Noen aspekter ved implementasjon og ytelse for kryptosystemer basert på elliptiske kurver

av

Terje Gjøsæter og Kjetil Haslum

Hovedoppgave til mastergraden i
informasjons- og kommunikasjonsteknologi

Høgskolen i Agder

Grimstad, Mai 2003

Kapittel 1

Sammendrag

I den senere tid har kryptografi basert på elliptiske kurver blitt tatt i bruk i økende grad. De fleste av dagens implementasjoner baserer seg på kurver på Weierstrassform, $y^2z + a_1xyz + a_3yz^2 = x^3 + a_2x^2z + a_4xz^2 + a_6z^3$. Hesseform av elliptiske kurver, $x^3+y^3+z^3 = Dxyz$, har en del egenskaper som gjør den velegnet til bruk i kryptografi. Basert på antall grunnoperasjoner som utføres i algoritmene, bør Hesseform være bedre enn Weierstrassform ytelsesmessig. I dette arbeidet utfører vi en del hastighetstester av punktoperasjoner på kurver på kort Weierstrass- og Hesseform. Vi har implementert algoritmer for punktaddisjon, punktdobling og punktmultiplikasjon for både kort Weierstrassform og Hesseform. I disse testene har vi begrenset oss til bare å behandle kurver over kropper med primtallskarakteristikk.

For å kunne utføre testene har vi bruk for kurver på både kort Weierstrassform og Hesseform som er birasjonale med hverandre. Vi har derfor laget konverteringsfunksjoner for kurver mellom de to formene. For å teste disse funksjonene har vi også laget en fullstendig oversikt over elliptiske kurver på Hesseform med karakteristikk mellom 5 og 23, hvor vi blant annet angir korresponderende kurver på kort Weierstrassform, og kommenterer gruppestrukturen.

Vi har laget en funksjon for å finne elliptiske kurver på Hesseform som egner seg for bruk i kryptografi. Vi har generert noen kurver over kropper med karakteristikk med lengde 160 bit og 240 bit som vi har konvertert til kort Weierstrassform, og disse danner grunnlag for ytelsestestene.

Vi ser på en del aspekter ved punktmultiplikasjonsalgoritmer, bruk av blandede koordinater, og forhåndslagring av doblede punkter. Vi ser på teoretiske kjøretider basert på antall grunnoperasjoner i algoritmene, og sammenligner disse med våre målte verdier. Vi observerer at Hesseform vinner ytelsesmessig på de fleste punkter, og bør dermed være velegnet for videre vurdering for bruk i kryptosystemer basert på elliptiske kurver.

Kapittel 2

Forord

Dette er en hovedoppgave for mastergraden i IKT ved Høyskolen i Agder.

Vi ønsker å takke veileder førsteamanuensis Trond Stølen Gustavsen for god oppfølging, og spesielt for god innføring i grunnleggende gruppeteori og annen bakgrunnskunnskap som har vært nødvendig for å kunne ta fatt på denne oppgaven.

Vi vil takke studieleder Stein Bergsmark for gode råd om hovedoppgaveskriving og konstruktiv kritikk til deler av rapporten.

Vi vil også takke forskningssjef Jakob Gjøsæter for gode råd og kommentarer til rapporten.

Vi har benyttet dokumentformateringssystemet LaTeX 2_ε til utforming av denne rapporten, og i den forbindelse vil vi takke medstudent Bent André Solheim for gode tips om bruk av dette systemet.

Innhold

1 Sammendrag **3**

2 Forord **5**

3 Innledning **11**
3.1 Kryptografi 11
3.2 Elliptiske kurver 12
3.3 Forskningsstatus på området i dag 13
3.4 Rapportens struktur 14

4 Introduksjon til kryptografi **15**
4.1 Innledning 15
4.2 Asymmetrisk kryptografi 15
4.2.1 Matematisk bakgrunn 15
4.2.2 Nøkkelutveksling 16
4.2.3 Diffie-Hellman 16
4.2.4 ElGamal 16
4.2.5 Hash-funksjoner 17
4.2.6 Digital signatur 17
4.2.7 Digitalt sertifikat 17
4.3 Kryptografi basert på elliptiske kurver 18
4.3.1 Elliptic Curve Diffie-Hellman 18
4.3.2 Elliptic Curve ElGamal 19
4.4 Transport Layer Security og Secure Socket Layer 19
4.5 Eksempel på bruk av TLS med ECC 20

5 Introduksjon til elliptiske kurver **23**
5.1 Innledning 23
5.2 Elliptiske kurver på Weierstrassform 23
5.2.1 Kort Weierstrassform 24
5.2.2 Gruppeloven 24
5.2.3 Formler for addisjon og dobling 27
5.3 Projektive koordinater 27
5.4 Jacobiske projektive koordinater 28
5.5 Gruppeorden 29
5.5.1 Hasses teorem 29
5.5.2 Beregning av gruppeorden 30
5.6 Elliptiske kurver på Hesseform 30

5.6.1 Gruppeloven 31
5.6.2 Tretorsjonspunkter 33

6 Implementasjon av Weierstrassform 37
6.1 Innledning 37
6.2 Pakker og klasser i LiDIA som vi har brukt 37
6.3 Hastighetstestmetoder 38
6.4 Hastighetstest av grunnoperasjoner i LiDIA 38
6.5 Eksempler på elliptiske kurver 39
6.6 Punktene på den elliptiske kurven 41
6.7 Affin punktaddisjon 41
6.8 Punktaddisjon i Jacobisk projektive koordinater 43
6.8.1 Blandede koordinater 44
6.9 Punktmultiplikasjon 46
6.9.1 "Right-to-left" algoritmen 46
6.9.2 "Left-to-right" algoritmen 46
6.9.3 Modifisert "Left-to-right" algoritme 47
6.9.4 Forhåndslagring 49
6.10 Oppsummering 49

7 Implementasjon av Hesseform 51
7.1 Innledning 51
7.2 Oversikt over "små" Hessekurver 51
7.2.1 Strukturen 53
7.3 Punktaddisjon 56
7.4 Generering av tilfeldige punkter 58
7.5 Omforming av kurver 59
7.5.1 Fra Hesseform til Weierstrassform 59
7.5.2 Fra Weierstrassform til Hesseform 60
7.6 Beskyttelse mot sidekanalangrep 61
7.7 Punkter på små elliptiske kurver på Hesseform 62
7.8 Kryptografisk sterke kurver på Hesseform 62
7.9 Oppsummering 63

8 Testresultater og drøfting 65
8.1 Grunnleggende kroppsoperasjoner i LiDIA 65
8.1.1 Diskusjon av testoppsett 65
8.1.2 Kommentarer til resultater 66
8.2 Grunnleggende punktoperasjoner 67
8.2.1 Valg av algoritmer for Weierstrassform 67
8.2.2 Valg av algoritmer for Hesseform 68
8.2.3 Resultater 68
8.3 Multiplikasjon 71
8.4 Observasjoner om gruppestruktur 73

9 Konklusjon 75

A Matematikk **79**
A.1 Innledning . . . 79
A.2 Gruppeteori . . . 79
A.2.1 Addisjon modulo n . . . 81
A.2.2 Sykliske grupper . . . 82
A.2.3 Lagranges teorem . . . 83
A.2.4 Fermats teorem . . . 84
A.2.5 Strukturteorem for abelske grupper . . . 85
A.2.6 Homomorfi og isomorfi . . . 87
A.2.7 Kvotienter av abelske grupper . . . 88
A.3 Ringer, modulær aritmetikk . . . 89
A.3.1 Integritetsområder og kropper . . . 90
A.3.2 Underringer og idealer . . . 91
A.4 Endelige kropper . . . 92
A.4.1 Polynomringer . . . 92
A.4.2 Irredusible polynomer . . . 93
A.4.3 Kroppsutvidelser . . . 94
A.4.4 Galoiskropper . . . 96

B Kompilering og installasjon av LiDIA **97**

C Kildekode **99**
C.1 Innledning . . . 99
C.2 Funksjoner . . . 99

D Eksempler på elliptiske kurver **113**
D.1 Innledning . . . 113
D.2 Kurver . . . 113

Kapittel 3

Innledning

3.1 Kryptografi

Kryptografi har mange anvendelser, og særlig etter at internett har blitt allemannseie har bruk av nettbanker og netthandel bidratt til at mange har kommet i kontakt med kryptografi. Man får imidlertid ikke nødvendigvis noe bevisst forhold til det som foregår, siden nettleseren i mange tilfeller utfører de nødvendige kryptografiske operasjonene automatisk.

