beta_i$ razcepen. Torej obstajata dva nekonstantna polinoma $r(\beta)$ in $s(\beta)$ iz $\mathbb{F}_2[m]$, da velja

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

Če razpišemo zgornjo enačbo dobimo

$$1 + \gamma + \gamma^{-1} + \gamma^2 + \gamma^{-2} + \cdots + \gamma^m + \gamma^{-m} = r(\gamma) \cdot s(\gamma)$$

Obe strani pomnožimo z γ^m:

$$\gamma^m + \gamma^{m+1} + \gamma^{m-1} + \gamma^{m+2} + \gamma^{m-2} + \cdots + \gamma^{2m} + 1 = r(\gamma) \cdot s(\gamma) \cdot \gamma^m$$

Sedaj pa pomnožimo še s polinomom $\gamma - 1$. Na levi strani ostane le

$$\gamma^{2m+1} - 1 = r(\gamma) \cdot s(\gamma) \cdot \gamma^m \cdot (\gamma - 1)$$

Element γ je $(2m+1)$–vi primitivni koren enote, kar po definiciji pomeni, da je γ ničla polinoma $x^{2m+1} - 1$, torej ničla polinoma na levi strani zgornje enačbe. Ker je γ primitiven $(2m + 1)$–vi koren enote, tudi ni enaka 1, zato γ tudi ni ničla polinoma $\gamma - 1$ na desni strani enačbe. Seveda $\gamma \neq 0$. Iz tega sledi, da mora biti γ ničla vsaj enega od polinomov $r(\gamma)$ oziroma $s(\gamma)$ na desni. Kar pa pomeni, da smo našli polinom stopnje manjše od m, ki ima γ za ničlo. Kar pa je v protislovju z definicijo γ. Torej je polinom $1 + \sum_{i=1}^{m} \beta_i$ res nerazcepen.

■

V nadaljevanju poglavja sem za predstavitev elementov danega obsega $\mathbb{F}_{2^m}$ uporabil $(m + 1)$–terico oziroma definicijo (4.4.). Razlog tiči v usmeritvi algoritmov k implementaciji, kjer sem uporabil enotno predstavitev elementov ne glede na računsko operacijo (kvadriranje ali invertiranje).
Trditev (4.5.) nam sedaj omogoči izvajanje Evklidovega algoritma za izračun inverznega elementa, ki ga definiramo v poglavju 5.

4.3 Osnovni algoritmi

V tem razdelku bomo spoznali aritmetiko Chebyshevih baz ter odkrili njene prednosti in slabosti. Razdelek predstavi le osnovne algoritme, saj izboljšave obravnavamo v poglavju 5. Najprej obdelamo kvadriranje, nato množenje, ki je tesno povezano z redukcijo in na koncu razdelka še najosnovnejši algoritem za izračun inverza.

4.3.1 Kvadriranje

Algoritem kvadriranje izpeljemo direktno iz leme (4.2.). Za začetek opisa algoritma vzemimo lastnost (ii). Za kvadriranje elementa $a(x) = (a_0, a_1, \ldots, a_m)$ je najpreprostejše izračunati $(2m+1)$–terico

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

Ker je kvadriranje elementov obsega le množenje enakih elementov baze s samimi seboj, saj množimo v obsegu s karakteristiko 2, vidimo, da je edina potrebna operacija

$$\beta_i \cdot \beta_i = (\beta_i)^2 = \beta_{2i}.$$

Tako dobimo pri kvadriranju iz prvotne največ $(m+1)$–terice največ $(2m+1)$– terico elementov umbralnega polinoma.

Na tem mestu lahko potegnemo vzporednice s kvadriranjem v polinomskih bazah, kjer je vmesni rezultat prav tako največ $(2m+1)$–terica. Končni rezultat, torej zopet m–terico pa dobimo tako v polinomskih kot tudi v umbralnih bazah z redukcijo.

Omenili smo že, da je redukcija v polinomskih bazah v primerjavi z redukcijo v umbralnih bazah pri kvadriranju relativno zamudna. Relativno zato, ker se ji pri kvadriranju v umbralnih bazah z lemo (4.2.) in njunima lastnostima (iii) in (iv) lahko elegantno izognemo.

Nadaljujmo torej z lastnostima

$$\beta_{-k} = \beta_k \quad \text{in} \quad \beta_{k+(2m+1)} = \beta_k.$$

Želimo, da bi se pri kvadriranju indeksi, večji od m hkrati že kar v istem koraku kot izračun podvojenega indeksa zreducirali na ustrezen indeks znotraj množice $\{0, 1, \ldots, m\}$. Združimo obe omenjeni lastnosti ter izpeljavo enačbe (4.5)

$$\beta_{m+1+i} = \beta_{(2m+1)-(m-i)} = \beta_{m-i}$$

in prilagodimo pravilo za kvadriranje

$$(\beta_i)^2 = \begin{cases} \beta_{2i} & \text{če je } i \leq m/2, \\ \beta_{m-i} & \text{sicer.} \end{cases}$$

Označimo $\boldsymbol{n = m + 1}$ in definirajmo n kot **sodo** število.

Oglejmo si, kako se permutirajo indeksi znotraj m–terice pri kvadriranju tabeli 4.1.

Strnimo zgornje ugotovitve v algoritem 2 za kvadriranje v Chebyshevih bazah.

Tabela 4.1: Permutacija elementov pri kvadriranju

pred	0	1	...	$n/2-1$	$n/2$	$n/2+1$	...	$n-1$	n
po	0	2	...	$n-2$	n	$n-1$	...	3	1

ALGORITEM 2: ***Kvadriranje v Chebyshevih bazah***

INPUT: Umbralni polinom $p(\beta) \in \mathbb{F}_{2^m}$, $\deg(p) \leq m$
OUTPUT: Umbralni polinom $q(\beta) = p(\beta)^2$, $\deg(q) \leq m$

1. m = m + 1;
2. for i = 0 to m
   2.1. if $i \leq m/2$ then q_{2i} = p_i;
   2.2. else $q_{2m-i} = p_i$;
3. Return $q(\beta)$;

4.3.2 Redukcija

Za izpeljavo redukcije uporabimo pravili (iii) in (iv) leme (4.2.). Najprej združimo njuni lastnosti za redukcijo m–terice[1] elementov z indeksi $m \leq i \leq 2m$ v

$$\beta_{2m+1-i} = \beta_{-i} \quad \text{za} \quad 0 < i \leq m$$

oziroma

$$\beta_{m+1+i} = \beta_{-m+i} \quad \text{za} \quad 0 < i \leq m.$$

Nato uporabimo še lastnost

$$\beta_{-i} = \beta_i$$

za $-m \leq i < 0$.

V algoritmu je vhodni podatek umbralni polinom stopnje manjše ali enake $3m$. Za lažje razumevanje postopka redukcije sem ohranil indekse v prvotnem zapisu, torej znotraj množice $\{-m, -m+1, \ldots, 2m-1, 2m\}$. Za potrebe implementacije je priporočeno množico indeksov preslikati v množico $\{0, 1, \ldots, 3m-1, 3m\}$ tako, da vsakemu indeksu iz prvotne množice prištejemo m.

[1] Ali je leva ali desna m–terica je v domeni implementacije

Redukcija je opisana v algoritmu 3.

ALGORITEM 3: ***Redukcija v Chebyshevih bazah***

INPUT: Umbralni polinom $p(\beta)$ stopnje $\deg(p) \leq 3m$
OUTPUT: Reducirani umbralni polinom $q(\beta)$ stopnje $\deg(q) \leq m$

1. i = m; j = $-m$; k = $m+1$;
2. while $i \geq 1$ do
   2.1. q_i = p_i + p_j + p_k;
   2.2. i = i - 1; j = j + 1; k = k + 1;
3. Return $q(\beta)$;

4.3.3 Množenje

Tudi za izpeljavo algoritma za množenje izhajamo iz leme (4.2.), hkrati pa povlečemo vzporednice s polinomskimi bazami.

Tudi pri množenju bomo algoritem razdelili na dva dela: na množenje umbralnih polinomov in redukcijo umbralnega polinoma na stopnjo največ m. Za množenje uporabimo lastnost umbralnih baz

$$\beta_i \cdot \beta_j = \beta_{i+j} + \beta_{i-j}, i,\ j = 0, \ldots, m.$$

Tako dobimo kot rezultat $(3m+1)$–terico elementov. V polinomskih bazah dobimo v istem koraku $(2m)$–terico. Ker predstavljamo le osnovni algoritem za množenje v umbralnih bazah, se zadovoljimo z dobljeno dolžino rezultata.

Množenje $c(\beta) = a(\beta) \cdot b(\beta)$ izvajamo z iteriranjem(indeks $i, 1 \leq i \leq m$) po elementih a_i polinoma $a(\beta)$ ter s sprotnim računanjem pomika polinoma $b(\beta)$ v za i mest levo in desno. S tem se izognemo iteriranju po obeh polinomih, saj bi bilo sicer za izračun rezultata c_{i+j} in c_{i-j} potrebno pregledati ob vsakem elementu a_i polinoma $a(\beta)$ vse elemente b_j polinoma $b(\beta)$ za indeks j, $1 \leq j \leq m$. S tem na vsakem koraku i izračunano vrednost elementa $b_{j\pm i}$, $1 \leq j \leq m$ oziroma $b(\beta)^i$ in $b(\beta)^{-i}$ ter s tem prihranimo siceršnjo dvojno zanko v implementaciji.

Metodo imenujemo **pomakni in prištej** in je opisana v algoritmu 4.

ALGORITEM 4: ***Pomakni in prištej***

INPUT: Umbralna polinoma $a(\beta)$ in $b(\beta)$
OUTPUT: Umbralni polinom $c(\beta) = a(\beta) \cdot b(\beta)$

1. c = 0; lb = b; rb = b;
2. for i = 1 to m do
   2.1. lb = left_shift(lb); rb = right_shift(rb);
   2.2. if a_i == 1 then c = c + lb + rb;
3. Redukcija($c(\beta)$);
4. Return $c(\beta)$;

4.3.4 Inverz

Razširjeni Euklidov algoritem je že preveč sofisticiran algoritem za to poglavje, zato bom predstavil najosnovnejši algoritem za iskanje inverza v končnih obsegih. Imenujemo ga **izračun inverza z grobo silo**(*brute force inverse*).

Naj bo e multiplikativna enota grupe G. Oglejmo si nekaj lastnosti grup, povzetih po [13]. Vemo, da za vsako grupo G in element $a \in G$ velja, da je

$$a^{\text{moč grupe}} = e.$$

Moč multiplikativne grupe $\mathbb{F}_{q^m}$ pa je enaka $q^m - 1$ in zato velja za element $a \in \mathbb{F}_{q^m}$

$$a^{q^m-1} = e$$

oziroma

$$a^{q^m} = a.$$

Vidimo, da iz enakosti lahko izluščimo inverz, če obe strani pomnožimo z inverznim elementov, torej

$$e^{q^m-1} = 1$$

oziroma

$$e^{q^m-2} \cdot e = 1.$$

Vidimo, da je $e^{-1} = e^{q^m-2}$. Potrebno je le še razmisliti, kako najučinkoviteje doseči $(q^m - 2)$–go potenco elementa e. Nadomestimo q z v implementaciji uporabljeno vrednostjo 2 in razpišimo gornji eksponent:

$$2^m - 2 = 2 + 2^2 + \cdots + 2^{m-1}.$$

Iz tega sledi, da lahko $(2^m - 2)$–go potenco elementa x zapišemo kot

$$x^{-1} = x^{2^m-2} = (x^2)(x^{2^2}) \cdot \cdots \cdot (x^{2^{m-1}}).$$

ALGORITEM 5: ***Inverz z grobo silo (Wangova metoda)***

INPUT: Umbralni polinom $a(\beta)$
OUTPUT: Umbralni polinom $b(\beta) = a(\beta)^{-1}$

---

1. b = a;
2. for k = 1 to $m-2$ do
   2.1. c = b^2;
   2.2. b = $c \cdot b$;
3. b = b^2;
4. Return $b(\beta)$;

O sami zahtevnosti algoritma in možnih optimizacijah bom pisal v poglavju 5. Oglejmo si še ilustrativen primer.

Primer 4.6. *Vzemimo obseg $\mathbb{F}_{2^{11}}$, kjer predstavimo elemente z dodanim ničelnim bitom. Vemo, da je $2^{11} - 2 = 2046$. Tokrat pišemo elemente obsega $\mathbb{F}_{2^m}$ kot vektor z elementi iz $\{0, 1\}$. Vhodni podatek je element*

$$a = (010000101111)$$

Potek algoritma je opisan v tabeli 4.2. Rezultat, da je $a^{-1} = a^{2046}$, je pravilen, saj lahko vidimo, da

$$a \cdot a^{-1} = a \cdot a^{2046} = a^{2047} = (111111111111)$$

Kar je enota za množenje[2].

[2]Čeprav smo v začetku razdelka v trditvi (4.5.) definirali multilikativno enoto kot $\sum_{i=1}^{m} \beta_i$, je v zgornjem zapisu skrajno desna enica zgolj stvar implementacije in ne spremenjene multipliaktivne enote.

Tabela 4.2: Iskanje inverza z grobo silo

Stopnja	Element
1	010000101111
2	010001011101
3	000101101101
6	110011010001
7	101111010101
14	101110110011
15	010111101110
30	111011011100
31	000001010101
62	100100010001
63	001110100110
126	011010110100
127	001110110111
254	011110110101
255	100101111011
510	110111000111
511	001001001111
1022	100001110101
1023	111100001100
2046	**000011000000**
2047	**111111111111**

Poglavje 5

Učinkoviti algoritmi

Jedro diplomskega dela predstavlja implementacija vseh algoritmov, ki sestavljajo aritmetiko Chebyshevih baz. Ker implementacija predstavlja otipljivo plat algoritmov oziroma ponudi uporabniku orodje, katerega hitrost se lahko meri, je bilo potrebno izpeljati hitrejše algoritme od tistih, opisanih v poglavju 4. Zato v tem poglavju uvedemo algoritme, ki prihranijo uporabniku precej časa pri npr. izračunu inverznega elementa.

Izboljšave so predvsem praktične narave, začinjene z nekaj računalniškimi prijemi. Algoritme bom poizkusil predstaviti karseda praktično in iz implementacijskega (programerskega) vidika.

V poglavju 4 sem naštel nekaj osnovnih algoritmov za kvadriranje, množenje, redukcijo in iskanje inverznega elementa. Algoritmi so v vseh pogledih pravilni in zagotovo vrnejo pravilen rezultat. Vendar pa so s stališča implementacije sila neučinkoviti. Čeprav je preprosto prepisovanje bitov pri kvadriranju in redukciji na prvi pogled lično se v implementaciji spremeni v zamudno premikanje in iskanje bitov znotraj besed.

V tem poglavju predpostavim, da so elementi obsega $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$ predstavljeni v vektorski obliki $(\beta_m\ \beta_{m-1}\ \ldots\ \beta_1\ \beta_0)$, kjer je element baze β_i predstavljen kot bit z vrednostjo $\beta_i \in \{0, 1\}$ za vsak $i \in \{0, 1, \ldots, m\}$.