Det nevnes ofte tre viktige problemstillinger som kryptografi forsøker å dekke; konfidensialitet, autentisitet, og integritet. Man ønsker å kommunisere uten å kunne avlyttes av tredjepart, man ønsker å være sikker på at den man kommuniserer med er den han gir seg ut for å være, og man ønsker å være sikker på at informasjonen ikke endres under overføring.

En mer formell definisjon av kryptografi finner vi i [1] side 4: *"Kryptografi er studiet av matematiske teknikker relatert til aspekter ved informasjonssikkerhet slik som konfidensialitet, dataintegritet, entitetsautentisering og dataopphavsautentisering."*

Det er imidlertid viktig å huske at en kjede ikke er sterkere enn svakeste ledd. Sterke kryptografiske protokoller og algoritmer er ikke nok. Dersom sikkerheten svikter i andre deler av systemet, kan man få informasjon ut som kan hjelpe til å knekke kryptosystemet. En variant av denne typen kryptografiske angrep, eller *kryptanalyse*, kalles sidekanalangrep, og baserer seg på å bruke informasjon fra fysiske sideeffekter av algoritmene, som for eksempel kjøretid eller strømforbruk.

Et av de tidligste kjente eksemplene på bruk av kryptografi er det såkalte Cæsar-chiffer. Det ble benyttet av Julius Cæsar, og går ut på å bytte hvert symbol med symbolet tre plasser til høyre modulo antall symboler i alfabetet. Modulo betyr enkelt forklart at når man kommer til enden av alfabetet, går man rundt og begynner på begynnelsen igjen. For å komme tilbake til utgangspunktet må man altså erstatte hvert symbol med symbolet tre plasser til venstre i alfabetet. Rot13 er en variant av dette som har samme operasjon for både kryptering og dekryptering siden den tar utgangspunkt i det engelske alfabetet som har 26 bokstaver.

Av senere kjente historiske eksempler på kryptosystemer kan nevnes *Enigma*, som var en mekanisk spesialmaskin for kryptering og dekryptering brukt før og under 2. verdenskrig. Alan Turing og Gordon Welchman konstruerte i 1939-40

en maskin som kunne knekke meldinger fra denne.

En kryptografisk operasjon foregår som regel etter en liste av presise instruksjoner, beskrevet i en *protokoll*. En kryptografisk protokoll defineres slik i [1]: *"En kryptografisk protokoll er en distribuert algoritme definert av en sekvens av skritt som presist spesifiserer handlingene som kreves av to eller flere entiteter for å oppnå et bestemt sikkerhetsformål."* Eksempler på kryptoprotokoller er *SSL* (Secure Socket Layer) og *TLS* (Transport Layer Security) som er en videreutvikling av SSL. Disse kan for eksempel benyttes av en nettleser når man skal ha en sikker kommunikasjon med et nettsted, for eksempel en nettbank.

En kryptografisk algoritme er et sett med matematiske operasjoner som til sammen utfører en kryptografisk basisoperasjon som for eksempel kryptering og dekryptering.

Kryptografiske protokoller vil ofte ha støtte for å velge mellom flere forskjellige algoritmer til de kryptografiske operasjonene.

Symmetrisk kryptografi har som utgangspunkt at samme nøkkel benyttes for både kryptering og dekryptering. Dette kan beskrives slik:

$$E_k(M) = C, \quad D_k(C) = M, \quad D_k(E_k(M)) = M$$

Her representerer E krypteringsfunksjonen, D er dekrypteringsfunksjonen, k er nøkkel, C er ciphertekst og M er klartekst.

Man opererer ofte på blokker av data av en gitt størrelse. Dette kalles *blockcipher*. Mye brukte algoritmer er AES, DES, Triple-DES, RC5, Blowfish, IDEA. Man kan også operere på en strøm av data, dette kalles *streamcipher*, som f. eks. i RC4.

Et problem med symmetrisk kryptografi er distribusjon av nøkler. Begge parter må ha identiske nøkler. Det finnes en løsning på dette problemet; asymmetrisk kryptografi, eller kryptosystemer med offentlig nøkkel som det også kalles. I asymmetrisk kryptografi har alle vanligvis en privat nøkkel som man holder strengt hemmelig, og en offentlig nøkkel som man deler med dem man skal kommunisere med. Ved hjelp av disse, kan man få en delt hemmelighet som kan benyttes i symmetrisk kryptografi. Asymmetrisk kryptografi benyttes også til autentisering, ved hjelp av såkalte *digitale signaturer*. Protokoller som benyttes mye er Diffie-Hellman, RSA og ElGamal. Elliptiske kurver har vist seg å være svært anvendelige i asymmetrisk kryptografi, de viktigste protokollene i asymmetrisk kryptografi kan tilpasses til å benytte elliptiske kurver.

3.2 Elliptiske kurver

Siden Koblitz og Miller uavhengig av hverandre foreslo å benytte elliptiske kurver i kryptosystemer med offentlig nøkkel i 1985, har man utviklet stadig mer effektive implementasjoner av kryptosystemer basert på elliptiske kurver. I dag er slike systemer like raske som vanlige heltallsfaktoriseringsbaserte systemer med samme nøkkellengde, se [2]. Siden systemer basert på elliptiske kurver gir samme sikkerhet som RSA med kortere nøkler er de mer effektive og vil derfor ofte kunne erstatte RSA. Den er dermed mer effektiv og derfor særlig velegnet til bruk i smartkort, mobiltelefoner og andre ytelsessvake systemer. Som en følge av dette er interessen for denne metoden økende, og mange viktige kryptografiske protokoller basert på offentlig nøkkel kan i dag benytte elliptiske kurver.

Punkter på en elliptisk kurve danner en abelsk gruppe, med gruppeoperasjonen addisjon.

Multiplikasjon av et punkt på en elliptisk kurve med et heltall, det vil si å addere punktet med seg selv flere ganger, er grunnlaget for bruk av elliptiske kurver i kryptografi.

Alle elliptiske kurver kan framstilles på såkalt *lang Weierstrassform*:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

Under noen forutsetninger kan man forenkle ligningen for den elliptiske kurven og skrive den på *kort Weierstrassform*:

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

Under andre forutsetninger kan de skrives på *Hesseform*:

$$x^3 + y^3 + 1 = Dxy$$

Hvilken form man benytter påvirker ytelsen til operasjonene i kryptosystemer, fordi de forskjellige representasjonsformene har forskjellige formler for punktmultiplikasjon.

Senere forskning tyder på at kurver på Hesseform har en rekke egenskaper som gjør dem velegnet for bruk i kryptografi, blant annet ytelsesmessig, men også ved at det er lettere å beskytte seg mot en type sidekanalangrep. Vi vil konsentrere oss om å utføre ytelsestester hvor vi sammenligner kort Weierstrassform og Hesseform.

3.3 Forskningsstatus på området i dag

Det er skrevet forholdsvis lite om Hesseform av elliptiske kurver; vi har funnet at følgende artikler har vært av særlig interesse under arbeidet med oppgaven:

Marc Joye, Jean-Jacques Quisquater: Hessian Elliptic Curves and Side-Channel Attacks, [3]: I tillegg til å ta for seg sidekanalangrep sier denne artikkelen en del generelt om Hesseform, om konvertering av kurve og punkter mellom Weierstrassform og Hesseform.

Hege R. Frium: The Group Law on Elliptic Curves on Hesse form, [4]: Dette artikkelen tar for seg matematisk teori rundt elliptiske kurver på Hesseform, med særlig vekt på gruppeloven i projektiv Hesseform, og torsjonspunkter.

N. P. Smart: The Hessian Form of an Elliptic Curve, [5]: Dette er en mer implementasjonsrettet vinkling på Hesseform av elliptiske kurver og den tilhørende gruppeloven, med vekt på ytelsesaspekter ved bruk av Hesseform.

N. P. Smart, E. J. Westwood: Point Multiplication on Ordinary Elliptic Curves over Fields of Characteristic Three, [6]: Her beskrives bruk av Hesseform på elliptiske kurver over endelige kropper med karakteristikk 3.

3.4 Rapportens struktur

I kapittel 4 gir vi en kort oversikt over noen kryptografiske begreper og protokoller, og viser eksempler på hvordan kryptografi med elliptiske kurver kan brukes. En kort introduksjon til elliptiske kurver finnes i kapittel 5. Kapittel 6 handler om Weierstrassform av elliptiske kurver, og kapittel 7 om Hesseform. Disse to kapitlene inneholder beskrivelse av vår implementasjon av gruppeoperasjonene, og annet som trengs for å kunne utføre tester hvor man sammenlikner de to formene. Resultatet fra disse testene, og drøfting av dem, er presentert i kapittel 8. Vi avslutter rapporten med konklusjon i kapittel 9.

Kapittel 4

Introduksjon til kryptografi

4.1 Innledning

Dette kapittelet vil gi en introduksjon til kryptografi, hvor vi spesielt vil sette fokus på de delene av kryptografien hvor elliptiske kurver benyttes, nemlig asymmetrisk kryptografi.

Vi vil ta for oss TLS (Transport Layer Security)-protokollen og se særlig på de delene av denne hvor det kan være naturlig å benytte kryptografi med elliptiske kurver. ECC (Elliptic Curve Cryptography) Cipher Suites for TLS, se [7], beskriver hvordan man kan benytte ECC i TLS-protokollen. OpenSSL-biblioteket er et kryptografibibliotek som inneholder støtte for både TLS og SSL (Secure Socket layer)-protokollene, og nyere utgaver av dette biblioteket inneholder støtte for ECC donert av Sun.

Vi vil avslutte kapittelet med å vise et eksempel på bruk av elliptiske kurver i et kryptosystem.

4.2 Asymmetrisk kryptografi

Diffie og Hellman fikk en banebrytende idé i 1976, nemlig at man man benytte to nøkler, en nøkkel til kryptering, og en annen nøkkel til dekryptering. Nøkkelen til dekryptering må være hemmelig, men krypteringsnøkkelen kan være offentlig. For eksempel kan den som har den offentlige nøkkelen til Alice, benytte den til å kryptere en melding slik at kun Alice kan lese den. Den kan bare dekrypteres med den hemmelige private nøkkelen til Alice. Kryptosystemer med offentlig nøkkel kan benyttes både til nøkkelutveksling, kryptering, og til signering av meldinger.

Siden symmetrisk kryptografi er langt mer effektivt enn asymmetrisk, vil man sjelden kryptere lange meldinger, men heller kryptere og oversende en sesjonsnøkkel for en symmetrisk algoritme og benytte denne i videre kommunikasjon.

4.2.1 Matematisk bakgrunn

Det finnes matematiske operasjoner som er enkle å utføre, men vanskelig eller tidkrevende å reversere. Slike funksjoner kalles enveisfunksjoner. En variant av

slike enveisfunksjoner kalles trapdoor-enveisfunksjoner, disse fungerer slik at det med en tilleggsopplysning (den private nøkkelen) blir gjennomførbart å reversere dem. De fleste av de asymmetriske algoritmene baserer seg på slike funksjoner, se [1] side 9. Det er for eksempel enkelt å multiplisere to store primtall, men mye mer tidkrevende å faktorisere produktet. RSA er en av de mest brukte protokoll innen asymmetrisk kryptografi, og den baserer sin sikkerhet på dette problemet. Diffie-Hellman og ElGamal er to andre algoritmer som benyttes mye, og de baserer seg begge på at det er vanskelig å beregne diskrete logaritmer i en endelig kropp.

4.2.2 Nøkkelutveksling

For å kunne benytte symmetriske kryptoalgoritmer må begge de kommuniserende partene ha en hemmelig nøkkel. Før kryptosystemer med offentlig nøkkel ble oppfunnet, var nøkkeldistribusjon et stort problem. Diffie-Hellman, ElGamal og RSA er protokoller som benyttes mye til nøkkelutveksling, og i de to siste tilfellene også til kryptering og signering.

4.2.3 Diffie-Hellman

Diffie-Hellman er en nøkkelutvekslingsprotokoll som kan brukes til å generere en delt hemmelighet, men ikke til å kryptere og dekryptere meldinger. Som nevnt i avsnitt 4.2.1 baserer den sin sikkerhet på vanskeligheten av å beregne diskrete logaritmer i endelige kropper.

Alice og Bob må først bli enige om et stort primtall p, og et annet stort primtall g. Hverken g eller p trenger å være hemmelige. Alice velger et tilfeldig stort heltall $x_A \in \{1, \ldots, p-1\}$ som sin private nøkkel, og sender X_A til Bob hvor

$$X_A = g^{x_A} \pmod{p}$$

Bob velger et tilfeldig stort heltall $x_B \in \{1, \ldots, p-1\}$ som sin private nøkkel, og sender X_B til Alice hvor

$$X_B = g^{x_B} \pmod{p}$$

Alice beregner

$$k = X_B^{x_A} \pmod{p}$$

Bob beregner

$$k' = X_A^{x_B} \pmod{p}$$

Alice og Bob har nå en delt hemmelighet $k = k' = g^{x_A x_B} \pmod{p}$ som kan benyttes som hemmelig nøkkel til en symmetrisk krypteringsalgoritme. Det eneste en som avlytter kanalen får tak i er p, g, X_A og X_B; og det er ikke nok informasjon til å finne k. Denne algoritmebeskrivelsen er hentet fra [8].

4.2.4 ElGamal

For å generere et nøkkelpar, velger man et primtall p, og to tilfeldige tall $g \in \{1, \ldots, p-1\}$ og $x \in \{1, \ldots, p-1\}$. Man beregner $y = g^x \pmod{p}$. Privat nøkkel er x, mens y, g og p er offentlig nøkkel. Flere brukere kan benytte samme g og p, se [8].

For å kryptere meldingen M (dette kan for eksempel være en nøkkel til en symmetrisk kryptoalgoritme for bruk i senere kommunikasjon), velger Alice et tilfeldig heltall k som er relativt prim til $p-1$. Hun beregner

$$a = g^k \pmod{p}$$

$$b = y^k M \pmod{p}$$

Kryptoteksten som består av a og b oversendes til Bob. Bob finner M ved å beregne

$$M = ba^{-x} \pmod{p}.$$

4.2.5 Hash-funksjoner

En hash-funksjon defineres slik i [1]: *"En hash-funksjon er en beregningsmessig effektiv funksjon som avbilder binære sekvenser av vilkårlig lengde til binære sekvenser av en fast lengde, som kalles hash-verdier".*

Dette "sammendraget" kan benyttes til å sjekke at en melding ikke har blitt endret, siden det er svært usannsynlig at den endres til en annen melding med samme hash-verdi. Dersom sammendraget for eksempel har lengde 128 bit, har man 2^{128} forskjellige mulige sammendrag, og sannsynligheten for at to meldinger gir samme sammendrag er dermed ca. $1/2^{128}$, se [9]. En kryptografisk egnet hash-funksjonen må også være vanskelig å reversere, slik at man ikke lett kan finne en falsk melding som gir samme hash-verdi som den opprinnelige meldingen. MD5 og SHA-1 er to hash-funksjoner som er mye brukt i kryptografi.

MAC (message autentication code) er en nøkkelavhengig enveishash-funksjon, det vil si at kun den som har en nøkkel kan verifisere verdien. MAC kan benyttes til å sikre mot at både melding og hashverdi endres siden man må kjenne nøkkelen for å generere en ny hashverdi, se [8].

4.2.6 Digital signatur

For å garantere autentisitet, kan man *signere* en melding ved å kryptere den med sin egen private nøkkel. Man får da en kryptert utgave av meldingen som kan legges ved meldingen selv. Andre kan da med signererens offentlige nøkkel sjekke autentisiteten av signaturen. Dersom meldingen har blitt forandret vil dette detekteres. Dette kan ofte være upraktisk siden meldingens lengde vil dobles, og man benytter ofte en sikker hash-funksjon og signerer et sammendrag av meldingen istedenfor hele meldingen. Siden en sikker hash-funksjon benyttes, er man sikker på at det den signerte informasjonen ikke kan endres uten at det oppdages. Digital signatur har mange bruksområder, det brukes både til å signere email, viktige dokumenter, og også til programvare for å være sikker på at det ikke er en modifisert versjon, for eksempel med virus eller trojanske hester.

4.2.7 Digitalt sertifikat

Ved distribusjon av offentlige nøkler må mottakeren kunne være sikker på at den offentlige nøkkelen han har fått er ekte, men den kan ikke signeres på vanlig måte, siden man da hadde trengt en sikker offentlig nøkkel fra avsenderen i utgangspunktet for å kunne verifisere signaturen.