Poglavje bom začel z opisom algoritma, ki hitreje potencira posamezne elemente baze. Sledil bo hitrejši algoritem za kvadriranje s pomočjo predhodno izračunanih tabel. Taiste tabele bom uporabil tudi v izboljšanem algoritmu za redukcijo stopnje umbralnega polinoma. Nadaljeval bom z učinkovitejšim algoritmom za množenje, poglavje pa bom zaključil z dvema učinkovitima algoritmoma za izračun inverznega elementa.

5.1 Izboljšano potenciranje elementa baze

Vzemimo naravno število e in naj $v(e)$ predstavlja število enic v binarnem zapisu števila e.

Ker je $v(e) \leq n$ za $0 \leq e \leq 2^n - 1$ vemo, da lahko potenco α^e izračunamo v $\mathcal{O}(n^2)$ bitnih operacijah. Za primerjavo naj omenim, da sta *Stinson* v [9] in *Von Zur Gathen* v [10] pokazala, da lahko za poljuben $\beta \in \mathbb{F}_{2^m}$, če je $\mathbb{F}_{2^n}$ predstavljen v optimalni normalni bazi, potenco β^e izračunamo v približno $\mathcal{O}(n/\log_2 m)$ množenjih v $\mathbb{F}_{2^n}$, torej z $\mathcal{O}(n^3/\log_2 m)$ bitnimi operacijami, če smatramo kvadriranje kot zastonj operacijo znotraj optimalne normalne baze in množenje zahtevnosti $\mathcal{O}(n^2)$ bitnih operacij.

V polinomskih bazah je mogoče z uporabo hitrih algoritmov za množenje potenco β^e izračunati z metodo *kvadriraj in množi* v $\mathcal{O}(n \log n \log\log n \log e)$.

Izrek 5.1. *Naj ima α lastnost iz izreka (3.11.), tj. generira optimalne normalne baze tipa II. Za vsako naravno število e, $e \leq 2^n - 1$ lahko potenco α^e izračunamo v $\mathcal{O}(n \cdot v(e))$ bitnih operacijah.*

Spomnimo se, kako smo napravili prehod iz optimalnih normalnih baz tipa II do umbralnih baz. Permutirali smo elemente s s pravilom, opisanim v enačbi (4.3). Definirajmo sedaj preslikavo $s : \mathbb{Z} \to \{0, 1, \ldots, m\}$ in preslikavo $p : \mathbb{Z} \to \{0, 1, \ldots, 2m\}$.

$$p(i) = \begin{cases} i \bmod (2m+1) & \text{če } 0 \leq i, \\ -i \bmod (2m+1) & \text{sicer,} \end{cases} \tag{5.1}$$

$$s(i) = \begin{cases} p(i) & \text{če } 0 \leq p(i), \\ 2m + 1 - p(i) & \text{sicer.} \end{cases} \tag{5.2}$$

Zaradi lastnosti elementov Chebyshevih baz $\beta_i \cdot \beta_j = \beta_{i+j} + \beta_{i-j}$ velja tudi

$$\beta_i \cdot \beta_j = \beta_{s(i+j)} + \beta_{s(i-j)}, \quad \text{za} \quad 1 \leq i, j \leq m.$$

Definirajmo

$$\gamma_i = \gamma^i + \gamma^{-i} = \beta_i.$$

Sedaj pokažimo, kako izračunamo produkt $\gamma_i \cdot A$, kjer je $1 \leq i \leq m$ in A poljuben element iz $\mathbb{F}_{2^m}$. Predpostavimo, da je $A = \sum_{k=1}^{n} a_k \gamma_k$, kjer je $a_k \in \mathbb{F}_2$. Potem velja

$$\gamma_i \cdot A = \sum_{k=1}^{n} a_k \gamma_i \cdot \gamma_k = \sum_{k=1}^{n} a_k (\gamma_{s(k+i)} + \gamma_{s(k-i)}).$$

Opazimo, da

$$\begin{aligned}
\sum_{k=1}^{n} a_k\gamma_{s(k+i)} &= \sum_{k=1}^{n-i} a_k\gamma_{k+i} + \sum_{k=n+1-i}^{n} a_k\gamma_{2n+1-(k+i)} \\
&= \sum_{k=i+1}^{n} a_{k-i}\gamma_k + \sum_{k=n+1-i}^{n} a_{2n+1-(k+i)}\gamma_k \\
&= \sum_{k=i+1}^{n} a_{s(k-i)}\gamma_k + \sum_{k=n+1-i}^{n} a_{s(k+i)}\gamma_k, \\
\sum_{k=1}^{n} a_k\gamma_{s(k-i)} &= \sum_{k=1}^{i} a_k\gamma_{k-i} + \sum_{k=i+1}^{n} a_k\gamma_{k-i} \\
&= \sum_{k=1}^{i} a_{i-k}\gamma_k + \sum_{k=1}^{n-i} a_{n+k}\gamma_k \\
&= \sum_{k=1}^{i} a_{s(k-i)}\gamma_k + \sum_{k=1}^{n-i} a_{s(k+i)}\gamma_k,
\end{aligned}$$

kjer postavimo $a_0 = 0$. Vidimo, da

$$\begin{aligned}
\gamma_i \cdot A &= \sum_{k=1}^{n} (a_{s(k-i)} + a_{s(k+1)})\gamma_k \\
&= \sum_{k=1}^{c} (a_{i-k} + a_{k+i})\gamma_k + \sum_{k=c+1}^{d} f(k)\gamma_k + \sum_{k=d+1}^{n} (a_{k-i} + a_{2n+1-(k+i)})\gamma_k,
\end{aligned}$$

kjer je

$$\begin{aligned}
c &= \min\{\, i, n-i \,\}, \\
d &= \max\{\, i, n-i \,\} = n - c \quad \text{in} \\
f(k) &= \begin{cases} a_{i-k} + a_{2n+1-(k+i)}, & \text{če } i > n-i, \\ a_{k-i} + a_{k+i} & \text{če } i < n-i. \end{cases}
\end{aligned}$$

S tem smo pokazali, da lahko produkt $\gamma_i \cdot A$ izračunamo v $\mathcal{O}(n)$ bitnih operacijah.

Sedaj pa za izračun potence α^e predpostavimo, da je $0 \leq e < 2^n - 1$, saj je $\alpha^{2^n-1} = 1$. Zapišimo

$$e = \sum_{k=0}^{n-1} e_k 2^k, \quad \forall e_k \in \{0, 1\}.$$

Potem je

$$\alpha^e = \prod_{k=0}^{n-1} (\alpha^{2^k})^{e^k} = \prod_{k=0}^{n-1} (\gamma_{s(2^k)})^{e^k}.$$

Algoritem 6 opisuje zgornji postopek.

ALGORITEM 6: ***Potenciranje elementa Chebysheve baze***

INPUT: število e, $0 \leq e \leq 2^n - 1$
OUTPUT: število α^e, kjer je $\alpha = \gamma + \gamma^{-1}$

---

1. $a = 1 = \sum_{k=1}^{n} \gamma_k$;
2. Izračunaj binarno reprezentacijo $e = \sum_{k=0}^{n-1} e_k 2^k$;
3. for $i = 0$ to $n - 1$
   3.1. if $e_k == 1$ then $a = \gamma_{s(2^k)} \cdot a$;
4. Return a;

5.2 Izboljšana kvadriranje in redukcija

V tem razdelku bom opisal izboljšave kvadriranja in redukcije s pomočjo predhodno izračunanih tabel.

5.2.1 Predhodno izračunane tabele

V algoritmu 2 se izvede kvadriranje s prepisom elementa na i–tem mestu na $2i$–to mesto razultata. Za dokončanje celotnega postopka potrebujemo toliko prepisov kot je stopnja umbralnega polinoma (število bitov).

Ker v računalniški aritmetiki ne operiramo z biti ampak z besedami[1] je premikanje in iskanje posameznih bitov znotraj posameznih besed časovno draga operacija. Zato je bolje uporabiti predhodno izračunane tabele, ki hkrati prepišejo po 8 ali 16 bitov besede v enem koraku. Tabele si lahko predstavljamo tudi kot funkcije

$$\text{Tabela}(a) = f(a),$$

[1]Beseda je podatkovni tip, ki vsebuje določeno število bitov, npr. 16 bitni *short integer*

kjer je spremenljivka a vhodna beseda, rezultat $f(a)$ pa je beseda ustrezne željene dolžine in oblike. V našem primeru definiramo tri tipe predhodno izračunanih tabel: tabele za **razširjanje bitov**, **obračanje bitov** in **hkratno razširjanje in obračanje bitov**. Razširjanje bitov je postavljanje ničel za vsak prvoten bit na naslednji način:

$$011010101 \rightarrow 000101000100010001.$$

Primeri tabel so podani v tabeli 5.1.

5.2.2 Kvadriranje

S tabelami lahko sedaj znatno pohitrimo proces kvadriranja elementov iz obsega $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$.

Uporabimo dejstvo, da element iz obsega predstavimo kot umbralni polinom iz definicije (4.4.). S tem zapisom dosežemo sodo število mest v vektorskem zapisu elementa obsega in se tako izognemo težavi pri iskanju $\lfloor m/2 \rfloor$. Prav tako upoštevamo dejstvo, da je m lih, vendar mu dodamo še eno mesto v zapisu umbralnega polinoma, zato se delamo, da je m sodo število. Z izjemo primera $m = 2$, ki ga ne obravnavamo.

Torej, od tukaj naprej je $\boldsymbol{m}$ **sodo število**.

Izboljšano kvadriranje je opisano v algoritmu 7.

5.2.3 Redukcija

Osnovni postopek redukcije je opisan v algoritmu 3. Tudi tukaj je iskanje posameznih bitov dolgotrajen postopek. Zato poenostavimo redukcijo s pomočjo do sedaj še ne uporabljene tabele za obračanje bitov.

V prvem koraku zaradi lastnosti

$$\beta_{m+1+i} = \beta_{-m+i} \quad \text{za} \quad m < i \leq 2m$$

Chebyshevih baz prenesemo bite z indeksi $(m+1, m+2, \ldots, 2m)$ preko bitov z indeksi $(-m, -m+1, \ldots, -1)$ in jih seštejemo. V drugem koraku pa upoštevamo lastnost

$$\beta_{-i} = \beta_i \quad \text{za} \quad 1 \leq i \leq m$$

in prezrcalimo $(m-1)$ bitov $(\beta_{-1}, \beta_{-2}, \ldots, \beta_{-m})$ preko ničelnega bita β_0 na začetne bite $(\beta_1, \beta_2, \ldots, \beta_m)$. Postopek izvajamo v prvem koraku z besedami, v drugem koraku pa s predhodno izračunanimi tabelami. Postopek opisuje algoritem 8.

Tabela 5.1: Primeri predhodno izračunanih tabel

tabela za razširjanje bitov

```
square_table[00000000] = 0000000000000000
square_table[00000001] = 0000000000000001
square_table[00000010] = 0000000000000100
square_table[00000011] = 0000000000000101
⋮
square_table[11111110] = 0101010101010100
square_table[11111111] = 0101010101010101
```

tabela za obračanje bitov

```
reverse_table[00000000] = 00000000
reverse_table[00000001] = 10000000
reverse_table[00000010] = 01000000
reverse_table[00000011] = 11000000
⋮
reverse_table[11111110] = 01111111
reverse_table[11111111] = 11111111
```

tabela za hkratno razširjanje in obračanje bitov

```
reverse_square_table[00000000] = 0000000000000000
reverse_square_table[00000001] = 1000000000000000
reverse_square_table[00000010] = 0010000000000000
reverse_square_table[00000011] = 1010000000000000
⋮
reverse_square_table[11111110] = 0010101010101010
reverse_square_table[11111111] = 1010101010101010
```

ALGORITEM 7: ***Kvadriranje s tabelami***

INPUT: umbralni polinom $p(\beta) = \sum_{i=0}^{m} \beta_i$ stopnje največ m
OUTPUT: umbralni polinom $q(\beta) = p(\beta)^2$

1. S tabelami razširimo prvih $m/2$ bitov.

$$p(\beta) = (\beta_m, \ldots, \beta_{m/2+1}, \boldsymbol{\beta_{m/2}}, \boldsymbol{\beta_{m/2-1}}, \ldots, \boldsymbol{\beta_0})$$
$$\downarrow$$
$$q'(\beta) = (\boldsymbol{\beta_{m/2}}, 0, \boldsymbol{\beta_{m/2-1}}, \ldots, \boldsymbol{\beta_1}, 0, \boldsymbol{\beta_0});$$

2. S tabelami razširimo in obrnemo zadnjih $m/2$ bitov.

$$p(\beta) = (\boldsymbol{\beta_m}, \ldots, \boldsymbol{\beta_{m/2+1}}, \beta_{m/2}, \beta_{m/2-1}, \ldots, \beta_0)$$
$$\downarrow$$
$$q(\beta) = (\beta_{m/2}, \boldsymbol{\beta_{m/2+1}}, \ldots, \boldsymbol{\beta_{m-1}}, \beta_1, \boldsymbol{\beta_m}, \beta_0);$$

3. Return $q(\beta)$;

5.3 Izboljšano množenje

V prejšnjem poglavju smo opisali algoritem 4 za množenje, ki je v osnovi enak algoritmu za množenje elementov v polinomskih bazah. Ozko grlo je predvsem končna redukcija umbralnega polinoma, saj je potrebno $3m$–terico bitov zreducirati na prvotno m–terico bitov. Kljub hitrejšemu algoritmu 8 za redukcijo bi se ji najraje izognili in tudi prostorsko omejili množenje le na $(m + 1)$ bitov.

Zopet izhajamo iz leme (4.2.). Oglejmo si točki (iii) in (iv) leme (4.2.) in predpostavimo, da pišemo umbralni polinom v obliki iz definicije (4.4.) s konstantnim členom na desni strani zapisa. Prva točka nam omogoči, da vrnemo skrajno desni bit ob premiku v desno nazaj na svoje mesto. Oglejmo si ta najbolj desni bit

$$(\beta_m, \ldots, \beta_1, \underline{\beta_0}) \quad >> \quad \text{(desni pomik)} \quad (\beta_{m-1}, \ldots, \beta_0, \underline{\beta_{-1}}).$$

Na tem mestu naj poudarim, da se enakost $\beta_0 = 0$ ne spoštuje v celoti. Za začetni β_0 upoštevamo, da je $\beta_0 = 0$ in s tem poskrbimo, da ne pokvarimo morebitne vrednosti ${\beta_{-1}}^2$, hkrati pa za prvotni β_1, ki se ob zamiku premakne na mesto β_0 ne upoštevamo zgornjega dejstva in v tem koraku elementa β_0 ne postavimo na 0. V nasprotnem

[2]Takšne zamike in aritmetiko si je najlažje predstavljati kot $3m$–terico bitov, ki jo navijemo na valj obsega m bitnih mest, tako da se posamezne m–terice prekrivajo.

ALGORITEM 8: ***Redukcija s tabelami***

INPUT: umbralni polinom $p(\beta)$ stopnje največ $3m$
OUTPUT: umbralni polinom $q(\beta)$ stopnje največ m

1. $q'(\beta)$ = (β_{m+1}, β_{m+2}, ..., β_{2m}) + (β_{-m}, β_{-m+1}, ..., β_{-1});
2. $q(\beta)$ = Reverse($q'(\beta)$) + (β_1, β_2, ..., β_m));
3. Return $q(\beta)$;

primeru bi nam namreč operacija postavila vse desne bite, ki bi se premaknili na mesto z indeksom 0 na vrednost 0.