En mulig løsning på dette problemet er å benytte digitale sertifikater. Et digitalt sertifikat er et signert dokument som bekrefter at dokumentets offentlige nøkkel tilhører personen hvis navn står på sertifikatet. Dette dokumentet vil vanligvis være signert av en tredjepart som alle involverte parter stoler på. Det er viktig å merke seg at man må allerede ha den offentlige nøkkelen til utstederen av sertifikatet for å kunne verifisere at det er ekte. Den pålitelige tredjeparten kan også ha et digitalt sertifikat, men det vil uansett alltid være en øverste sertifiseringsautoritet (CA) i en slik kjede som man må skaffe offentlig nøkkel fra på andre måter, for eksempel ved at de følger med en nettleser.

4.3 Kryptografi basert på elliptiske kurver

Kryptografi basert på elliptiske kurver er en metode som utnytter det at punkter på en elliptisk kurve danner en additiv abelsk gruppe som har visse egenskaper som gjør den velegnet til bruk i asymmetrisk kryptografi. Hovedpoenget er at det er enkelt å multiplisere et punkt med et stort heltall, det vil si å legge punktet sammen med seg selv flere ganger. Å reversere denne operasjonen er derimot svært vanskelig når man opererer med punkter på elliptiske kurver over endelige kropper som har karakteristikk med en stor primtallsfaktor. Dette problemet kalles ofte *elliptisk kurve diskret logaritme problemet*.

4.3.1 Elliptic Curve Diffie-Hellman

ECDH (Elliptic Curve Diffie-Hellman) er en algoritme som fungerer tilsvarende klassisk Diffie-Hellman, men istedenfor potensregning i en endelig kropp, vil "enveisfunksjonen" være punktmultiplikasjon på en elliptisk kurve over en endelig kropp, og den inverse operasjonen har vanskelighetsgrad tilsvarende *diskret logaritme problemet*.

Alice og Bob blir enig om en endelig kropp, en elliptisk kurve over denne, og om et tilfeldig punkt på kurven; $\mathbf{P}$.

Alice finner et tilfeldig heltall x_A, og sender $\mathbf{P_A}$ til Bob hvor

$$\mathbf{P_A} = [x_A]\mathbf{P}$$

Denne notasjonen betyr at punktet $\mathbf{P}$ på den elliptiske kurven multipliseres med heltallet x_A, dette gjøres for å skille punktmultiplikasjon fra vanlig multiplikasjon.

Bob finner et tilfeldig heltall x_B, og sender $\mathbf{P_B}$ til Alice hvor

$$\mathbf{P_B} = [x_B]\mathbf{P}$$

Nå kan begge beregne den delte hemmeligheten, punktet $\mathbf{P_k}$:

$$\mathbf{P_k} = [x_A]([x_B]\mathbf{P}) = [x_B]([x_A]\mathbf{P})$$

Man vil kun benytte x-verdien i punktet, siden y-verdien er en funksjon av denne, se [2].

4.3.2 Elliptic Curve ElGamal

Alice og Bob blir enig om en endelig kropp, en elliptisk kurve over denne, og om et tilfeldig punkt på kurven; $\mathbf{P}$. Alice finner et tilfeldig heltall x_A, og sender $\mathbf{P_A}$ til Bob hvor

$$\mathbf{P_A} = [x_A]\mathbf{P}$$

Bob finner et tilfeldig heltall x_B, og sender $\mathbf{P_B}$ til Alice hvor

$$\mathbf{P_B} = [x_B]\mathbf{P}$$

Hittil er dette akkurat som i ECDH. Når Alice skal kryptere en melding M, må denne først representeres med et punkt $\mathbf{P_M}$ på den elliptiske kurven.

For å kunne representere meldingen som et punkt må man finne et punkt hvor x-verdien inneholder meldingen. Siden ikke alle mulige x-verdier oppfyller ligningen til kurven, må man reservere noen bit i punktets x-verdi. Punkter på en kurve over en endelig kropp vil være nesten uniformt fordelt. Dersom man representerer punktet med flere bit enn meldingens bitlengde, vil man kunne variere de reserverte bitene til man finner en x-verdi som tilfredstiller ligningen til kurven. Etter et par forsøk vil man nesten helt sikkert finne et punkt som representerer meldingen. Denne prosedyren forklares nærmere i kapittel 6.4 i [2].

Deretter velger Alice et tilfeldig tall r, og beregner to punkter;

$$\mathbf{P_r} = [r]\mathbf{P}$$

og

$$\mathbf{P_h} = \mathbf{P_m} + [r]\mathbf{P_B}$$

Hun sender $\mathbf{P_r}$ og $\mathbf{P_h}$ til Bob.

Bob beregner

$$\mathbf{P_s} = [x_B]\mathbf{P_r}$$

og finner meldingspunktet $\mathbf{P_m}$

$$\mathbf{P_m} = \mathbf{P_h} - \mathbf{P_s}$$

Vi ser at dette virker ved å sette inn for $\mathbf{P_h}$ og $\mathbf{P_s}$ fra tidligere ligninger

$$\begin{aligned}
\mathbf{P_m} &= \mathbf{P_h} - \mathbf{P_s} \\
\mathbf{P_m} &= \mathbf{P_h} - [x_B]([r]\mathbf{P}) \\
\mathbf{P_m} &= \mathbf{P_m} + [r]([x_B]\mathbf{P}) - [x_B]([r]\mathbf{P}) \\
\mathbf{P_m} &= \mathbf{P_m}.
\end{aligned}$$

4.4 Transport Layer Security og Secure Socket Layer

TLS, se [10], er en IETF-standard som er en bakoverkompatibel videreføring av Netscapes SSL. TLS er en transportslagprotokoll som ligger på toppen av TCP eller en annen pålitelig transportlagsprotokoll, og sørger for at forbindelsen er konfidensiell og pålitelig. Integritetssjekk av meldingene foregår ved hjelp av

sikre enveishashfunksjoner (f. eks. SHA, MD5, e.l.), og meldingene som overføres holdes konfidensielle ved hjelp av symmetriske krypteringsalgoritmer (f. eks. DES, RC4, e.l.). Sesjonsnøkkel for denne kryptreringen skaffes ved hjelp av asymmetriske kryptoalgoritmer (f. eks. RSA, e.l.). Disse asymmetriske kryptoalgoritmene kan erstattes med varianter basert på elliptiske kurver.

TLS består av flere underprotokoller; record og handshake er de to viktigste. Handshake-protokollen sørger for eventuell autentisering av partene, og forhandling om algoritmer for kryptering og integritetssikring, nøkkellengder og andre parametere til record-protokollen.

Meldinger overføres med record-protokollen som ligger oppå transportlaget (vanligvis TCP) og sørger for konfidensialitet og integritet av de overførte meldingene. Denne kan benyttes av applikasjonsprotokollen på samme måte som man ville benyttet TCP for å overføre meldinger.

OpenSSL er et bibliotek med åpen kildekode som leverer kryptografiske funksjoner blant annet til bruk i SSL og TLS til en lang rekke andre programmer.

HTTPS er egentlig vanlig HTTP (Hyper Text Transfer Protocol) over TLS, se [11], eller SSL, men oftest på en annen TCP/IP-port enn vanlig, vanligvis 443.

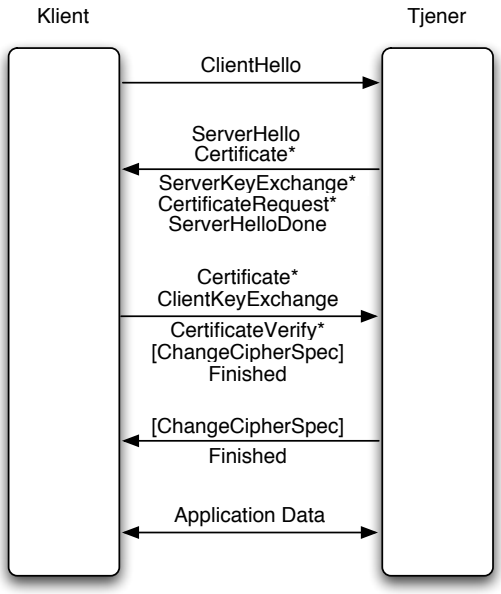
4.5 Eksempel på bruk av TLS med ECC

Alice ønsker å kjøpe en bok i Bobs nettbokhandel. Bobs nettbokhandel har støtte for sikker netthandel, den har en sikker webtjener som støtter HTTPS protokollen, dvs. HTTP over TLS. Han benytter en implementasjon av TLS som støtter ECC, basert på beskrivelsen i [7]. Nettleseren til Alice støtter også TLS med ECC.

Idet Alice legger boken i den virtuelle handlekurven og klikker på linken merket checkout, vil nettleseren hennes initiere en sikker HTTP-forbindelse. Dette gjøres med handshake-protokollen, som vist i Figur 4.5. Nettleseren sender en hello-melding med protokollversjon, et tilfeldig tall på 28 bytes for å sikre at meldingen er unik, en sesjonsidentifikator (tom siden dette er en ny sesjon og ikke en videreføring av en tidligere sesjon), og en liste med forslag til sett med kryptografiske innstillinger. Hvert element i denne listen har et sett med algoritmer for nøkkelutveksling, datakryptering, og MAC. Alices nettleser foreslår primært TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA, men er villig til å ta til takke med TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA eller i verste fall TLS_ECDH_ECDSA_WITH_RC4_128_SHA. Alle disse har ECDH_ECDSA som nøkkelutvekslingsalgoritme, det vil si Diffie-Hellman nøkkelutveksling basert på elliptisk kurve, med ECDSA-sertifikat. SHA er valgt for meldingsautentisitetskontroll, mens datakrypteringsalgoritmen kan velges mellom AES_256_CBC, AES_128_CBC og RC4_128.

Tjeneren svarer med en hello-melding hvor det går fram at algoritmesettet TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA er det beste den kan klare. Siden man har blitt enig om å benytte ECDH_ECDSA til nøkkelutveksling, sender den også sertifikatet sitt, som inneholder den offentlige nøkkelen til Bobs nettbokhandel, signert av Trent, som er en sertifiseringsautoritet som begge stoler på.

Alice validerer Bobs sertifikat, ekstraherer Bobs offentlige nøkkel fra det, og sjekker at den er egnet til ECDH.



Figur 4.1: Denne figuren viser meldingsflyten i TLS-handshakeprotokollen. Meldingene merket med * vil ikke alltid sendes, avhengig av algoritmevalg.

Alices nettleser genererer et ECDH-nøkkelpar på samme kurve som Bobs offentlige nøkkel, og sender sin offentlige nøkkel til Bob i en KeyExchange-melding

Bob verifiserer at Alices offentlige nøkkel er på samme elliptiske kurve som Bobs.

De utfører en ECDH-operasjon, og benytter den delte hemmeligheten som *premaster secret*. Premaster secret er grunnlaget for sesjonsnøkkel, initialiseringsvektor, etc.

Nå benytter begge underprotokollen *change cipher spec protocol* for å signalisere til record-protokollen at den skal ta i bruk de framforhandlede algoritmer og nøkler, og sender deretter en finished-melding til hverandre for å signalisere at de er ferdig med handshake-prosessen.

Så er det klart for konfidensiell og integritetskontrollert dataoverføring med record-protokollen: Nettleseren sender informasjon om boken Alice vil bestille til record-protokollen som krypterer med AES med 128 bits nøkkel, genererer enveishash med SHA, og sender den via TCP-protokollen til Bobs datamaskin hvor den tas imot av record-protokollen, og dekrypteres og autentisitetssjekkes. Deretter sendes den til den sikre webtjeneren, som behandler den, og sender tilbake et bestillingsskjema til Alices nettleser med samme prosedyre. Alice skriver inn navn, adresse og kredittkortnummer, og klikker submit-linken. Dermed sendes informasjonen som Alice har skrevet inn til Bobs nettbokhandel, konfidensiell og integritetssjekket. Bobs nettbokhandel sender en kvittering tilbake til Alice, og dermed er transaksjonen over.

Kapittel 5

Introduksjon til elliptiske kurver

5.1 Innledning

I dette kapittelet vil vi ta for oss den matematiske bakgrunnen for elliptiske kurver, med spesiell vekt på gruppelov for punkter på elliptiske kurver. Vi vil behandle både Weierstrassform og Hesseform, siden vi senere vil utføre tester på kurver på disse formene.

5.2 Elliptiske kurver på Weierstrassform

La $\mathbb{K}$ være en kropp, $\overline{\mathbb{K}}$ dens algebraiske tillukning[1] og $\mathbb{K}^*$ dens multiplikative gruppe. En elliptisk kurve over $\mathbb{K}$ er definert som mengden av løsninger i det projektive[2] planet $\mathbb{P}^2(\overline{\mathbb{K}})$ av den homogene Weierstrassligningen på formen

$$E : y^2z + a_1xyz + a_3yz^2 = x^3 + a_2x^2z + a_4xz^2 + a_6z^3 \tag{5.1}$$

hvor $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$. Denne ligningen kalles også for **lang Weierstrassform**. Kurven må ikke være singulær. Kurven har bare ett punkt hvor z-koordinaten er null, nemlig punktet $(0 : 1 : 0)$. Dette kalles punktet i uendelig, og skrives $\mathcal{O} = (0 : 1 : 0)$.

Med $E(\mathbb{K})$ mener vi mengden av klasser $(x : y : z) \in \mathbb{K}^3/\{(0 : 0 : 0)\}$ som tilfredstiller ligningen for den elliptiske kurven E. Ordenen til kurven $\#E(\mathbb{K})$ er antall punkter i mengden $E(\mathbb{K})$.

Definisjon 1. *En elliptisk kurve over $\mathbb{K}$ er en plan ikke-singulær kubisk kurve med minst ett $\mathbb{K}$-rasjonalt[3] punkt $\mathcal{O}$.*

I kapittel 3 i [13] er det en mer grundig beskrivelse av elliptiske kurver.

[1] Se teorem 31.17 i [12] for mer informasjon om algebraiske tillukninger.
[2] Det projektive planet beskrives i avsnitt 5.3
[3] At et punkt er $\mathbb{K}$-rasjonalt betyr at koordinatene er elementer i $\mathbb{K}$.

5.2.1 Kort Weierstrassform

Dersom karakteristikken til $\mathbb{K}$ er ulik 2 og 3, så kan vi gjøre følgende variabelskifte:

$$x = x' - \frac{b_2}{12} \tag{5.2a}$$

$$y = y' - \frac{a_1}{2}(x' - \frac{b_2}{12}) - \frac{a_3}{2} \tag{5.2b}$$

$$b_2 = a_1^2 + 4a_2 \tag{5.2c}$$

Variabelskiftet transformerer ligningen fra lang Weierstrassform til en isomorf[4] kurve på kort Weierstrassform.

$$E : y^2 = x^3 + ax + b \tag{5.3}$$

Dersom en kurve skal være en elliptisk kurve må den som tidligere nevnt ikke være singulær, det vil si at det ikke må finnes et punkt på kurven hvor alle de partiellderiverte er null. Dette kan vi sjekke ved å beregne diskriminanten til E.Vi definerer diskriminanten til kurven E på følgende måte:

$$\Delta = -16(4a^3 + 27b^2). \tag{5.4}$$

Teorem 1. *Kurven gitt ved likning 5.3 er ikke-singulær, hvis og bare hvis diskriminanten* $\Delta \neq 0$.

Bevis. Se [13] side 31. □

Dersom $\Delta \neq 0$ kan vi introdusere j-invarianten til kurven E:

$$j(E) = -\frac{1728(4a)^3}{\Delta} \tag{5.5}$$

Teorem 2. *To elliptiske kurver som er isomorfe over* $\mathbb{K}$ *har samme* j*-invariant, og motsatt to elliptiske kurver som har samme* j*-invariant er isomorfe over den algebraiske tillukningen* $\overline{\mathbb{K}}$.

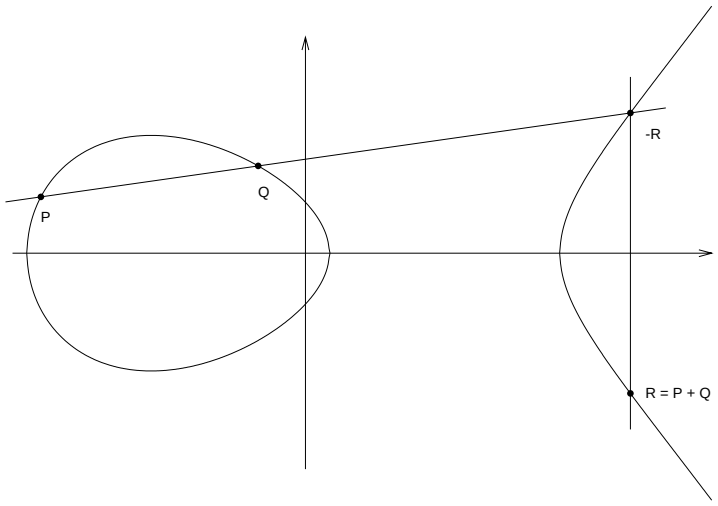
Bevis. Se lemma III.1 i [13], side 31. □

To kurver på kort Weierstrassform kan være isomorfe selv om de har forskjellige a- og b-parametere.