Ob ponovitvi premika dobimo:

$$(\beta_{m-1}, \ldots, \beta_0, \beta_{-1}) \quad >> \quad (\beta_{m-2}, \ldots, \beta_{-1}, \beta_{-2}).$$

Točka (iii) leme (4.2.) nam pove, da je

$$(\beta_{m-2}, \ldots, \beta_{-1}, \beta_{-2}) = (\beta_{m-2}, \ldots, \beta'_2, \beta'_1),$$

kjer je

$$\beta'_1 = \beta_{-1} + \beta_1 \quad \text{in} \quad \beta'_2 = \beta_2 + \beta_0 = \beta_2.$$

Če naredimo še en korak, vidimo, da je

$$(\beta_{m-2}, \ldots, \beta_{-1}, \beta_{-2}) \quad >> \quad (\beta_{m-3}, \ldots, \beta_{-2}, \beta_{-3}).$$

In zopet

$$(\beta_{m-2}, \ldots, \beta_{-2}, \beta_{-3}) = (\beta_{m-2}, \ldots, \beta'_3, \beta'_2),$$

kjer sta

$$\beta'_3 = \beta_3 + \beta_0 \quad \text{in} \quad \beta'_2 = \beta_2 + \beta_{-2}.$$

Pomagamo si z dejstvom, da redukcija zrcalno preslika elemente z negativnimi indeksi na svoja prvotna mesta. Pravilo nam pove, da se skrajno desni elementi, ki dobijo ob pomiku v desno negativne indekse cilkično vračajo na svoja mesta oziroma v resnici se ob pomiku v desno pomikajo v levo. Vendar je potrebno te elemente hraniti v posebnem registru, saj sicer prepišejo svoje prvotne vrednosti in pokvarijo rezultat.

Desna stran že ima postopek za hkratno redukcijo. Poglejmo si sedaj še levo stran. Ob pomiku v levo se indeksi elementov umbralnega polinoma povečajo za 1:

$$(\underline{\beta_m}, \ldots, \beta_1, \beta_0) \quad << \quad \text{(levi pomik)} \quad (\underline{\beta_{m+1}}, \ldots, \beta_2, \beta_1)$$

Če na zgornjem primeru uporabimo točko (iv) leme (4.2.) vidimo, da je

$$(\beta_{m+1}, \ldots, \beta_2, \beta_1) = (\beta'_m, \ldots, \beta_2, \beta_1),$$

kjer je

$$\beta'_m = \beta_m + \beta_{m+1}.$$

Če naredimo še en zamik v levo:

$$(\beta_{m+1}, \ldots, \beta_2, \beta_1) \quad << \quad (\beta_{m+2}, \ldots, \beta_3, \beta_2)$$

in ponovimo redukcijo

$$(\beta_{m+2}, \beta_{m+1}, \ldots, \beta_4, \beta_3) = (\beta'_m, \beta'_{m-1}, \ldots, \beta_4, \beta_3),$$

kjer je

$$\beta'_m = \beta_m + \beta_{m+1} \quad \text{in} \quad \beta'_{m-1} = \beta_{m-1} + \beta_{m+2},$$

vidimo, da se tudi na levi strani skrajno levi biti ciklično ob pomiku v levo pomikajo v desno na svoja prvotna mesta. Naj ponovno poudarim, da opisana hkratna redukcija deluje le v primeru, da bite, ki na desni in na levi ob zamiku zlezejo iz prvotne m–terice hranimo v posebnih registrih in jih ne mešamo z originalnimi biti.

Zgoraj opisane zamike imenujemo **zamiki po modulu**.

Hitrejši algoritem za množenje je opisan v algoritmu 9. Funkcija `left_shift_mod(lb,rlb)` zamakne vektor `lb` v levo, skrajno levi bit iz vektorja `lb` pa prepiše kot skrajno desni bit vektorja `rlb`. Funkcija `right_shift_mod(rb,rrb)` zamakne vektor `rlb` v desno, skrajno desni bit iz vektorja `rb` pa prepiše kot skrajno levi bit vektorja `rlb`.

Oglejmo si potek algoritma v praksi.

Primer 5.2. *Oglejmo si obseg* $\mathbb{F}_{2^{11}}$. *Elemente označimo kot vektorje dolžine* 12 *s koordinatami iz množice* $\{0, 1\}$. *Izberimo elementa*

$$a = (001101011000) \quad in \quad b = (000101011110).$$

Zapisi v registrih ob izvajanju algoritma so prikazani v tabeli 5.2.

Produkt elementov je torej

$$a \cdot b = (111001000000).$$

ALGORITEM 9: ***Množenje s hkratno redukcijo***

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 = 0; lb = b; rb = b; rlb = 0; rrb = 0;
2. for i from 1 to m do
   2.1. lb = left_shift_mod(lb, rlb);
   2.2. rb = right_shift_mod(rb, rrb);
   2.3. if a_i == 1 then $c = c + lb + rb + rlb + rrb$;
3. Return $c(\beta)$;

Preverimo, ali rezultat drži:

$$\begin{aligned}
a(\beta) &= \beta_9 + \beta_8 + \beta_6 + \beta_4 + \beta_3 \\
b(\beta) &= \beta_8 + \beta_6 + \beta_4 + \beta_3 + \beta_2 + \beta_1 \\
a(\beta) \cdot b(\beta) &= (\beta_9 + \beta_8 + \beta_6 + \beta_4 + \beta_3) \cdot (\beta_8 + \beta_6 + \beta_4 + \beta_3 + \beta_2 + \beta_1) = \\
&= \beta_{17} + \beta_1 + \beta_{15} + \beta_3 + \beta_13 + \beta_5 + \beta_{12} + \beta_6 + \beta_{11} + \beta_7 + \beta_{10} + \beta_8 + \\
&+ \beta_{16} + \beta_{14} + \beta_2 + \beta_{12} + \beta_4 + \beta_{11} + \beta_5 + \beta_{10} + \beta_6 + \beta_9 + \beta_7 + \\
&+ \beta_{14} + \beta_2 + \beta_{12} + \beta_{10} + \beta_2 + \beta_8 + \beta_7 + \beta_1 + \beta_6 + \beta_2 + \beta_5 + \beta_3 + \\
&+ \beta_{12} + \beta_4 + \beta_{10} + \beta_2 + \beta_8 + \beta_7 + \beta_1 + \beta_6 + \beta_2 + \beta_5 + \beta_3 + \\
&+ \beta_{11} + \beta_5 + \beta_9 + \beta_3 + \beta_7 + \beta_1 + \beta_6 + \beta_5 + \beta_1 + \beta_4 + \beta_2 = \\
&= \beta_{17} + \beta_{16} + \beta_{15} + \beta_{13} + \beta_{11} + \beta_9 + \beta_8 + \beta_7 = \\
&= \beta_{11} + \beta_{10} + \beta_9 + \beta_6 = \\
&= (111001000000).
\end{aligned}$$

5.4 Izboljšan algoritem za iskanje inverza

V prejšnjem poglavju smo opisali najpreprostejši algoritem za iskanje inverza v končnem obsegu – algoritem z grobo silo, ki je inverze poiskal tako, da je element obsega dvignil na $(2^m - 2)$–go potenco. Za izvedbo je bilo potrebno narediti $(m-2)$ množenj in $(m-1)$ kvadriranj. Oglejmo si sedaj algoritem, ki zreducira število množenj na $\log_2(m-1)$.

Začnimo pri osnovnem algoritmu za iskanje inverza z grobo silo. Izhajamo iz dejstva,

Tabela 5.2: Registri pri množenju s hkratno redukcijo

(i)	bit i	lb	rm	rlb	rrb	c
1	0	001010111100	000010101111	000000000000	000000000000	
2	0	010101111000	000001010111	000000000000	000000000010	
3	1	101011110000	000000101011	000000000000	000000000110	101011011101
4	1	010111100000	000000010101	100000000000	000000001110	011100100110
5	0	101111000000	000000001010	010000000000	000000011110	
6	1	011110000000	000000000101	101000000000	000000111100	101010011111
7	0	111100000000	000000000010	010100000000	000001111010	
8	1	111000000000	000000000001	101010000000	000011110100	111011101010
9	1	110000000000	000000000000	110101000000	000111101010	111001000000
10	0	100000000000	000000000000	111010100000	001111010100	
11	0	000000000000	000000000000	111101010000	011110101000	

da za element obsega $\alpha \in \mathbb{F}_{2^m}$ velja

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

Sedaj pa glede na sodost ali lihost m ločimo dva primera:

1. m je **sodo** število. Potem velja

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

   oziroma

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

   Torej, če poznamo vrednost $\alpha^{2^{(m-2)/2}-1}$ potrebujemo za izračun $\alpha^{2^{m-1}-1}$ dve množenji in $m/2 + 1$ kvadriranje.

2. m je **liho** število. Potem velja

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

   oziroma

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

   Če poznamo $\alpha^{2^{(m-1)/2}-1}$ potrebujemo za izračun $\alpha^{2^{m-1}-1}$ $(m-1)/2$ kvadriranj in eno množenje.

Algoritem je zapisan v algoritmu 10. Definirajmo funkcijo `a%b` kot ostanek pri deljenju `a` z `b`.

ALGORITEM 10: ***Hitrejši inverz z grobo silo***

INPUT: umbralni polinom $a(\beta)$
OUTPUT: umbralni polinom $b(\beta) = a(\beta)^{-1}$

```
1. while m > 1 do
  1.1. if m % 2
    1.1.1 m = (m − 1)/2;
    1.1.2 a = a^(2^m+1);
  1.2. else
    1.2.1 m = (m − 2)/2;
    1.2.2 a = a^(2(2^m+1)+1);
2. Return a^2;
```

Primer 5.3. *Vzemimo obseg* $\mathbb{F}_{2^{11}}$. *Inverzni element je podan kot* $x^{-1} = x^{2^{11}-2} = x^{2046}$. *Če razcepimo število* $2^{11} - 2$ *dobimo*

$$\begin{aligned} 2^{11} - 2 &= 2(2^5 - 1)(2^5 + 1) \\ 2^5 - 1 &= 2(2^2 - 1)(2^2 + 1) + 1 \\ 2^2 - 1 &= (2 - 1)(2 + 1) \end{aligned}$$

Izračunamo ga torej po naslednjem postopku:

1	$(x)^2 = x^2$	1 *kvadriranje*
2	$x^2 x = x^3$	1 *množenje*
3	$(x^3)^{2^2} = x^{12}$	2 *kvadranji*
4	$x^{12} x^3 = x^{15}$	1 *množenje*
5	$(x^{15})^{2^4} = x^{240}$	4 *kvadriranja*
6	$x^{240} x^{15} = x^{255}$	1 *množenje*
7	$(x^{255})^{2^2} = x^{1020}$	2 *kvadriranji*
8	$x^{1020} x^3 = x^{1023}$	1 *množenje*
9	$(x^{1023})^2 = x^{2046}$	1 *kvadriranje*

Vidimo, da smo v primeru potrebovali 4 *množenja in* 10 *kvadriranj, kar je precej hitreje kot v primeru 4.6., kjer smo potrebovali* 10 *kvadriranj in* 9 *množenj.*

Oglejmo si še dejanski primer iz implementacije (isti vhodni podatki kot pri osnovnem algoritmu 5).

Primer 5.4. *Vzemimo obseg* $\mathbb{F}_{2^{11}}$ *in element* $a = (010000101111)$. *Algoritem poteka na naslednji način:*

stopnja	*element*
1	010000101111
2	010000101111
3	000101101101
12	000101101101
15	010111101110
240	010111101110
255	100101111011
1020	100101111011
1023	111100001100
2046	**000011111010**
2047	**111111111111**

Rezultat je enak kot v primeru 4.6. v prejšnjem poglavju.

5.5 Razširjen binarni Evklidov algoritem

Za hitrejše računanje inverza v Chebyshevih bazah bomo uporabili binarni razširjeni Evklidov algoritem. Pridevnik *binarni* vstavimo v naslov ker računamo inverz v obsegu $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$. Nastopajo torej samo dve binarni vrednosti, 0 in 1.

Več o Evklidovem algoritmu si lahko bralec prebere v diplomskem delu Matjaža Urlepa.

Tudi pri računanju inverza lahko potegnemo očitne vzporednice s polinomskimi bazami. Za izvedbo Evklidovega algoritma potrebujemo naslednje:

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

Za implementacijo oziroma za ugoditev zgornjih treh zahtev bomo uporabili trditev (4.5.). V primeru, da je umbralna stopnja nekega polinoma $p(\beta) \in \mathbb{F}_{2^m}$ večja od m, jo s pomočjo nerazcepnega polinoma $f(\beta) = \beta^m + \beta^{m-1} + \cdots + \beta + 1$ zreduciramo na stopnjo največ m. Nato lahko izvedemo običajen razširjeni Evklidov algoritem, le s pravilom za množenje v umbralnih bazah. Ko dobimo rezultat, preverimo, ali vsebuje konstantni člen. Če je različen od nič, je enak enoti za množenje in ga lahko po lastnosti (iii) trditve (4.5.) nadomestimo z vsoto $\sum_{i=1}^{m} \beta_i$. Rezultat je sedaj polinom z umbralno stopnjo največ m in brez konstantnega člena, kar je željen rezultat.

Poglejmo si primer izračuna inverza z razširjenim Evklidovim algoritmom.

ALGORITEM 11: ***Razširjen binarni Evklidov algoritem***

INPUT: umbralni polinom $a(\beta) \in \mathbb{F}_{2^m}$
OUTPUT: umbralni polinom $b(\beta) = a(\beta)^{-1}$

---

1. Redukcija polinoma $a(\beta)$ s polinomom $1 + \sum_{i=1}^{m} \beta_i$;

2. Razširjeni Evklidov algoritem

   2.1. $b = 1$; $c = 0$; $u = a$; $v = f$;

   2.2. while deg(u) != 0 do

      2.2.1 j = deg(u) - deg(v);

      2.2.2 if $j < 0$ then;

            $u \leftrightarrow v$; $b \leftrightarrow c$; $j = -j$;

      2.2.3 $u = u + x^j\ v$; $b = b - x^j\ c$;

   2.3. Return $b(\beta)$;

3. if $b_0 == 0$ then $b(\beta) = r(\beta)$;

   polinom $b(\beta)$ nadomestimo s polinomom $r(\beta)$ iz točke (ii) trditve 4.5.

4. Return $b(\beta)$;

Primer 5.5. *Naj bo $m = 11$. Poiščimo inverz elementa*

$$\alpha(\beta) = \beta_7 + \beta_4 + \beta_3 + 1.$$

Definirajmo $r(\beta)$ in $s(\beta)$ kot pomožna umbralna polinoma, ki nam bosta v pomoč pri izvajanju Evklidovega algoritma.

Sedaj lahko začnemo Evklidov algoritem z modificiranim pravilom za množenje $\beta_i\beta_j = \beta_{i+j} + \beta_{|i-j|}$

$$\begin{aligned} r_{-2} &= \beta_{11} + \cdots + \beta_1 + 1 = \\ &= \underline{(\beta_4 + 1)}(\beta_7 + \beta_4 + \beta_3 + 1) + \beta_{10} + \beta_9 + \beta_7 + \beta_6 + \beta_5 + \beta_4 + \beta_3 + \beta_2 \\ &= s_0 r_{-1} + r_0 \end{aligned}$$

$$r_{-1} = \beta_9 + \beta_8 + \beta_5 + \beta_4 + \beta_1 + 1 =$$

$$\begin{aligned} &= \underline{\beta_2}(\beta_7+\beta_6+\beta_3+\beta_2+\beta_1)+\beta_5+\beta_4+\beta 3+\beta_1+1= \\ &= s_1r_0+r_1 \end{aligned}$$

$$\begin{aligned} r_0 &= \beta_7+\beta_6+\beta_3+\beta_2+\beta_1= \\ &= \underline{(\beta_2+1)}(\beta_5+\beta_4+\beta_3+\beta_1+1)+\beta_4+\beta_2+1= \\ &= s_2r_1+r_2 \end{aligned}$$

$$\begin{aligned} r_1 &= \beta_5+\beta_4+\beta_3+\beta_1+1= \\ &= \underline{(\beta_1+1)}(\beta_4+\beta_2+1)+\beta_3+\beta_2+1= \\ &= s_3r_2+r_3 \end{aligned}$$

$$\begin{aligned} r_2 &= \beta_4+\beta_2+1= \\ &= \underline{(\beta_1+1)}(\beta_3+\beta_2+1)+1= \\ &= s_4r_3+r_2 \end{aligned}$$

Ob tem izvajamo tudi razširjeni del algoritma

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

Dobljeni rezultat je

$$b_4(\beta)=\beta_8+\beta_5+\beta_2+1=\alpha^{-1}(\beta)$$

Oziroma zaradi konstantnega člena

$$b_4(\beta)=\beta_{11}+\beta_{10}+\beta_9+\beta_7+\beta_6+\beta_4+\beta_3+\beta_1=\alpha^{-1}(\beta)$$

Za preizkus moramo zmnožiti $\alpha(\beta)\cdot\alpha(\beta)^{-1}$. Upoštevamo pravila za množenje v umbralnih bazah in lemo (4.2.), po kateri velja

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

Sledi

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

kot smo želeli.

Primer 5.6. *Vzemimo za vhodni podatek umbralni polinom* $a(\beta) = \beta_{10} + \beta_6 + \beta_3$. *Oglejmo si, kako poteka algoritem. Prvi del koraka je evklidov algoritem, drugi del pa je razširjeni del algoritma. Posamezni polinomi znotraj algoritma so zapisan v vektorski obliki, saj so preneseni direktno iz implementiranega algoritma.*

Korak Evklidovega algoritma

$$111111111111 = 000000000010x010001001000 + 010101001011$$

Korak razširjenega dela

$$000000000010 = 000000000010 \cdot 000000000001 + 000000000000$$

$$010001001000 = 000000000010 \cdot 010101001011 + 000100000011$$
$$000000000010 = 000000000010 \cdot 000000000001 + 000000000011$$

$$010101001011 = 000000000100 \cdot 000100000011 + 000000000110$$
$$000000001111 = 000000000100 \cdot 000000000011 + 000000000010$$

$$000100000011 = 000001000000 \cdot 000000000110 + 000010110011$$
$$001111111011 = 000001000000 \cdot 000000001111 + 000000000011$$

$$000010110011 = 000000100000 \cdot 000000000110 + 000001101011$$
$$001000000111 = 000000100000 \cdot 000000001111 + 001111111011$$

$$000001101011 = 000000010000 \cdot 000000000110 + 000000000111$$
$$001011111001 = 000000010000 \cdot 000000001111 + 001000000111$$

$$000000000110 = 000000000010 \cdot 000000000111 + 000000000001$$
$$001011111001 = 000000000010 \cdot 000000001111 + 001011110110$$

$$000000000111 = 000000000100 \cdot 000000000001 + 000000000010$$
$$101101101001 = 000000000100 \cdot 001011110110 + 001011111001$$

Rezultat ki ga dobimo je 001011110110 *oziroma umbralni polinom* $b(\beta) = \beta_9 + \beta_7 + \beta_6 + \beta_5 + \beta_4 + \beta_2 + \beta_1$. *Preverimo, ali je produkt dobljenega elementa z inverzom multiplikativna enota:*

$$\begin{aligned}
a(\beta) \cdot \beta(\beta) &= (\beta_{10} + \beta_6 + \beta_3) \cdot (\beta_9 + \beta_7 + \beta_6 + \beta_5 + \beta_4 + \beta_2 + \beta_1) \\
&= \beta_{19} + \beta_1 + \beta_{17} + \beta_3 + \beta_{16} + \beta_4 + \beta_{15} + \beta_5 + \beta_{14} + \beta_6 + \beta_{12} + \beta_8 + \beta_{11} + \beta_9 \\
&+ \beta_{15} + \beta_3 + \beta_{13} + \beta_1 + \beta_{12} + \beta_{11} + \beta_1 + \beta_{10} + \beta_2 + \beta_8 + \beta_4 + \beta_7 + \beta_5 \\
&+ \beta_{12} + \beta_6 + \beta_{10} + \beta_4 + \beta_9 + \beta_3 + \beta_8 + \beta_2 + \beta_7 + \beta_1 + \beta_5 + \beta_1 + \beta_4 + \beta_2 \\
&= \beta_{19} + \beta_{17} + \beta_{16} + \beta_{14} + \beta_{13} + \beta_9 + \beta_8 + \beta_5 + \beta_3 + \beta_2 + \beta_1 \\
&= \beta_4 + \beta_6 + \beta_7 + \beta_{10} + \beta_{11} + \beta_9 + \beta_8 + \beta_5 + \beta_3 + \beta_2 + \beta_1 \\
&= \beta_{11} + \beta_{10} + \beta_9 + \beta_8 + \beta_7 + \beta_6 + \beta_5 + \beta_4 + \beta_3 + \beta_2 + \beta_1.
\end{aligned}$$

S tem smo zaključili primer.

Poglavje 6

Analiza algoritmov

V poglavjih 4 in 5 smo opisali najprej osnovne različice algoritmov, nato pa njihove možne izboljšave. Diplomska naloga je osredotočena predvsem na izboljšave z vidika časovne zahtevnosti algoritmov. Prostorsko zahtevnost algoritmov sem se odločil izpustiti, saj je bila implementacija prirejena običajnemu prenosnemu računalniku z 32 bitnim procesorjem in namiznemu računalniku z 64 bitnim procesorjem, kjer prostorska zahtevnost ne igra velike vloge.

Algoritme bom analiziral in razbil na osnovne besedne in vektorske operacije. Tako bom dobil za vsak algoritem nekakšno objektivno merilo za primerjavo z ostalimi algoritmi. Podal bom tudi praktična dejstva in ugotovitve iz implementacije.

Poglavje vsebuje analizo vsakega opisanega algoritma iz poglavij 4 in (5), za vsak algoritem iz aritmetike (kvadriranje, redukcija, ...) pa je podana tudi tabela izmerjenih časovnih zahtevnosti. Na začetku poglavja so opisane osnovne besedne in vektorske operacije ter nekatere druge osnovne informacije.

6.1 Uvod

Naj kot uvod v poglavje podam oris arhitekture, ki stoji pod implementiranimi algoritmi. V tem poglavju se ne bom spuščal v kodo samo, vendar menim da je potrebno opisati osnovne gradnike implementacije. Tako bom v nadaljevanju opisal osnovne uporabljene tipe besed, definiral vektorje ter nekatere simbole, ki jih bom uporabljal tekom celotnega poglavja.

Hardware

Implementacija je bila narejena za 64–bitni operacijski sistem, da sem lahko preveril učinkovitost te trenutno zelo popularne aritmetike. Sistem oziroma računalnik, nakaterem so bile izvedene vse meritve je opisan v tabeli 6.1.

Tabela 6.1: Specifikacije sistema

Procesor	AMD Athlon64 3000
Spomin	1 GB
Operacijski sistem	Ubuntu 5.10
Linux jedro	2.6.12
Prevajalnik	gcc 4.0.1
Namizje	Gnome
Programski orodje	KDevelop 3.2.0
Optimizacije prevajanja	`-lm -O0 -g3 -m64 -march=athlon64 -std=c99 -mno-align-double -O3 -ffast-math -funroll-all-loops -fpeel-loops -ftracer -funswitch-loops -ftree-vectorize -Wall`

Tipi besed

Implementacijo sem postavil na štiri tipe besed. Razlikujejo se po številu bitov, ki jih uporabljajo za predstavitev v računalniškem spominu. Dolžino posameznega tipa besede bom označil s simbolom $\boldsymbol{w}$. Tipi besed so opisani v tabeli 6.2. Vsi tipi so nepredznačeni (`unsigned` - beseda ne vsebuje podatkov o predznaku števila, le ustrezne bite), saj bi imeli sicer v nasprotnem primeru težave z osnovno bitno aritmetiko.

Tabela 6.2: Tipi besed

tip	w	primer
`unsigned char`	8	`00100101`
`unsigned short int`	16	`0100010100101011`
`unsigned int`	32	`01010110101000101000001010110101`
`unsigned long`	64	`0100101001010011101010100010011 0-` `0111001101001001001001101111010 1`

Besedne operacije

Sedaj, ko poznamo osnovne tipe besed, je potrebno opisati osnovne operacije na omenjenih tipih. Najpogostejše operacije so *seštevanje* in *odštevanje* (kar je v obsegih $\mathbb{F}_{2^m}$ ki ležijo nad obsegi $\mathbb{F}_2$ enaka operacija) ter *prirejanje* vrednosti. Sledijo jim še pomik besede v levo ali desno stran *SHIFT* ter logična relacija *IN* v dveh različicah: *bitni* in *besedni*. V implementaciji se uporabljajo tudi še nekatere druge

operacije (npr. logični *ALI*) vendar sem se osredotočil na zgoraj naštete. Primeri operacij so opisani v tabeli 6.3 na tipu besed `unsigned char`.

Tabela 6.3: Osnovne besedne operacije

operacija	simbol	primer
prirejanje	`=`	`01110110 = 01110110`
seštevanje	`+`, `XOR`, `^`	`01010101 = 11001011 + 10011110`
pomik	<<, >>	`00110101 << 4 = 01010000`
bitni IN	`&`	`01000101 & 00000111 = 00000101`
besedni IN	`&&`	`10101110 && 10010010 = 00000001`

Vsaka zgoraj omenjena operacija potrebuje določeno količino časa, da se izvede. Formalno vsaki operaciji pripišemo neko vrednost(v odvisnosti od časa trajanja) in tako potem izpeljali časovno trajanje celotnega algoritma. Operacije ločimo kljub dejstvu, da se v praksi izkaže, (tabela 6.4) da se operacije ne razlikujejo drastično po časovni zahtevosti. Trajanje posamezne operacije označimo s simbolom t_{op}, npr. t_+.

Vektorska predstavitev

Naj najprej odgovorim na vprašanje, kako so elementi obsegov $\mathbb{F}_{2^m}$ nad $\mathbb{F}_2$ predstavljeni znotraj računalniške arhitekture. Število bitov, potrebnih za predstavitev elementa iz obsega $\mathbb{F}_{2^m}$ bom označil s simbolom $\boldsymbol{m}$. Elemente obsega predstavimo z **vektorji**. Vektorji so zaporedno nanizane besede. Dolžino ustreznega vektorja označimo s simbolom $\boldsymbol{\ell}$ in jo izračunamo po formuli

$$\ell = \lceil m/w \rceil.$$

Kot zanimivost naj opišem, kako se zgornja formula prevede v računalniški zapis. Ker se želimo izogniti klicanje dodatnih funkcij npr. `ceil()` shranimo vrednosti

Tabela 6.4: Trajanje posamezne besedne operacije

operacija	trajanje (μs)
`=`	0.0242
`+`	0.0243
<<, >>	0.0246
`&`	0.0246
`&&`	0.0257

spremenljivk m in w v spremenljivke tipa `int`, torej celi števili. Tako nam izračun

$$\ell = m/w = \lfloor m/w \rfloor$$

pravzaprav vrne spodnjo mejo. Zato je potrebno vrednosti ℓ prišteti 1. Vendar nam formula v primerih, ko je m večkratnik w (kar se lahko zgodi zaradi definicije (4.4.)) vrne napačen rezultat: dolžina vektorja je za eno mesto prevelika. Tako je potrebno formulo rahlo modificirati v:

```
l = m / w + ((m % w) ?  1 :  0)
```
[1],

oziroma

$$\ell = \begin{cases} m/w + 1 & \text{ostanek}(m/w) \neq 0, \\ m/w & \text{sicer.} \end{cases}$$

V računalniškem jeziku C (ali $C{+}{+}$) jih označimo npr. `unsigned char vektor[3]`. Oznaka pomeni, da zaporedoma nanizamo tri besede tipa `unsigned char`. Nekaj primerov je opisanih v tabeli 6.5.

Tabela 6.5: Primeri vektorske predstavitve

obseg	tip besede	ℓ	primer
$\mathbb{F}_{2^{41}}$	`unsigned char`	6	`00000001 00100101 01101010 10110100 01010010 01110101`
$\mathbb{F}_{2^{41}}$	`unsigned short`	3	`0000000100100101 0110101010110100 0101001001110101`
$\mathbb{F}_{2^{41}}$	`unsigned int`	2	`00000000000000000000000100100101 01101010101101000101001001110101`
$\mathbb{F}_{2^{41}}$	`unsigned long`	1	`00000000000000000000000100100101- 01101010101101000101001001110101`

Vektorske operacije

Sedaj lahko posplošimo besedne operacije na vektorske operacije. Posamezno besedno operacijo izvedemo na vsaki besedi vektorja posebej. Operacije prirejanja, seštevanja in logičnega IN so trivialne. Težava se pojavi pri pomiku posameznih besed vektorja v levo ali desno. Pri operaciji pomika je potrebno paziti na skrajno desni bit ob pomiku v levo oziroma na skrajno levi bit ob pomiku v desno. Primera pomika v levo in desno sta opisana v tabeli 6.6.

[1] funkcija `( ?  :  )` se uporablja na sledeči način:

```
pogoj ?  a :  b
```

funkcija vrne vrednost `a`, če je `pogoj` resničen(`1,true`) oziroma vrednost `b` če je `pogoj` neresničen(`0,false`)

Tabela 6.6: Primera pomika vektorja

INPUT	00101001 00100010 11010011
LEFT	01010010 01000101 10100110
LEFT	10100100 10001011 01001100
INPUT	01010010 00100101 10100010
RIGHT	00101001 00010010 11010001
RIGHT	00010100 10001001 01101000

Oglejmo si najprej pomik v desno. Vsako besedo vektorja je potrebno pomakniti v desno, hkrati pa je potrebno pogledati k levi sosednji besedi kakšno vrednost ima njen skrajno desni bit. V primeru, da je bit enak 1 ga je potrebno po zamiku besede prišteti na skrajno desni konec besede.

Podobno poteka postopek tudi pri pomiku v levo, le da v tem primeru opazujemo skrajno desne bite besed.

V obeh primerih si v naprej pripravimo skrajno desni bit in skrajno levi bit, tako da izvedemo prištevanje enega samega bita s samo eno operacijo(in se izognemo nepotrebnemu pomikanju enega samega bita). Spodaj sta primera takšnih bitov za tip `unsigned char`:

skrajno levi bit	10000000,
skrajno desni bit	00000001.

Sedaj povzemimo v tabeli 6.7 število besednih operacij potrebnih za posamezno(npr. en pomik) vektorsko operacijo. Vektor je dolžine ℓ, besede so dolžine w. Predpostavi se, da se prištevanje skrajno desnega oziroma skrajno levega bita pri pomikanju vedno izvede. Razčlenitev je na posamezne besedne operacije, definirane v tabeli 6.3.