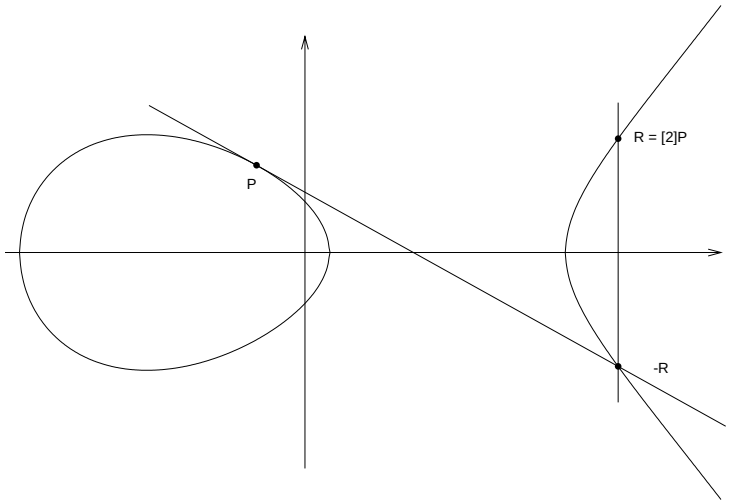
5.2.2 Gruppeloven

La E være en elliptisk kurve over kroppen $\mathbb{K}$, og punktene $\mathbf{P}$ og $\mathbf{Q}$ være to forskjellige $\mathbb{K}$-rasjonale punkter på kurven E. Da må linjen som går gjennom punktene $\mathbf{P}$ og $\mathbf{Q}$ krysse kurven E i et tredje punkt $-\mathbf{R}$, siden E er en kubisk kurve. Punktet $-\mathbf{R}$ vil også være $\mathbb{K}$-rasjonalt siden linjen, kurven og punktene selv er definert over $\mathbb{K}$. Speiler vi $-\mathbf{R}$ om x-aksen får vi et annet $\mathbb{K}$-rasjonalt punkt som vi kaller $\mathbf{R} = \mathbf{P} + \mathbf{Q}$. Figur 5.1 viser et eksempel på addisjon av to forskjellige punkter $\mathbf{P}$ og $\mathbf{Q}$ på kurven $y^2 = x^3 - 8x + 2$ over $\mathbb{R}$.

Når man skal doble $\mathbf{P}$, tar man utgangspunkt i tangenten til i kurven E i punktet $\mathbf{P}$. Tangenten må krysse kurven E i eksakt ett annet punkt $\mathbf{R}$, siden E er en kubisk kurve. Igjen speiler vi $\mathbf{R}$ om x-aksen og kaller dette punktet for



Figur 5.1: Addisjon av to punkter på en elliptisk kurve $y^2 = x^3 - 8x + 2$



Figur 5.2: Dobling av et punkt på en elliptisk kurve $y^2 = x^3 - 8x + 2$

$\mathbf{P}+\mathbf{P}$ eller [2]$\mathbf{P}$. Figur 5.2 viser et eksempel på dobling av punktet $\mathbf{P}$ på kurven $y^2 = x^3 - 8x + 2$ over $\mathbb{R}$.

Det kan vises at punktene $E(\mathbb{K})$ på en elliptisk kurve danner en abelsk gruppe hvor $\mathcal{O}$ er identitetselement, og gruppeoperasjonen er prosessen som er beskrevet over, se [13] side 32. Vi betegner gruppeoperasjonen med $+$.

Vi vil nå kort forklare hvordan gruppeloven for elliptiske kurver på kort Weierstrassform fremkommer. Vi antar at $\mathbb{K} = \mathbb{F}_p$, og at p er et primtall større enn 3.

La $\mathbf{P} = (x_0, y_0)$ og $\mathbf{Q} = (x_1, y_1)$ være to forskjellige punkter på den elliptiske kurven $E : y^2 = x^3 + ax + b$, da vil linjen $L : y - y_0 = \lambda(x - x_0)$ mellom punktene $\mathbf{P}$ og $\mathbf{Q}$ krysse kurven i et tredje punkt $-\mathbf{R}$, ifølge korde-tangent-loven. Stigningstallet for linjen L vil være $\lambda = \frac{x_1 - x_0}{y_1 - y_0}$, og koordinatene til punktet $\mathbf{R}$ kaller vi x_2 og y_2. Dersom vi kombinerer ligningen E for kurven og ligningen L for linjen, vil vi få en ny tredjegradslikning som vi kan bruke til å finne det tredje krysningspunktet.

Vi løser L med hensyn på y, og får:

$$L : y = \lambda(x - x_0) + y_0 \tag{5.6a}$$

Deretter beregner vi y^2 og setter dette inn i E:

$$y^2 = \lambda^2 x^2 - 2\lambda^2 x_0 x + 2xy_0\lambda - 2x_0 y_0 \lambda + y_0^2 \tag{5.6b}$$

Vi setter nå sammen T og E og får:

$$x^3 - \lambda^2 x^2 + (a + 2\lambda^2 x_0 - 2y_0\lambda)x + b - \lambda^2 x_0^2 + 2x_0 y_0 \lambda - y_0^2 = 0 \tag{5.6c}$$

Vi har nå en tredjegradslikning, vi kjenner også to av løsningene (x_0, x_1). Derfor foretar vi to polynomdivisjoner, hvor vi først deler ligningen på $x - x_0$, og deretter deler vi ligningen på $x - x_1$. Vi skal da ende opp med en likning med en løsning x_2.

Vi deler likning 5.6c på $x - x_0$, og får følgende likning:

$$x^2 + (x_0 - \lambda^2)x + (a + x_0^2 + x_0\lambda^2 - 2y_0\lambda) = 0 \tag{5.6d}$$

Videre deler vi likning 5.6d på $x - x_1$, og får:

$$x + (x_0 + x_1 - \lambda^2) = 0 \tag{5.6e}$$

Vi løser likning 5.6e med hensyn på x, og setter deretter inn i ligningen for linjen L for å finne y-verdien til krysningspunktet mellom linjen og kurven. Vi må skifte fortegn på y-verdien for å få $\mathbf{R}$ og ikke $-\mathbf{R}$.

$$x_2 = \lambda^2 - x_0 - x_1 \tag{5.7a}$$

$$y_2 = \lambda(x_0 - x_2) - y_0 \tag{5.7b}$$

Vi har nå vist hvordan gruppeloven fungerer dersom $\mathbf{P} \neq \mathbf{Q}$, dersom $\mathbf{P} = \mathbf{Q}$ må man bruke tangenten i punktet $\mathbf{P}$, istedenfor linjen gjennom punktene $\mathbf{P}$ og $\mathbf{Q}$.

[4] To elliptiske kurver er isomorfe hvis man kan transformere den ene til den andre ved hjelp av et koordinatskifte.

5.2.3 Formler for addisjon og dobling

La $\mathbf{P} = (x_0, y_0)$, $\mathbf{Q} = (x_1, y_1)$ være to punkter på den elliptiske kurven $E : y^2 = x^3 + ax + b$, da vil:

$$-\mathbf{P} = (x_0, -y_0) \tag{5.8}$$

Når $x_0 \neq x_1$ setter vi:

$$\lambda = \frac{y_1 - y_0}{x_0 - x_1}, \tag{5.9}$$

og når $x_0 = x_1$, $y_0 \neq 0$ setter vi:

$$\lambda = \frac{3x_0^2 + a}{2y_0}. \tag{5.10}$$

Hvis

$$\mathbf{R} = (x_2, y_2) = \mathbf{P} + \mathbf{Q} \neq \mathcal{O}$$

så er x_2 og y_2 gitt ved formlene:

$$x_2 = \lambda^2 - x_0 - x_1 \tag{5.11a}$$

$$y_2 = \lambda(x_0 - x_2) - y_0 \tag{5.11b}$$

Dette avsnittet er basert på avsnitt III.3.1 i [13].

5.3 Projektive koordinater

Ved bruk av projektive koordinater er det mulig å utføre punktaddisjon og dobling, uten å dividere. Divisjon over kropper er som oftest mye mer tidkrevende enn multiplikasjon, det lønner seg derfor som oftest å bruke projektive koordinater, selv om det trengs flere addisjoner og multiplikasjoner enn ved bruk av affine[5] koordinater.