Tabela 6.7: Trajanje vektorskih operacij

operacija	=	+	&	<<,>>	**skupaj**
=	l				l
+		l			l
&			l		l
<<,>>		$l-1$	$l-1$	l	$3l-2$

Maske

Ostane nam le še vprašanje iskanja posameznega bita znotraj besede. Ker smo spoznali operacijo pomika besede v levo oziroma desno bi lahko za iskanje i–tega bita znotraj besede uporabili naslednji postopek:

ALGORITEM 12: ***Iskanje i–tega bita s pomiki***

INPUT: Beseda `a`
OUTPUT: i–ti bit $\mathtt{a}_i$

---

1. a `=` a `<<` $(w-i-1)$`;`
2. a `=` a `>>` $(w-1)$`;`
3. `if(`a `!=` 0`) Return` 1`;`
   `else Return` 0`;`

Algoritem sicer vrne vrednost i–tega bita, vendar porabi $2w-i-2$ operacij, kar je preveč. Na tem mestu definiramo **maske**. Maske so predhodno naračunane tabele s katerimi si olajšamo iskanje posameznih bitov. V implementaciji sem uporabil dva tipa mask, ki sta opisana v tabeli 6.8.

Tabela 6.8: Tipa mask

right maske za $i+1$ desnih bitov	**single bit** maske za i–ti bita
`rmask[0] = 00...0001`	`sbitmask[0] = 00...0001`
`rmask[1] = 00...0011`	`sbitmask[1] = 00...0010`
⋮	⋮
`sbitmask[w-1] =` $\overbrace{11\ldots1111}^{\mathtt{w}}$	`sbitmask[w-1] =` $1\overbrace{00\ldots0000}^{\mathtt{w-1}}$

S pomočjo mask se algoritem 12 poenostavi v hitrejši algoritem 13.
S tem algoritmom najdemo i–ti bit le z eno besedno operacijo: logičnim *IN*.

6.2 Osnovni algoritmi

V tem razdelku bom podal analizo algoritmov aritmetike Chebyshevih baz. Za vsak algoritem bom preštel bitne operacije, potrebne za dokončanje postopka. Prav tako

ALGORITEM 13: ***Iskanje i–tega bita z maskami***

INPUT: Beseda `a`
OUTPUT: i–ti bit $\mathtt{a}_i$

```
1. if (a & sbitmask[i]) Return 1;
   else Return 0;
```

bom za vsak implementiran algoritem tudi podal izmerjene rezultate.

Algoritme bom analiziral na naslednji način: za vsak korak algoritma preštejem potrebne operacije in jih nato na koncu strnem v celovito sliko o hitrosti in učinkovitosti algoritma. Dolžino vektorja označimo z ℓ, dolžino posamezne besede pa z w. Dolžina bitne predstavitve je označena z m. Oznake za trajanje posameznih operacij so definirane v vsakem razdelku posebej. Trajanja osnovnih vektorski operacij so s simboli označena v tabeli 6.9.

Tabela 6.9: Pojasnilo uporabljenih simbolov

simbol	opis
t_+	trajanje vektorske operacije `+`
$t_=$	trajanje vektorske operacije `=`
$t_\&$	trajanje vektorske operacije `&`
$t_{<<}$	trajanje vektorskega pomika v levo ali desno

6.2.1 Prehod med bazama

Analizo bom začel z algoritmom 1, ki opisuje prehod med optimalno normalno bazo tipa II in Chebyshevo bazo.

1. `for` zanka z indeksom `i` se izvede m–krat

   1.1. Na vsakem koraku zanke izvedemo eno prirejanje s pomočjo mask.

V celoti potrebujemo za prehod med bazama natanko m prirejanj. Čas, ki ga potrebujemo za izvedbo algoritma je torej

$$t_{switch} = m \cdot t_{op} \tag{6.1}$$

oziroma

$$t_{switch} = m \cdot t_= \tag{6.2}$$

6.2.2 Kvadriranje

Osnovna različica kvadriranja v Chebyshevih bazah je opisana v algoritmu 2. Trajanje pogojnih `if` stavkov zanemarimo.

1. `for` zanka z indeksom `i` se izvede $(m+1)$–krat

   1.1. Prvi pogoj izvede seštevanje:

   ```
   q[2i/w] += (p[i/w] & sbitmask[i%w])?sbitmask[2i%w]:0.
   ```

   1.2. Drugi pogoj izvede seštevanje:

   ```
   q[(2m-i)/w] += (p[i/w] & sbitmask[i%w])?(sbitmask[(2m-i)%w]:0.
   ```

   1.3. Oba pogoja pokrijeta natanko vseh $(m+1)$ korakov. Vsako seštevanje vsebuje tudi dve operaciji bitnega *IN*.

Kvadriranje v svoji najpreprostejši obliki potrebuje za izvedbo čas

$$t_{s(b)} = 3(m+1) \cdot t_{op} \tag{6.3}$$

oziroma, v pedantnejši obliki:

$$t_{s(b)} = (m+1)(t_{+} + 2t_{\&}). \tag{6.4}$$

6.2.3 Redukcija

Redukcija je opisana v algoritmu 3. Zopet si oglejmo najpreprostejšo obliko algoritma.

1. `for` zanka z indeksom `i` se izvede (m)–krat.

   1.1. Na vsakem koraku zanke se izvede seštevanje:

   ```
   q[i / w] += ((p[i / w] & sbitmask[i % w]) +
   (p[j / w] & sbitmask[j % w]) +
   (p[k / w] & sbitmask[k % w])) ?  sbitmask[i % w] :  0.
   ```

   Vsak korak zanke torej vsebuje 3 seštevanja in 3 bitne *IN*.

Redukcija potrebuje torej

$$t_{r(b)} = 6m \cdot t_{op} \tag{6.5}$$

oziroma

$$t_{r(b)} = m(3t_{+} + 3t_{\&}). \tag{6.6}$$

6.2.4 Množenje

Sedaj, ko smo analizirali redukcijo se lahko osredotočimo na množenje. Osnovni algoritem je opisan v algoritmu 4. Oglejmo si, koliko vektorskih operacij potrebujemo za izvedbo množenja v Chebyshevih bazah.

1. Postavitev spremenljivk `lb` in `rb` na začetno vrednost zahteva dve prirejanji. Brez škode za splošnost ju lahko zanemarimo.
2. Zanka `for` se izvede m–krat.
   2.1. Pomik v levo vektorja `lb` zahteva en vektorski pomik v levo, pomik v desno spremenljivke `rb` pa en vektoski pomik v desno.
   2.2. V primeru, da je `a[i/w] & sbitmask[i%2]` $\neq$ `0` opravimo seštevanje `c += lb + rb`.

   Za grobo oceno lahko predpostavimo, da se seštevanje izvede vsak korak. Torej potrebujemo za vsak korak zanke dve seštevanji, dva pomika in en bitni *IN*.
3. Po zaključeni zanki izvedemo redukcijo.

Za uspešno množenje potrebujemo torej

$$t_{m(b)} = 5m \cdot t_{op} + t_{r(b)} = 5m \cdot t_{op} + 6m \cdot t_{op} = 11m \cdot t_{op} \tag{6.7}$$

oziroma

$$\begin{aligned} t_{m(b)} &= m(2t_{+} + 2t_{<<} + t_{\&}) + t_{r(b)} \\ &= m(2t_{+} + 2t_{<<} + t_{\&}) + m(3t_{+} + 3t_{\&}) \\ &= m(5t_{+} + 4t_{\&} + 2t_{<<}). \end{aligned} \tag{6.8}$$

6.2.5 Inverz

Za inverz z grobo silo lahko pričakujemo, da bo potreboval precej operacij. Oglejmo si algoritem 5.

1. Postavitev spremenljivke `y` na vrednost `a` zahteva eno prirejanje, kar zanemarimo.
2. Zanka `for` se izvede $(m-2)$–krat. Seveda s predpostavko, da je `m` $\geq$ `2`
   2.1. Vsak korak se izvede eno kvadriranje `z = y`^{2}.
   2.2. Vsak korak se izvede eno množenje `y = z` $\cdot$ `y`
3. Na koncu izvedemo eno kvadriranje.

V celoti potrebujemo na vsakem koraku zanke eno kvadriranje in eno množenje, na koncu pa še eno kvadriranje.

Algoritem tako zahteva

$$\begin{aligned} t_{i(bf)} &= (m-2)(t_{s(b)} + t_{m(b)}) + t_{s(b)} \\ &= (m-2)(3(m+1)\cdot t_{op} + 11m \cdot t_{op}) + (3(m+1)\cdot t_{op}) \\ &= (m-2)(14m+1)t_{op} + (3m+3)t_{op} \\ &= (14m^2 - 24m + 1)t_{op}. \end{aligned} \tag{6.9}$$

Pogledano še natančneje dobimo:

$$\begin{aligned} t_{i(bf)} &= (m-2)(t_{s(b)} + t_{m(b)}) + t_{s(b)} = (m-1)t_{s(b)} + (m-2)t_{m(b)} \\ &= (m-2)((m+1)(t_+ + 2t_\&) + m(5t_+ + 4t_\& + 2t_{<<})) + (m+1)(t_+ + 2t_\&) \\ &= (m-2)((6m+1)t_+ + (6m+2)t_\& + 2mt_{<<}) + (m+1)t_+ + 2(m+1)t_\& \\ &= (6m^2 - 11m - 2)t_+ + (6m^2 - 10m - 4)t_\& + 2m(m-2)t_{<<} + (m+1)t_+ + 2(m+1)t_\& \\ &= (6m^2 - 10m - 1)t_+ + (6m^2 - 8m - 2)t_\& + 2m(m-2)t_{<<}. \end{aligned} \tag{6.10}$$

6.2.6 Meritve

Rezultate analize osnovnih algoritmov lahko sedaj povzamemo v dveh tabelah. Tabeli 6.10 in 6.11 podata **natančno število vektorskih operacij** za dan algoritem glede na velikost obsega ob opisanih predpostavkah.

Tabeli 6.12 in 6.13 pa navedeta meritve trajanj(v μs) operacij za vse štiri omenjene tipe besed.

Tabela 6.10: Število vektorskih operacij za redukcijo, kvadriranje in množenje

	Kvadriranje				**Redukcija**				**Množenje**			
m	&	+	<<	$\sum$	&	+	<<	$\sum$	&	+	<<	$\sum$
11	24	12		36	33	33		66	44	55	22	121
23	48	24		72	69	69		138	92	115	46	253
41	84	42		126	123	123		246	164	205	82	451
89	180	90		270	267	267		534	356	445	178	979
113	228	114		342	339	339		678	452	565	226	1243
191	384	192		576	573	573		1146	764	955	382	2101
223	448	224		672	669	669		1338	892	1115	446	2453
281	564	282		846	843	843		1686	1124	1405	562	3091
359	720	360		1080	1077	1077		2154	1436	1795	718	3949

Tabela 6.11: Število vektorskih operacij za invertiranje

	Inverz	
m	št. kvadriranj	št. množenj
11	10	9
23	22	21
41	40	39
89	88	87
113	112	111
191	190	189
223	222	221
281	280	279
359	358	357

Tabela 6.12: Trajanje posameznih operacij v Chebyshevih bazah v μs

	Redukcija				**Kvadriranje**			
m	`char`	`short`	`int`	`long`	`char`	`short`	`int`	`long`
11	0.20	0.20	0.30	0.20	0.10	0.20	0.20	0.40
23	1.40	0.90	0.50	0.50	0.10	0.20	0.20	0.20
41	1.00	1.50	1.80	1.50	0.30	0.10	0.30	0.40
89	1.80	1.80	2.00	3.00	0.40	1.10	0.60	1.10
113	2.30	3.00	2.80	2.40	0.80	0.70	0.70	0.60
191	5.80	5.30	4.60	4.70	1.60	1.30	0.90	1.40
223	6.10	6.30	5.80	5.40	1.60	1.70	1.90	1.70
281	7.60	8.20	7.90	7.80	2.70	2.00	2.50	2.70
359	9.70	10.20	11.30	10.30	3.40	3.20	2.60	2.70

Tabela 6.13: Trajanje posameznih operacij v Chebyshevih bazah v μs

	Množenje				**Inverz**			
m	`char`	`short`	`int`	`long`	`char`	`short`	`int`	`long`
11	1.40	0.60	0.70	1.10	10.0	0.0	0.0	10.0
23	3.00	1.80	2.10	2.60	80.0	40.0	30.0	50.0
41	7.40	5.20	2.70	2.10	300.0	170.0	110.0	120.0
89	24.90	15.90	9.10	8.50	2310.0	1370.0	910.0	620.0
113	40.40	21.50	14.20	11.10	4720.0	2740.0	1490.0	1120.0
191	107.20	65.20	33.00	22.80	19880.0	10840.0	5630.0	3370.0
223	143.60	76.90	41.10	33.30	30980.0	16680.0	8480.0	6710.0
281	229.40	120.50	64.00	41.80	64540.0	33100.0	16540.0	11830.0
359	357.60	193.40	89.70	60.30	124420.0	66060.0	30630.0	20750.0

Če si ogledamo za primer izračun inverza za $m = 359$, tip besede `long`, dobimo $t_{i(bf)} = 357 \cdot 60.3 + 358 \cdot 2.7 = 21527.1 + 966.6 = 22538.7\ \mu s$. Izmerjena vrednost predstavlja $20750/22538.7 = 92.1\%$ izračunane vrednosti.

V primeru, ko je $m = 191$, tip `int` dobimo $t_{i(bf)} = 190 \cdot 0.9 + 189 \cdot 33 = 6408\ \mu s$. Izmerjena vrednost predstavlja tako $5630/6408 = 88\%$ teoretičnega algoritma.

Za odstopanja so krive predpostavke, da so znotraj zank pogoji preverjanja vedno izpolnjeni in se zato vedno izvajajo določene operacije(npr. seštevanje znotraj zanke množenja).

6.3 Izboljšani algoritmi

Za spodnjo mejo izboljšav sedaj postavimo v prejšnjem razdelku izmerjene čase in se posvetimo analizi izboljšanih algoritmov ter merjenju razsežnosti izboljšav.

Število bitov v reprezentaciji elementa Chebysheve baze zopet označimo z m, dolžino tipa besede z w, dolžino vektorja, v katerem hranimo element obsega pa označimo z ℓ.

6.3.1 Potenciranje elementa Chebysheve baze

Najprej si zgolj s stališča preštevanja operacij oglejmo algoritem 6 za potenciranje posameznega elementa baze.

1. Za izračun binarne reprezentacije elementa $e = \sum_{k=0}^{n-1} e_k 2^k$ potrebujemo največ n deljenj z 2, vendar deljenje s konstantami zanemarimo.

2. Zanka `for` se izvede n krat.

   2.1. Predpostavimo, da je pogoj izpolnjen na vsakem koraku. Tako se na vsakem koraku izvede eno prirejanje in eno množenje s konstanto, kar zopet zanemarimo.

Celoten algoritem tako zahteva

$$t_{pow} = n \cdot t_{op} \tag{6.11}$$

oziroma

$$t_{pow} = n \cdot t_{=}. \tag{6.12}$$

6.3.2 Izboljšano kvadriranje

Oglejmo si algoritem 7, ki za poskrbi za hitrejše kvadriranje s pomočjo predhodno izračunanih tabel.

1. V prvem koraku razširimo prvih $(m/2)$ bitov s pomočjo predhodno izračunanih tabel za podvajanje ter podvajanje in obračanje bitov. Predpostavimo, da je dolžina predhodno izračunanih tabel 8^2 bitov. $(m/2)$ bitov je shranjenih v $(m/2)/w$ besedah. Za vsako besedo pa bomo potrebovali $(w/8)$ tabel. Iz tega sledi, da v prvem koraku izvedemo

   $$(m/2)/w \cdot (w/8) = m/16$$

   prirejanj.

2. V drugem koraku ponovimo postopek iz prvega koraka, le da sedaj hkrati razširimo in obrnemo drugih $(m/2)$ bitov ter jih prištejemo bitom, izračunanim v prvem koraku. Tako se v drugem koraku izvede $(m/16)$ seštevanj.

Celoten algoritem za kvadriranje 7 tako za izvedbo potrebuje

$$t_{s(t)} = \lceil m/16 \rceil \cdot t_{op} \tag{6.13}$$

oziroma

$$t_{s(t)} = \lceil m/16 \rceil \cdot t_{+}. \tag{6.14}$$

[2]Mogoča je uporaba 16 bitnih tabel vendar je potrebno poizkati kompromis med hitrostjo in prostorskimi zahtevami

6.3.3 Izboljšana redukcija

V prejšnjem razdelku analizirani algoritem za redukcijo smo izboljšali z algoritmom 8, ki prav tako kot zgoraj opisani algoritem za kvadriranje uporablja predhodno izračunane tabele.

1. Ohranimo notacijo iz prvotnega opisa algoritma. V prvem koraku prištejemo bite

   $(\beta_{m+1}, \beta_{m+2}, \ldots, \beta_{2m})$ k bitom $(\beta_{-m}, \beta_{-m+1}, \ldots, \beta_{-1})$.

   Seštevanje izvajamo z besedami, tako da porabimo za seštevanje zgornjih bitov (m/w) besednih seštevanj.

2. V naslednjem koraku nato bitom $(\beta_1, \beta_2, \ldots, \beta_m)$ prištejemo obrat zgoraj dobljenega rezultata. Potrebujemo torej zopet (m/w) seštevanj in $(m/8)$ prirejanj. Seveda moramo upoštevati tudi operacije, ki se dogajajo znotraj vsake besede, torej za vsako omenjeno seštevanje še $(w/8)$ besednih operacij oziroma kar vektorskih operacij.

V celoti porabi analiziran algoritem

$$t_{r(t)} = \lceil m/w \rceil \cdot t_{op} + \lceil m/w \rceil \cdot t_{op} = 2\lceil m/w \rceil \cdot t_{op} \tag{6.15}$$

oziroma

$$t_{r(t)} = 2\lceil m/w \rceil \cdot t_{+}. \tag{6.16}$$

besednih operacij. Če upoštevamo, da je potrebno vsaki besedi prišteti še $(w/8)$ tabel, dobimo rezultat

$$t_{r(t)} = \lceil m/w \rceil \cdot t_{op} + \lceil m/w \rceil (w/8) \cdot t_{op} = (2 + (w/8))\lceil m/w \rceil \cdot t_{op} \tag{6.17}$$

oziroma

$$t_{r(t)} = (2 + (w/8))\lceil m/w \rceil \cdot t_{+}. \tag{6.18}$$

Modificirani rezultati so podani v tabeli 6.16 v oklepajih.

6.3.4 Izboljšano množenje

V algoritmu 9 smo kljub dejstvu, da smo precej pohitrili redukcijo opisali množenje, ki redukciji izvaja kar med samim izvajanjem. Oglejmo si analizo algoritma.

1. Začetne predpise vrednosti bomo zanemarili.

2. Zanka `for` z indeksom `i` se izvede m–krat

2.1. Ker hranimo podatke v štirih vektorjih(dva ki predstavljata pomike v levo in desno ter dva, ki predstavljata pomike v obratni smeri oziroma bite, ki so padli pod indeks 0 oziroma nad indeks m) se zato na vsakem koraku izvedejo 4 vektorski pomiki: dva leva in dva desna. Poleg tega se ustrezna bita preneseta iz prednjih registrov v zadnja dva. Zato potrebujemo še dodatni 2 besedni seštevanji, ki jih bom zanemaril.

2.2. Predpostavimo, da seštevanj vseh štirih registrov

```
c = c + lb + rb + rlb + rrb
```

izvedemo na vsakem koraku. Za izvedbo le–tega potrebujemo še štiti vektorska seštevanja.

Končna vsota operacij je sledeča:

$$t_{m(m)} = m(4t_{op} + 4t_{op}) = 8m \cdot t_{op} \tag{6.19}$$

oziroma

$$t_{m(m)} = m(4t_{<<} + 4t_{+}). \tag{6.20}$$

6.3.5 Izboljšano iskanje inverza

Algoritem 10 bistveno pohitri algoritem za iskanje inverza z grobo silo opisan v algoritmu 5. Zaradi različnih razčlenitev različnih vrednosti m bom število potrebnih operacij podal v tabeli 6.14.

Tabela 6.14: Število operacij za izboljšano iskanje inverza

m	št. kvadriranj	št. množenj
11	9	4
23	21	6
41	39	6
89	87	8
113	112	8
191	179	12
223	221	12
281	279	10
359	357	12

6.3.6 Razširjeni Evklidov algoritem

Oglejmo si še zadnji opisan algoritem v poglavju 5, algoritem 11 oziroma razširjeni binarni Evklidov algoritem.

- Redukcijo polinoma v praksi izpustimo, saj ne računamo s polinomi, katerih stopnja bi bila večja od m.
- Prirejanja spremenljivkam na začetku algoritma zanemarimo.
- `while` zanka se izvaja dokler je stopnja vhodnega podatka, umbralnega polinoma `u` večja od 0. Predpostavimo, da je začetna stopnja polinoma `u` enaka m in da na vsakem koraku zmanjšamo njegovo stopnjo za 1.
    1. Izračun vrednosti spremenljivke `j` kot razliko stopenj umbralnih polinomo `u` in `v` zanemarimo.
    2. Izmenjavo vrednosti spremenljivk `u` in `v` v primeru, da je `j`< 0 zanemarimo.
    3. Na vsakem koraku tako izvedemo dve vektorski seštevanji in dve množenji v Chebyshevih bazah. Uporabljamo rahlo modificirano množenje(saj gre za množenje s konstanto oziroma le z umbralnim polinomom b, ki ima le en neničelen element β_i) ki se v praksi pokaže za 3 do 5–krat(oziroma za faktor f) hitrejše kot običajno množenje. Izognemo se preverjanju vsakega bita posebej in izvedemo le ustrezna zasuka v levo in desno s hkratno redukcijo za ustrezih i mest. Trajanje slednjega označimo s simbolom

$$t_{m(e)} = t_{m(t)}/f.$$

Pri seštevanju operacij bomo zanemarili vektorski seštevanji iz koraka 3, saj sta v primerjavi z množenjem v Chebyshevih bazah zanemarljivo kratki. Za uspešno izveden razširjen Evklidov algoritem potrebujemo v **najslabšem** primeru

$$t_{i(eu)} = 2m \cdot t_{m(t)}/f, \tag{6.21}$$

kjer je f faktor, za katerega je modificirano množenje hitrejše od klasičnega množenja oziroma

$$t_{i(eu)} = 2m \cdot t_{m(e)}. \tag{6.22}$$

6.3.7 Meritve

Kot v prejšnjem poglavju bom v tabelah 6.16 in 6.15 opisal število vektorskih operacij za izboljšane algoritme kvadriranja, redukcije in množenja. Sledile bodo metritve porabljenih časov za posamezne izboljšane algoritme taiste algoritme redukcije, kvadriranja, množenja in invertiranja v tabelah 6.17, 6.18 in 6.19.

Vsi podatki so v $\mu s = 10^{-6}s$.
Iz rezultatov lahko izluščimo dve ugotovitvi, ki sta plod preizkusa algoritmov v praksi.

Tabela 6.15: Število vektorskih seštevan za izboljšano redukcijo glede na tip besede

	Redukcija			
m	`char`	`short`	`int`	`long`
11	4(8)	2(3)	2(3)	2(3)
23	6(15)	4(8)	2(3)	2(3)
41	12(48)	6(15)	4(8)	2(3)
89	24(168)	12(48)	6(15)	4(8)
113	30(255)	16(80)	8(24)	4(8)
191	48(624)	24(168)	12(48)	6(15)
223	56(840)	28(224)	14(63)	8(24)
281	72(1368)	36(360)	18(99)	10(35)
359	90(2115)	46(575)	24(168)	12(48)

Tabela 6.16: Število vektorskih operacij za izboljšano kvadriranje in množenje

	Kvadriranje				**Modificirano množenje**			
m	`&`	`+`	$<<$	$\sum$	`&`	`+`	$<<$	$\sum$
11		1		1		44	44	88
23		2		2		92	92	184
41		3		3		164	164	328
89		6		6		456	356	712
113		8		8		452	452	904
191		12		12		764	765	1528
223		14		14		892	892	1784
281		18		18		1124	1124	2248
359		23		23		1436	1436	2872

Tabela 6.17: Trajanje posameznih izboljšanih operacij v Chebyshevih bazah

	Redukcija				**Kvadriranje**			
m	`char`	`short`	`int`	`long`	`char`	`short`	`int`	`long`
11	0.40	0.40	0.30	0.60	0.20	0.20	0.40	0.60
23	0.70	1.00	1.20	1.20	0.10	0.20	0.20	0.20
41	1.40	1.10	0.90	0.60	0.20	0.00	0.10	0.10
89	2.10	1.50	1.10	2.00	0.70	0.00	0.30	0.40
113	2.60	2.70	1.40	1.00	0.20	0.60	0.90	0.30
191	1.60	1.50	1.30	0.90	0.40	0.30	0.50	0.30
223	1.70	1.20	2.10	3.80	0.40	0.50	1.00	0.40
281	6.30	3.50	3.20	4.70	0.60	0.60	0.30	0.60
359	3.50	5.90	4.60	3.80	0.00	0.80	0.60	0.40

Tabela 6.18: Trajanje posameznih izboljšanih množenj v Chebyshevih bazah

	Množ. brez red.				**Množ. s hitro redukcijo**				**Modificirano množ.**			
m	`char`	`short`	`int`	`long`	`char`	`short`	`int`	`long`	`char`	`short`	`int`	`long`
11	0.5	0.8	1.0	0.7	1.0	1.5	1.5	1.6	0.1	0.6	0.5	0.0
23	2.6	1.2	1.1	1.2	2.9	2.5	1.9	2.4	1.0	1.0	0.6	0.4
41	6.0	3.2	2.3	1.9	7.2	5.1	3.1	2.1	1.2	1.2	1.0	0.3
89	22.3	10.9	7.1	5.6	25.7	15.4	9.1	6.8	8.1	5.0	2.4	1.1
113	32.0	16.4	8.9	7.3	39.1	22.5	14.1	10.3	13.1	7.7	4.5	2.9
191	80.4	45.6	24.0	14.2	106.0	57.5	30.3	18.3	31.1	16.1	7.5	5.6
223	107.4	64.2	31.1	19.4	135.9	73.5	34.8	29.6	42.1	24.5	11.5	8.3
281	173.4	91.9	45.2	28.9	228.0	114.7	57.1	41.5	74.7	37.6	18.1	13.0
359	264.9	144.9	81.0	41.3	351.4	188.0	82.4	57.6	106.6	55.9	31.2	17.2

Tabela 6.19: Trajanje posameznih računanj inverza v Chebyshevih bazah

	Izboljšan inverz				**Razširjen Evklidov algoritem**			
m	`char`	`short`	`int`	`long`	`char`	`short`	`int`	`long`
11	0.00	0.00	10.00	0.00	10.00	0.00	0.00	0.00
23	20.00	20.00	30.00	10.00	0.00	20.00	20.00	10.00
41	60.00	30.00	10.00	40.00	40.00	10.00	10.00	30.00
89	290.00	120.00	50.00	60.00	140.00	70.00	60.00	40.00
113	360.00	190.00	150.00	100.00	150.00	130.00	110.00	70.00
191	1380.00	800.00	490.00	370.00	420.00	300.00	150.00	170.00
223	1760.00	730.00	710.00	390.00	680.00	450.00	160.00	300.00
281	2590.00	1280.00	680.00	690.00	980.00	640.00	400.00	340.00
359	4400.00	2570.00	1520.00	980.00	1390.00	940.00	670.00	490.00

- Čeprav je izboljšana redukcija na prvi pogled ožitno hirejša(tip `char`, $m = 359$: razmerje 90/2154 operacij) se izkaže da je potrebno upoštevati še vektorska seštevanja vsakih 8 obrnjenih bitov vsaki besedi vektorja, tako da dobimo precej večje število operacij (isti primer: razmerje 2115/2154). Drugi razlog za počasnost redukcije s tabelami v praksi je dejstvo, da je potrebno $(3m-2)$–ico bitov priraviti za ustrezno obračanje s pomočjo tabel, kar nanese v najslabšem primeru $2(w-1)$ vektorskih pomikov v levo in desno.

- Evklidov algoritem potrebuje za izvedbo približno 20 do 25 optimiziranih modificiranih množenj oziroma približno 10 množenj brez redukcije za $m = 359$. Algoritmu uspe zreducirati stopnjo začetnega polinoma v približno 10–ih korakih oziroma v približno $10/359 = 3\%$ korakov vrednosti m.

Poglavje 7

Napadi s stranskim kanalom

Živimo v svetu, kjer se ne moremo izogniti digitalnim komunikacijam. Pravzaprav je skoraj težje komunicirati analogno kot digitalno. Zato postaja varnost in zaščita komunikacij v digitalni obliki vse bolj pomembna.

Digitalne komunikacije in njihova varnost so odvisne od kriptografskih sistemov, ki jih ščitijo. Ti pa so odvisni od strojne opreme, na kateri so postavljeni. Ta je lahko najrazličnejša, od mobilnih telefonov do prenosnih računalnikov. Vsem elektronskim napravam pa lahko prisluškujemo. Tako lahko spremljamo energijsko porabo procesorja, oddajanje toplote procesorja, porabo sistemskega spomina ob določenih operacijah. Vse to pa postavi osnove za **napade s stranskim kanalom**, napade na strojno opremo.

Poglavje se omeji na t.i. **preproste energijske analize** (*SPA– Simple power analysis*). Predpostavljamo, da kriptosistem, ki temelji na eliptičnih krivuljah izvede eno operacijo(npr. podpisovanje), ki jo nato analiziramo. Alternativa slednjim so *diferencialne energijske analize*(*DPA–differential power analysis*), kjer je potrebno kriptosistem zagnati večkrat, nato pa rezultate napademo s statističnimi orodji. Slednjim se izognemo že s preprosto naključnostjo vhodnih podatkov.

V tem poglavju najprej opišem binarni algoritem za potenciranje točke eliptične krivulje ter Montgomeryevo metodo seštevanja točk na eliptični krivulji, nato nekaj preprostih dejstev o energijskih napadih, nenazadnje pa opišem še dva načina za zaščito kriptosistemov pred takšnimi napadi.

7.1 Binarno potenciranje in Montgomeryeva metoda

V poglavju 2 smo opisali osnovni način za seštevanje in podvajanje točk. Hiter algoritem, ki izvaja seštevanje in podvajanje, opisano v razdelku 2.2.2 poglavja 2,

imenujemo **binarno potenciranje od leve proti desni**. Podan je v algoritmu 14.