Med det *projektive planet* $\mathbb{P}_2(\mathbb{K})$ mener vi mengden av ekvivalensklasser av talltripler $(x, y, z) \in \mathbb{K}^3/\{(0,0,0)\}$ under ekvivalensrelasjonen $(x_1, y_1, z_1) \sim (x_2, y_2, z_2)$ hvis og bare hvis $(x_2 : y_2 : z_2) = (ax_1 : ay_1 : az_1),\ a \neq 0 \in \mathbb{K}$.

Erstatter vi x med x/z og y med y/z i likning 5.3 får vi den homogene Weierstrassformen.

$$y^2 z = x^3 + axz^2 + bz^3 \tag{5.12}$$

Homogen betyr at summen av eksponentene til x, y og z i hvert ledd i formelen er den samme, i dette tilfelle er det 3. Dette er grunnen til denne formen for projektive koordinater ofte omtales som **projektive homogene koordinater**. Det er også vanlig å bruke kolon i stedet for komma til å skille koordinatene i et projektivt punkt.

Følgende forhold gjelder mellom projektive homogene koordinater og affine koordinater. Det affine punktet (x, y) representeres av det projektive punktet $(ax : ay : a)$, og motsatt; det projektive punktet $(x : y : z)$ tilsvarer det affine punktet $(x/z, y/z)$.

[5] Med det affine planet mener vi mengden av alle tallpar $(x, y) \in \mathbb{K}^2$

5.4 Jacobiske projektive koordinater

Det finnes andre projektive representasjoner som fører til mer effektive implementasjoner av gruppeoperasjonen for kort Weierstrassform. Vi vil derfor introdusere *Jacobiske projektive koordinater*, med følgende ekvivalensrelasjon $(x_1 : y_1 : z_1) \sim (x_2 : y_2 : z_2)$ hvis og bare hvis $(x_2 : y_2 : z_2) = (a^2 x_1 : a^3 y_1 : a z_1),\ a \neq 0 \in \mathbb{K}$.

Det affine punktet (x, y) representeres av det projektive punktet $(a^2 x : a^3 y : a)$, og motsatt; det projektive punktet $(x : y : z)$ tilsvarer det affine punktet $(x/z^2, y/z^3)$. Erstatter vi x med x/z^2 og y med y/z^3 i likning 5.3 får vi en vektet Weierstrasslikning på formen

$$y^2 z = x^3 + axz^4 + bz^6 \tag{5.13}$$

Vi vil nå vise hvordan addisjonsformelen for kort Weierstrassform kan gjøres om til Jacobiske projektive koordinater. Vi tar utgangspunkt i likning 5.11a og likning 5.11b for addisjon av to punkter på kort Weierstrassform, og erstatter alle x-er med x/z^2 og alle y-er med y/z^3.

$$\lambda = \frac{y_2/z_2^3 - y_1/z_1^3}{x_2/z_2^2 - x_1/z_1^2} = \frac{z_1^3 y_2 - z_2^3 y_1}{x_2 z_1^3 z_2 - x_1 z_1 z_2^3} \tag{5.14}$$

Vi lager noen nye hjelpevariabler:

$$\begin{aligned}
\lambda_1 &= x_1 z_2^2 \\
\lambda_2 &= x_2 z_1^2 \\
\lambda_3 &= \lambda_1 - \lambda_2 \\
\lambda_4 &= y_1 z_2^3 \\
\lambda_5 &= y_2 z_1^3 \\
\lambda_6 &= \lambda_4 - \lambda_5
\end{aligned}$$

Vi kan nå skrive:

$$\lambda = \frac{\lambda_4 - \lambda_5}{z_1 z_2 (\lambda_2 - \lambda_1)} = -\frac{\lambda_6}{z_1 z_2 \lambda_3} \tag{5.15}$$

Vi velger:

$$z_3 = z_1 z_2 \lambda_3 \tag{5.16}$$

$$\begin{aligned}
\frac{x_3}{z_3^2} &= \left(\lambda^2 - \frac{x_1}{z_1^2} - \frac{x_2}{z_2^2}\right) \\
x_3 &= \left(-\frac{\lambda_6}{z_1 z_2 \lambda_3}\right)^2 (z_1 z_2 \lambda_3)^2 - \left(\frac{x_1}{z_1^2} z_1^2 z_2^2 \lambda_3^2 + \frac{x_2}{z_2^2} z_1^2 z_2^2 \lambda_3^2\right) \\
&= \lambda_6^2 - (x_1 z_2^2 + x_2 z_1^2)\lambda_3^2
\end{aligned}$$

Vi definerer hjelpevariabelen:

$$\lambda_7 = \lambda_1 + \lambda_2$$

Vi får dermed:

$$x_3 = \lambda_6^2 - \lambda_7 \lambda_3^2 \tag{5.17}$$

Videre kan man gjøre tilsvarende beregninger for å finne y_3:

$$\lambda_9 = \lambda_7\lambda_3^2 - 2x_3$$
$$y_3 = (\lambda_9\lambda_6 - \lambda_8\lambda_3^3)/2 \tag{5.18}$$

For punktdobling i Jacobiske koordinater gjelder følgende formler:

$$\lambda_1 = 3x_1^2 + az_1^4 \tag{5.19}$$
$$z_3 = 2y_1z_1 \tag{5.20}$$
$$\lambda_2 = 4x_1y_1^2 \tag{5.21}$$
$$x_3 = 8y_1^4 \tag{5.22}$$
$$y_3 = \lambda_1(\lambda_2 - x_3) - \lambda_3 \tag{5.23}$$

Dette avsnittet er inspirert av formelen som står på side 59 og 60 i [13], men vi har tilført noen ekstra mellomregninger.

5.5 Gruppeorden

Det er av avgjørende betydning for sikkerheten å kunne sjekke ordenen til elliptiske kurver som skal brukes til kryptering. Vi ønsker som regel at ordenen skal være delelig med et stort primtall, det vil si at ordenen til kurven er produktet av et lite tall s og et stort primtall r.

$$\#E(\mathbb{K}) = n = s \cdot r \tag{5.24}$$

Grunnen til at man ønsker å ha en stor primtallsfaktor i gruppeordenen, er at sikkerheten er proporsjonal med kvadratroten til den største primtallsfaktoren i gruppeordenen, se [13] kapittel 5 for mer detaljer. Det finnes ingen enkel metode som kan brukes til å telle antall punkter på en stor elliptisk kurve. En metode for å beregne antall punkter på en stor elliptisk kurve, er en modifisert utgave av Schoofs algoritme som står nærmere beskrevet i [13]. I den senere tid er imidlertid nye og raskere metoder utviklet.

5.5.1 Hasses teorem

Det viser seg at likning 5.3 har løsning for omtrent halvparten av $x \in \mathbb{F}_q$, dette vil si at omtrent $q/2$ x-verdier gir en løsning, se [13]. For hver x-verdi som gir løsning vil vi få to punkter på kurven; et med negativ y-verdi og et med positiv y-verdi, dette gir tilsammen ca q punkter. Legger vi til punktet i uendelig vil vi anta at antall punkter ligger i nærheten av $q + 1$. Dette bekreftes av følgende teorem.

Teorem 3. *(Hasses teorem) La E være en elliptisk kurve definert over $\mathbb{F}_q$. Ordenen til den elliptiske kurven kan da skrives som $\#E(\mathbb{F}_q) = q+1-t$, hvor $|t| \leq 2\sqrt{p}$.*

Bevis. Se teorem III.3 i [13] side 35. □

For store verdier av q gir Hasses teorem et relativt smalt område, som kalles Hasseintervallet og er $4\sqrt{q}$ bredt og sentrert rundt $q + 1$, som må inneholde ordenen til den elliptiske kurven.

Dette teoremet er spesielt nyttig; dersom vi har beregnet en verdi n som kan være ordenen til en elliptisk kurve, kan vi i mange tilfeller bruke Hasses teorem til å sjekke om n virkelig er ordenen til den elliptiske kurven.