ALGORITEM 14: ***Binarno potenciranje od leve proti desni***

INPUT: točka P in element obsega $k \in \mathbb{F}_q$, $k = \sum_{j=1}^{i} k_j 2^j$
OUTPUT: $Q = k \cdot P$

1. Q = P
2. if k_0 = 1 then R = P else R = 0
3. for i = 0 to $i-1$ do
   3.1. Q = $2Q$
   3.2. if k_i = 1 then R = R + Q
4. Return R

Peter L. Montgomery je v [3] leta 1987 razvil učinkovitejši algoritem za seštevanje in podvajanje točk na krivulji. Metoda je v tem razdelku opisana, ker jo v razdelku 7.2.2 o varnosti pred napadi s stranskim kanalom tudi ustrezno zaščitimo.

Zopet srečamo grupo eliptične krivulje $E(K)$, postavljeno nad obseg $K = \mathbb{F}_q$.

Montgomeryevo metodo uporabimo, ko računamo vsoto dveh točk $2P + Q$ [1] in ne potrebujemo vmesnih rezultatov podavajanja točke $2P$ ali vsote $P + Q$. S tem se znebimo enega množenja v obsegu $\mathbb{F}_q$, med samim začetnim računanjem pa ne potrebujemo koordinat y_1 in y_2 točk $P = (x_1, x_2)$ in $Q = (x_2, y_2)$. Metoda se nanaša na krivulje nad obsegom $\mathbb{F}_q$ in se enačba krivulje glasi

$$E \ : \ y^2 = x^3 + ax + b, \tag{7.1}$$

kjer sta števili $a, b \in \mathbb{F}_q$ in zanju velja $4a^3 + 27b^2 \neq 0$.

V algoritmih, opisanih za tip enačbe (7.1) porabimo za izračun podvojene točke $2P$ eno množenje, dve kvadriranji in eno deljenje v obsegu $\mathbb{F}_q$ (Pri tem, ne štejemo množenja s konstantama 2 ali 3)[2]. Za seštevanje dveh različnih točk $P + Q$ pa

[1] Operacijo uporabljamo predvsem pri potenciranju točke na krivulji do zelo velike potence (npr. faktorizacija velikih števil z eliptičnimi krivuljami ali digitalni podpisi z eliptičnimi krivuljami, saj je ekvivalentna metodi *square and multiply*, ki jo prav tako uporabljamo za izračun visokih potenc točke)

[2] Več o tem v razdelku 2.2.1 poglavja 2

porabimo eno množenje, eno kvadriranje in eno deljenje v obsegu $\mathbb{F}_q$. Podvajanje in seštevanje hkrati, ki vrne rezultat $2P+Q$ tako porabi dve množenji, tri kvadriranja in dve deljenji, če je izračunano v zaporedju $(P+P)+Q$, torej najprej podvajanje, nato pa seštevanje.

7.1.1 Montgomeryeva metoda

Izboljšan algoritem za operacijo podvajanja in seštevanja točk porabi eno množenje, dve kvadriranji in dve deljenji, poleg dodatnega kvadriranja v primeru, da sta točki enaki($P = Q$). Predpostavka je, da ne potrebujemo vmesnih rezultatov podvajanja $2P$ ali vsote $P + Q$.

Rezultat dobimo na spodaj opisani način.

1. Najprej izračunamo vsoto $P + Q$, kjer imata točki P in Q koordinate $P = (x_1, x_2)$ in $Q = (x_2, y_2)$, vendar zanemarimo izračun koordinate y_2 vsote $P+Q$, saj je nepomembna za nadaljni izračun. S tem se izognemo enemu množenju v obsegu $\mathbb{F}_q$.

2. Sedaj izračunamo še vsoto $(P + Q) + P$.

Predpostavimo, da sta točki $P = (x_1, x_2)$ in $Q = (x_2, y_2)$ različni na krivulji E in koordinati $x_1 \neq x_2$. Vmesna vsota $P + Q$ ima koordinati $P + Q = (x_3, y_3)$, končni rezultat pa koordinati $2P + Q = (x_4, y_4)$

Na takšen način se znebimo enega množenja v obsegu $\mathbb{F}_q$.

Primer 7.1. *Vzemimo obseg* $\mathbb{F}_2$ *in si oglejmo izračun* $1133044P = (10001010010011111010 0)_2 P$ *z binarnim potenciranjem od leve proti desni. Potenco točke bomo izračunali na dva načina: običajnega in izboljšanega. V obeh primerih predpostavimo, da je* $3P$ *predhodno izračunan. V tabeli so podani rezultati, kjer je notacija sledeča: a – seštevanje točk(addition), d – podvajanje točk(doubling), div – deljenje v obsegu(division), s – kvadriranj v obsegu(squaring), m – množenj v obsegu(multiplications)*

	klasična impl.	*izboljšana impl.*
$1133044P = 4(283261P)$	$2d$	$2d$
$283261P = 128(2213P) - 3P$	$7d + 1a$	$6d + 2a$
$2213P = 8(277) - 3P$	$3d + 1a$	$2d + 2a$
$277P = 8(35) - 3P$	$3d + 1a$	$2d + 2a$
$35P = 8(4P) + 3P$	$3d + 1a$	$2d + 2a$
$4P = P + 3P$	$1a$	$1a$
skupaj	23div + 41s + 23m	23div + 37s + 19m

Z izboljšanim algoritmom prihranimo 4 *kvadriranja in* 4 *množenja. Če ocenimo zahtevnost deljenja z petimi množenji, je prihranek približno* 4.5%.

ALGORITEM 15: ***Montgomeryeva ugotovitev***

INPUT: točki $P = (x_1, x_2)$ in $Q = (x_2, y_2)$
OUTPUT: točka $2P + Q$

1. $\lambda_1 = (y_2 - y_1)/(x_2 - x_1)$;
2. $x_3 = \lambda_1^2 - x_1 - x_2$;
   (skip $y_3 = (x_1 - x_3)\lambda_1 - y_1$;)
3. $\lambda_2 = -\lambda_1 - 2y_1/(x_3 - x_1)$; (and not $\lambda_2 = (y_3 - y_1)/(x_3 - x_1)$)
4. $x_4 = \lambda_2^2 - x_1 - x_3$;
5. $y_4 = (x_1 - x_4)\lambda_2 - y_1$;
6. Return (x_4, y_4);

7.1.2 Projektivne koordinate

Vse zgornje formule za izračun λ vsebujejo invertiranje v obsegu K, kar je precej zamudno. Zato uvedemo projektivne koordinate.

V okviru **projektivnih homogenih koordinat** postavimo koordinato x na vresdnost $x = X/Z$ in koordinato y na vrednosti $y = Y/Z$ in tako spremenimo Weierstrassevo enačbo v

$$E \ : \ Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$$

Točka v neskončnosti je predstavljena kot $(0, \theta, 0)$ za nek $\theta \in K \setminus \{\infty\}$. Afina točka (x_1, y_1) je predstavljena z projektivno točko $(\theta x_1, \theta y_1, \theta)$ za nek $\theta \in K \setminus \{\infty\}$. Projektivna točka (X_1, Y_1, Z_1) ustreza afini točki $(X_1/Z_1, Y_1/Z_1)$.

Pravila za seštevanje in podvajanje dobimo iz zgornjih formul, če nadomestimo x_1, x_2, y_1 in y_2 z njihovimi projektivnimi ekvivalenti.

7.2 SPA napadi

Pri kriptosistemih z eliptičnimi krivuljami je najpogosteje uporabljen algoritem za potenciranje točk *double and add* algoritem, ki ga izvajamo na način, kot ga opiše algoritem 14.

Predpostavimo, da sta podvajanje točke in seštevanje dveh različnih točk implementirana na različna načina. Posledica je, da lahko z SPA napadom ločimo operaciji. Iz izvedbe operacije lahko pridobimo precej informacij, npr. porabo energije procesorja(energijsko sled), temperaturo procesorja, prostorsko porabo pomnilnika. Ko recimo energijska sled pokaže, da podvajanju sledi seštevanje vemo, da je trenutni bit, recimo mu k_i, enak 1. Sicer je $k_i = 0$.

Najpreprostejši načini, da preprečimo takšne napade so:

1. vstavljanje lažnih operacij,
2. uporaba drugačnega tipa parametrizacije eliptične krivulje in
3. uporaba algoritma, ki je že varen pred SPA napadi.

Oglejmo si primer za točko (i). V primeru 7.1. smo si ogledali delovanje algoritma za binarno podvajanje in seštevanje točk.

Vidimo, da se v primeru, ko je $k_i = 1$ izvede tudi seštevanje dveh točk, tako da bo lahko napadalec iz zapisa energijske sledi razbral vrednosti k_i. Zato je *Jean Sebastien Coron* v [8] predlagal *double and* ***always*** *add* algoritem, kjer tudi v primeru, ko je $k_i = 0$ opravimo lažno seštevanje dveh točk. Na takšen način izpeljemo varen oziroma nerazločen algoritem za seštevanje in podvajanje točk na račun večjega števila seštevanj, ki je opisan v algoritmu 16.

ALGORITEM 16: ***Varno binarno potenciranje***

INPUT: točke P, T_1, T_2 in element obsega $k \in \mathbb{F}_q$, $k = \sum_{j=1}^{i} k_j 2^j$
OUTPUT: $Q = k \cdot P$

---

1. Q = P
2. if k_0 = 1 then R = P else R = 0
3. for i = 0 to $i - 1$ do
   - 3.1. Q = $2Q$
   - 3.2. if k_i = 1 then R = R + Q
   - 3.3. else T_1 = T_1 + T_2
4. Return R

7.2.1 Pregled formule za seštevanje točk

Ponovimo najprej geometrijo, ki se skriva za seštevanjem dveh točk oziroma podvajanju točke na eliptični krivulji.

Skozi točki P in Q potegnemo premico ℓ (v primeru podvajanja točke($P = Q$) je premica ℓ enaka tangenti). Premica seče krivuljo v tretji točki eliptične krivulje R'. Točko R' nato prezrcalimo preko x osi in tako dobimo vsoto $R = P + Q$. Možno pa je zapisati naklon premice ℓ λ na takšen način, da bo formula pravilna tako za seštevanje kot za podvajanje.

Trditev 7.2. *Naj bo E eliptična krivulja nad obsegom K, podana z enačbo*

$$E_{/K} \ : \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

Naj bosta točki $P = (x_1, y_1), Q = (x_2, y_2) \in E(K) \setminus \{0\}$, kjer za koordinati y obeh točk velja zveza $y_1 \neq -y_2$. Naj bodo

$$\begin{aligned} x_3 &= \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \\ y_3 &= -(\lambda + a_1)x_3 - \mu - a_3 \ in \\ \mu &= y_1 - \lambda x_1. \end{aligned}$$

Potem je vsota enaka $P + Q = (x_3, y_3)$ in

$$\lambda = \frac{x_1^2 + x_1x_2 + x_2^2 + a_2x_1 + a_2x_2 + a_4 - a_1y_1}{y_1 + y_2 + a_1x_2a_3}.$$

Dokaz. Pogoj $y(P) \neq y(-Q)$ je ekvivalenten $y_1 \neq -y_2 - a_1x_2 - a_3$. Iz definicije λ dobimo

$$\begin{aligned} \lambda &= \frac{y_1 - y_2}{x_1 - x_2} \\ &= \frac{y_1 - y_2}{x_1 - x_2} \cdot \frac{y_1 - (-y_2 - a_1x_2 - a_3)}{y_1 - (-y_2 - a_1x_2 - a_3)} \\ &= \frac{y_1^2 + a_1x_2y_1 + a_3y_1 - y_2^2 - a_1x_2y_2 - a_3y_2}{(x_1 - x_2)(y_1 + y_2 + a_1x_2 + a_3)} \\ &= \frac{(y_1^2 + a_1x_1y_1 + a_3y_1) - (y_2^2 + a_1x_2y_2 + a_3y_2) + a_1x_2y_1 - a_1x_1y_1}{(x_1 - x_2)(y_1 + y_2 + a_1x_2 + a_3)} \\ &= \frac{(x_1^3 + a_2x_1^2 + a_4x_1 + a_6) - (x_2^3 + a_2x_2^2 + a_4x_2 + a_6) - a_1y_1(x_1 - x_2)}{(x_1 - x_2)(y_1 + y_2 + a_1x_2 + a_3)} \\ &= \frac{x_1^2 + x_1x_2 + x_2^2 + a_2x_1 + a_2x_2 + a_4 - a_1y_1}{y_1 + y_2 + a_1x_2 + a_3}. \end{aligned}$$

Če v formuli zamenjamo koordinato x_2 z x_1 in koordinato y_2 z y_1 (v primeru enakosti $P = Q$), dobimo za λ enačbo

$$\lambda = \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3},$$

kar je enačba za λ pri podvajanju točke. ■

Zgornjo trditev lahko prenesemo še na preprostejšo obliko Weierstrasseve enačbe, opisani v poglavju 2.

Posledica 7.3. *Naj bo K obseg karakteristike $char(K) \neq 2, 3$ in naj bo E eliptična krivulja, podana z enačbo*

$$E_{/K} \ : \ y^2 = x^3 + ax + b.$$

Potem za poljubni točki $P = (x_1, y_1), Q = (x_2, y_2) \in E(K) \setminus \{0\}$, kjer je $y_1 \neq y_2$, velja, da je vsota $P + Q = (x_3, y_3)$, za katero velja

$$x_3 = \left(\frac{x_1^2 + x_1x_2 + x_2^2 + a}{y_1 + y_2}\right)^2 - x_1 - x_2 \tag{7.2}$$

in

$$y_3 = \left(\frac{x_1^2 + x_1x_2 + x_2^2 + a}{y_1 + y_2}\right)(x_1 - x_3) - y_1. \tag{7.3}$$

Posledica 7.4. *Naj bo K obseg karakteristike $char(K) = 2$ in naj bo E nesupersingularna eliptična krivulja, podana z enačbo*

$$E_{/K} \ : \ y^2 + xy = x^3 + ax + b.$$

Potem za poljubni točki $P = (x_1, y_1), Q = (x_2, y_2) \in E(K) \setminus \{0\}$, kjer sta koordinati $y_1 \neq y_2 + x_2$ velja za vsoto enakost $P + Q = (x_3, y_3)$, v kateri za koordinati velja

$$\begin{aligned} x_3 &= \left(\frac{x_1^2 + x_1x_2 + x_2^2 + ax_1 + ax_2 + y_1}{y_1 + y_2 + x_2}\right)^2 + \\ &+ \left(\frac{x_1^2 + x_1x_2 + x_2^2 + ax_1 + ax_2 + y_1}{y_1 + y_2 + x_2}\right) + a + x_1 + x_2 \end{aligned} \tag{7.4}$$

in

$$y_3 = \left(\frac{x_1^2 + x_1x_2 + x_2^2 + ax_1 + ax_2 + y_1}{y_1 + y_2 + x_2}\right)(x_1 + x_3) + x_3 + y_1. \tag{7.5}$$

Zaradi dejstva, da nad obsegom K s karakteristiko $char(K) \neq 2, 3$ velja enakost $x_1^2 + x_1x_2 + x_2^2 = (x_1 + x_2)^2 - x_1x_2$, v zgornjih formulah (7.2) in (7.3) potrebujemo le eno invertiranje in 5 množenj za seštevanje dveh točk. V obsegih s karakteristiko

$char(K) = 2$, v katerih veljata formuli (7.4) in (7.5) pa potrebujemo eno invertiranje in 3 množenja v obsegu skupaj z dodatnim množenjem s konstanto za seštevanje dveh točk. Zanemarili smo ceno kvadriranja v obsegu K.

Enačbi (7.2) in (7.3) pretvorimo še v projektivno (homogeno) obliko.

Zapišimo

$$\lambda = \frac{x_1^2 + x_1x_2 + x_2^2 + a}{y_1 + y_2} = \frac{(x_1 + x_2)^2 - x_1x_2 + a}{y_1 + y_2}.$$

Zaradi simetrije λ lahko enačbo (7.3) prepišemo v obliki

$$y_3 = \lambda(x_2 - x_3) - y_2,$$

saj je vsota komutativna ($P + Q = Q + P$). Zato dobimo enakost

$$2y_3 = \lambda(x_1 + x_2 - 2x_3) - (y_1 + y_2).$$

Če postavimo $x_i = X_i/Z_i$ in $y_i = Y_i/Z_i$ dobimo po nekaj korakih enačbe

$$\begin{aligned} X_3 &= 2FW, \\ Y_3 &= R(G - 2W) - L^2 \text{ in} \\ Z_3 &= 2F^3, \end{aligned} \tag{7.6}$$

kjer so

$$\begin{array}{llll} U_1 = X_1Z_2 & U_2 = X_2Z_1 & S_1 = Y_1Z_2 & S_2 = Y_2Z_1 \\ Z = Z_1Z_2 & T = U_1 + U_2 & M = S_1 + S_2 & R = T^2 - U_1U_2 + aZ^2 \\ F = ZM & L = MF & G = TL & W = R^2 - G. \end{array}$$

Vidimo, da seštevanje dveh točk z zgornjimi poenotenimi formulami zahteva 17 množenj plus eno množenje s konstanto oziroma v primeru, ko je vrednost $a = -1$ in lahko zapišemo $R = (T - Z)(T + Z) - U_1U_2$, le 16 množenj.

7.2.2 Posplošitev Montgomeryeve metode

V začetku poglavja smo opisali Montgomeryevo metodo seštevanja točk na eliptični krivulji, kjer v vmesnem računanju ne uporabljamo y koordinat točk vmesnih rezultatov.

Opazimo, da razlika točk $R_1 - R_0$ ostane nespremenjena skozi algoritem. Zato jo lahko označimo kot $R_1 - R_0 = P$. Montgomery se je zaradi učinkovitosti omejil le na eliptčne krivulje, katerih enačba ima obliko $y^2 = x^3 + ax^2 + b$ nad obsegom K karakteristike $char(K) \neq 2, 3$. Podajmo trditev, ki opiše formulo v splošnem.

ALGORITEM 17: ***Montgomeryeva metoda seštevanja točk***

INPUT: P, $k = (k_{l-1}, \ldots, k_0)_2$
OUTPUT: $x(kP)$

1. $R_0 = P$; $R_1 = 2P$
2. for $i = l-2$ downto 0 do
   if $k_i = 0$ then
   $x(R_1) = x(R_0 + R_1)$; $x(R_0) = x(2R_0)$;
   else if $k_i = 1$ then
   $x(R_0) = x(R_0 + R_1)$; $x(R_1) = x(2R_1)$;
3. Return $x(R_0)$

Trditev 7.5. *Naj bo K obseg karakteristike $char(K) \neq 2, 3$ in naj bo E eliptična krivulja podana z enačbo*

$$E_{/K} \; : \; y^2 = x^3 + ax + b.$$

Naj bosta točki $P = (x_1, y_1)$ in $Q = (x_2, y_2) \in E(K) \setminus \{0\}$, za kateri velja $P \neq \pm Q$. Skupaj z izračunano razliko $P - Q = (x, y)$ velja za x koordinato vsote $P + Q = (x_3, y_3)$ enakost

$$x(P+Q) = x_3 = \frac{-4b(x_1 + x_2) + (x_1 x_2 - a)^2}{x(x_1 - x_2)^2}. \tag{7.7}$$

Poleg tega v primeru, da je koordinata $y_1 \neq 0$, velja za x koordinato točke $2P = (x_4, y_4)$ še enakost

$$x(2P) = x_4 = \frac{(x_1^2 - a)^2 - 8bx_1}{4(x_1^3 + ax_1 + b)}. \tag{7.8}$$

Dokaz. Oglejmo si enačbo (7.2) in koordinato x_3 vsote $P + Q$:

$$\begin{aligned} x_3(x_1 - x_2)^2 &= (y_1 - y_2)^2 - (x_1 + x_2)(x_1 - x_2)^2 \\ &= (y_1^2 + y_2^2 - 2y_1 y_2) - (x_1^3 + x_2^3 - x_1^2 x_2 - x_1 x_2^2) \\ &= -2y_1 y_2 + 2b + (a + x_1 x_2)(x_1 + x_2). \end{aligned}$$

Podobno za x koordinato $P - Q$ velja

$$x(x_1 - x_2)^2 = 2y_1 y_2 + 2b + (a + x_1 x_2)(x_1 + x_2).$$

Če sedaj pomnožimo obe enačbi med seboj dobimo enakost

$$\begin{aligned} x_3 \cdot x(x_1 - x_2)^2 &= -4y_1^2y_2^2 + (2b + (a + x_1x_2)(x_1 + x_2))^2 \\ &= -4(x_1^3 + ax_1 + b)(x_2^3 + ax_2 + b) + (2b + (a + x_1x_2)(x_1 + x_2))^2 \\ &= (-4b(x_1 + x_2) + (x_1x_2 - a)^2)(x_1 - x_2)^2, \end{aligned}$$

kar po deljenju z $(x_1 - x_2)^2$ da pravilen rezultat.

Kadar je koordinata $y_1 \neq 0$ (npr. ko je točka $2P \neq 0$) sledi iz enačbe (7.2) enakost

$$x(2P) = \frac{(3x_1^2 + a)^2}{4y_1^2} - 2x_1 = \frac{(x_1^2 - a)^2 - 8bx_1}{4(x_1^3 + ax_1 + b)}.$$

■

Še ena uporabna lastnost Montgomeryeve metode je, da lahko y koordinato točke P izračunamo iz njene x koordinate, x koordinate neke druge točke Q in iz koordinat razlike $P - Q$. Natančneje to opredeli naslednja trditev.

Trditev 7.6. *Naj bo K obseg karakteristike $char(K) \neq 2, 3$ in naj bo E eliptična krivulja podana z enačbo*

$$E_{/K} \ : \ y^2 = x^3 + ax + b.$$

Naj bosta točki $P = (x_1, y_1)$ in $Q = (x_2, y_2) \in E(K) \setminus \{0\}$ in naj velja $P \neq \pm Q$. Skupaj z izračunano točko $P - Q = (x, y)$ velja za y koordinato točke P

$$y(P) = y_1 = \frac{2b + (a + xx_1)(x + x_1) - x_2(x - x_1)^2}{2y} \tag{7.9}$$

Dokaz. Definirajmo točko $D = P - Q = (x, y)$. Ker je vsota $Q = P + D = (x_2, y_2)$, dobimo iz enačbe (7.2) enakost

$$x_2 = \left(\frac{y_1 - y}{x_1 - x}\right)^2 - x_1 - x = \frac{-2yy_1 + 2b + (a + xx_1)(x + x_1)}{(x_1 - x)^2},$$

kar nam pomnoženo z $(x_1 - x)^2$ zaključi dokaz. ■

Oglejmo si zgornje ugotovitve še v (homogenih) projektivnih koordinatah. Predpostavimo, da delamo nad obsegom K z $char(K) \neq 2, 3$. Enačba 7.7 tako postane

$$\begin{aligned} X(P + Q) &= -4bZ_1Z_2(X_1Z_2 + X_2Z_1) + (X_1X_2 - aZ_1Z_2)^2, \\ Z(P + Q) &= x \cdot (X_1Z_2 - X_2Z_1)^2. \end{aligned}$$

Seštevanje točk torej zahteva 7 množenj plus 3 množenja s konstanto. Za podvajanje točk se formula (7.8) spremeni v

$$\begin{aligned} X(2P) &= (X_1^2 - aZ_1^2)^2 - 8bX_1Z_1^3, \\ Z(2P) &= 4Z_1(X_1^3 + aX_1Z_1^2 + bZ_1^3)/ \end{aligned}$$

in zahteva 7 množenj in 2 množenji s konstanto.

7.3 Zaključek

V poglavju smo pokazali dva osnovna načina zaščite kriptosistemov in algoritmov pred napadi s stranskim kanalom in preprosto energijsko analizo.

Na prvem mestu smo omenili preprost a kljub temu drag način zaščite z dodajanjem lažnih operacij. S tem dosežemo enakost operacij za opazovalca vendar izgubimo morebitne prednosti pri hitrosti ene izmed obeh operacij. *Double and always add* metoda navzven ne kaže razlike med seštevanjem in podvajanjem točk in tako ne omogoča določitve vrednosti števila k v rezultatu $R = kP$.

Sledil je način združevanja formul v eno samo. Tako smo v razdelku 7.2.1 združili formuli za seštevanje in podvajanje točk v eno samo. Izvirne enačbe namreč niso omogočale uporabe v obeh primerih: seštevanju in podvajanju točk. Žal nas je izboljšava stala še dodatno kvadriranje in množenje v primeru podvajanja točke in dve dodatni kvadriranji in dodatno množenje v primeru seštevanja točk. V praksi se iskaže, da je deljenje tista operacija katere število želimo obdržati čim manjše.

Enak prijem smo si ogledali na *Montgomeryevi* metodi podvajanja in seštevanja točk. Prvoten osnovni algoritem za izračun rezultata $R = kP$ v vsakem koraku tri kvadriranja in dve deljenji za izračun le x komponente rezultata. Pri tem je potrebno poudariti, da v osnovni metodi ločimo med podvajanjem($2P$) in seštevanjem točk($P + Q$). Izboljšana metoda uporablja za obe operaciji eno kvadriranje dve kvadriranji, eno množenje in eno deljenje oziroma skupaj štiri kvadriranja, štiri množenja in dve deljenji ob predpstavki, da predhodno poznamo vrednost rezultata $P - Q$. Pomembno dejstvo je, da sedaj ne ločimo med obema operacijama.

Z opisanimi prijemi smo pokazali, kako lahko kriptosistem na eliptičnih krivuljah zaščitimo pred napadi s stranskim kanalom na takšen način, da zunanjo opazovalec ne more ločiti operacij in tako določiti skritih podatkov.

Dodatek A

Zahtevnost Algoritma

Algoritme najpogosteje ločimo med seboj glede na porabljen čas in prostor. Ker je natančen čas in prostor težko opredeliti, si pomagamo z **asimptotičnimi ocenami**, ki nam povedo, kako se potreben čas in prostor povečujeta s povečanjem vhodnih podatkov. Te asimptotične ocene imenujemo **časovna** in **prostorska** zahtevnost. Za predstavitev zahtevnosti uporabljamo $\mathcal{O}$–**notacijo**. Le–to bomo sedaj definirali. Naj bosta $f, g : \mathbb{N} \to \mathbb{R}$ neki funkciji. Pravimo, da

f asimptotično ne raste hitreje kot g oziroma, da je g asimptotična zgornja meja za f, če obstajata taki konstanti $c > 0$ in $n_0 > 0$, da velja $0 \leq f(n) \leq c\, g(n)$ za vse $n \geq n_0$ ali ekvivalentno, če obstaja $\lim_{n\to+\infty} f(n)/g(n)$. Simbolično bomo to zapisali z $f(n) = \mathcal{O}(g(n))$ ali krajše z $f = \mathcal{O}(g)$. Na primer, $\mathcal{O}(n^3) = 3n^3 + 2n + 2$.

Velikost vhodnih podatkov v algoritem se meri s številom bitov, ki jih zasedajo. Na primer, za zapis števila n potrebujemo $m = \lfloor \log_2 n \rfloor + 1$ bitov. zato običajno izrazimo zahtevnost algoritma v odvisnosti od števila m. S tem dosežemo najbolj jasno predstavo o zahtevnosti algoritma. Naj bo f časovna zahtevnost algoritma in e osnova naravnega algoritma. Glede na f delimo algoritme na

- **konstantne**, tj. $f(m) = \mathcal{O}(1)$. Na primer, $f(m) = c$, kjer je c poljubna pozitivna konstanta
- **polinomske**, tj. $f(m) = \mathcal{O}(m^k)$, kjer je k neka pozitivna konstanta. Na primer $f(m) = m^2$.
- **eksponentne**, tj. $f(m) = e^{g(m)}$, kjer je $g : \mathbb{N} \to \mathbb{R}$ taka funkcija, za katero velja $g(m) = \mathcal{O}(m^k)$, in k neka pozitivna konstanta. Na primer, $f(m) = e^m$.

Polinomske algoritme običajno smatramo za **učinkovite**, eksponentne algoritme pa za **neučinkovite**. Ker delamo v končnih obsegih predpostavimo, da opravimo vsako operacijo v obsegu v konstantnem času. Zato merimo zahtevnost algortimov s številom opravljenih operacij v obsegu.

Literatura

[1] T. Itoh, S. Tsuji *A fast Algorithm for Computing Multiplicative Inverses in $\mathbb{F}_{2^m}$ Using Normal Bases*, Information and Computation, Volume 78, Issue 3: 171–177, 1988.

[2] A. Jurišić, *Umbral optimal normal bases*, rokopis.

[3] P. L. Montgomery, *Speeding the Pollard and elliptic curve methods of factorization*, Mathematics of computation, 48(177): 243–264, 1987.

[4] I. F. Blake, Gadiel Serioussi, Nigel P. Smart, *Advances in elliptic curve cryptography*, London mathematical society, Canbridge university press, 2005.

[5] D. Hankerson, A. Menezes, S. Vanstone, *Guide to elliptic curve cryptography*, Springer 2004.

[6] D. Hankerson, J. L. Hernandez and A. Menezes, *Software implementation of elliptic curve cryptography over binary fields*, Cryptographic hardware and embedded systems - CHES 2000, 2000.

[7] A. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, T. Yaghoobian, *Applications of finite fields*, The International Series in Engineering and Computer Science, Springer, 1993.

[8] J. S. Coron, *Resistance against differential power analysis for elliptic curve cryptosystems*, In C.K. Koc and C. Paar, editors, *Cryptographic Hardware and embedded systems CHES 99*, volume 1717 of lecture notes in computer science, pages 292–302, Springer-Verlag, 1999.

[9] D. H. Stinson, *Some observations on parallel algorithms for fast exponentiation in $GF(2^n)$*, SIAM J. Computing 19: 711-717, 1990.

[10] J. Von Zur Gathen, *Efficient and optimal exponentiation in finite fields*, Computational Complexity 1: 360-394, 1991.

[11] V. Nastran, *Baze binarnih končnih obsegov*, Diplomsko delo, Fakulteta za matematiko in fiziko, Univerza v Ljubljani, Ljubljana, 2003.

[12] R. Lidl, H. Niederreiter, *Finite fields*, Cambridge University Press, 1987.

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

[14] I. Vidav, *Kubične krivulje*, Obzornik za matematiko in fiziko 34(2), DMFA, Ljubljana 1987.