5.5.2 Beregning av gruppeorden

For elliptiske kurver over kroppen $\mathbb{F}_p$, hvor p er et primtall som er $p < 10000$, er det relativt enkelt å beregne ordenen. Det finnes nemlig en måte å beregne hvor mange løsninger ligningen $x^2 \equiv a \pmod{p}$ har. Vi kan beregne *Legendresymbolet*, som er definert på følgende måte:

Definisjon 2. *La a være et heltall og $p > 2$ et primtall. Vi definerer da Legendresymbolet $\left(\frac{a}{p}\right)$ lik 0, 1, eller -1 på følgende måte:*

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{Hvis } a \text{ er et kvadrat modulo } p\text{, det vil si at vi har to løsninger.} \\ 0 & \text{Hvis } a|p\text{, det vil si at vi bare har løsningen 0.} \\ -1 & \text{Ellers, det vil si at vi ikke har noen løsning.} \end{cases}$$

Legendresymbolet $\left(\frac{a}{p}\right)$ kan beregnes på følgende måte:

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p} \tag{5.25}$$

Vi kan nå beregne ordenen til kurven, ved å beregne antall løsninger av likning 5.3, og legge til 1 for punktet i uendelig:

$$\#E(\mathbb{F}_p) = p + 1 + \sum_{x=0}^{p-1} \left(\frac{x^3 + ax + b}{p}\right) \tag{5.26}$$

Denne metoden fungerer ikke på kurver som kan brukes til kryptering, fordi det vil bli umulig å beregne denne summen for så store primtall.

5.6 Elliptiske kurver på Hesseform

Beskrivelsen av elliptiske kurver på Hesseform vil bli noe mer grundig enn beskrivelsen av Weierstrassform, fordi det er vanskeligere å finne gode beskrivelser av elliptiske kurver på Hesseform i litteraturen. Vi har basert denne beskrivelsen på følgende artikler [3] og [4]. Vi konsentrerer oss om kurver over $\mathbb{K} = \mathbb{F}_p$ hvor p er et primtall større en 3. Vi starter med å definere Hesseform:

Definisjon 3. *En elliptisk kurve over $\mathbb{K}$ på Hesseform er en plan kubisk kurve gitt av en likning på formen:*

$$E : x^3 + y^3 + 1 = Dxy, \tag{5.27}$$

eller i projektive koordinater

$$E : x^3 + y^3 + z^3 = Dxyz \tag{5.28}$$

hvor $D \in \mathbb{K}$ og $D^3 \neq 27$.

Vi vil i neste teorem vise at en elliptisk kurve på Hesseform ikke er singulær hvis $D^3 \neq 27$.

Teorem 4. *En kubisk kurve på Hesseform er singulær hvis og bare hvis $D^3 = 27$.*

Bevis. La $\mathbf{P} = (x_1 : y_1 : z_1)$ være et singulært punkt på kurven $f(x,y,z) = x^3 + y^3 + z^3 - Dxyz$. Da vil $\frac{\partial f(\mathbf{P})}{\partial x} = \frac{\partial f(\mathbf{P})}{\partial y} = \frac{\partial f(\mathbf{P})}{\partial z} = 0$, det vil si $3x_1^2 - Dy_1z_1 = 3y_1^2 - Dx_1z_1 = 3z_1^2 - Dx_1y_1 = 0$ ved å multiplisere de partiellderiverte med henholdsvis x_1, y_1 og z_1 og deretter trekke dem fra hverandre, får vi at $x_1^3 = y_1^3 = z_1^3 \neq 0$. Det må derfor eksistere $k \in \mathbb{K}^*$ og $r, s, t \in \mathbb{Z}_3$ slik at $x_1 = k\omega^r$, $y_1 = k\omega^s$, $z_1 = k\omega^t$, hvor ω er en ikketriviell kuberot av 1 i $\mathbb{K}$. Sammen med likning for kurven får vi at $Dk^3\omega^{r+s+t} = 3k^3$, vi opphøyer begge sider i 3 og får at $D^3 = 27$. □

Dersom Hessekurven ikke er singulær, kan vi beregne j-invarianten på følgende måte, se [4]:

$$j(E_D) = -\frac{D^3(D^3+216)^3}{-D^9 + 81D^6 - 2187D^3 + 19683} \tag{5.29}$$

Setning 1. *I $\mathbb{F}_p$ har ligningen $x^n = 1$ løsningene $x = 1, \omega^r, \omega^{2r}, \cdots, \omega^{(d-1)r}$, hvor $d = \gcd(n, p-1)$, $r = (p-1)/d$ og ω er en primitiv rot i $\mathbb{F}_p$.*

Bevis. Se [4] side 5. □

Dersom vi setter $z = 0$ inn i likning 5.28 får vi $x^3 + y^3 = 0 \Leftrightarrow x^3 = -y^3$, vi velger $x = 1$. I følge Setning 1 er $y = -1$ den eneste løsningen for y dersom $p \equiv 2 \pmod 3$. Dermed blir punktet i uendelig:

$$\mathcal{O} = (1, -1, 0) \tag{5.30}$$

La $\mathbf{P} = (x_1 : y_1 : z_1)$ være et punkt på kurven, punktene $\mathbf{P}$, $\mathcal{O}$ og $-\mathbf{P}$, vil ligge på linjen:

$$y = -x + (x_1 + y_1)z. \tag{5.31}$$

Vi ser videre på den affine versjonen av linjen, det vil si at vi setter $z = 1$. Vi kombinerer ligningen for linjen og for kurven og får:

$$x^3 + (-x + (x_1 + y_1))^3 + 1 = Dx(-x + (x_1 + y_1)) \tag{5.32}$$

$$x^2(3x_1 + 3y_1 + D) - x(x_1 + y_1)(3x_1 + 3y_1 + D) + (x_1 + y_1)^3 + 1 = 0 \tag{5.33}$$

Dersom vi utnytter at $\mathbf{P}$ er et punkt på kurven kan vi gjøre følgende forenkling:

$$(y_1 + x_1)^3 + 1 = (x_1^3 + y_1^3 + 1) + (3y_1 + 3x_1)x_1y_1 \tag{5.34}$$

$$(y_1 + x_1)^3 + 1 = (3D + x_1 + y_1)x_1y_1 \tag{5.35}$$

Setter vi forenklingen gitt ved likning 5.35 inn i ligningen 5.33 får vi:

$$x^2(3x_1 + 3y_1 + D) - x(x_1 + y_1)(3x_1 + 3y_1 + D) + (3x_1 + 3y_1 + D)x_1y_1 = 0 \tag{5.36}$$

$$x^2 - (x_1 + y_1)x + x_1y_1 = 0 \tag{5.37}$$

Da blir x-koordinaten til $-\mathbf{P}$ y_1, og y-koordinaten finner vi ved å setter x-koordinaten inn i ligningen 5.31.

$$-\mathbf{P} = (y_1 : x_1 : z_1) \tag{5.38}$$

5.6.1 Gruppeloven

Følgende formler er tatt med fordi de vil bli benyttet under utledning av gruppeloven på Hesseform.

Cauchy-Desboves Formel

La $f(x,y,z) = 0$ være en (homogen) likning for en generell kubisk kurve, og la $\mathbf{P_1} = (x_1 : y_1 : z_1)$ og $\mathbf{P_2} = (x_2 : y_2 : z_2)$ være to punkter på kurven. Videre skriver vi:

$$\varphi = \frac{\partial f(\mathbf{P_1})}{\partial x},\ \chi = \frac{\partial f(\mathbf{P_1})}{\partial y},\ \psi = \frac{\partial f(\mathbf{P_1})}{\partial z} \tag{5.39}$$

Tangenten til punktet $\mathbf{P_1}$ vil da krysse kurven i et tredje punkt:

$$\left(\frac{f(0,\psi,-\chi)}{x_1^2} : \frac{f(-\psi,0,\varphi)}{y_1^2} : \frac{f(\chi,-\varphi,0)}{z_1^2}\right) \tag{5.40}$$

Videre vil linjen som går gjennom $\mathbf{P_1}$ og $\mathbf{P_2}$ krysse kurven i et tredje punkt:

$$(x_1\Theta - x_2\Upsilon : y_1\Theta - y_2\Upsilon : z_1\Theta - z_2\Upsilon) \tag{5.41}$$

hvor

$$\Theta = x_1\frac{\partial f(\mathbf{P_2})}{\partial x} + y_1\frac{\partial f(\mathbf{P_2})}{\partial y} + z_1\frac{\partial f(\mathbf{P_2})}{\partial z} \tag{5.42}$$

$$\Upsilon = x_2\frac{\partial f(\mathbf{P_1})}{\partial x} + y_2\frac{\partial f(\mathbf{P_1})}{\partial y} + z_2\frac{\partial f(\mathbf{P_1})}{\partial z} \tag{5.43}$$

Disse likningene er hentet fra side 9 i [3], og bevis finnes i [14].

Punktaddisjon

Ligningen for en elliptisk kurve på Hesseform er helt symmetrisk med hensyn på x, y og z, dette vil lette arbeidet med å utlede addisjonsformlene. Utledningen vil basere seg på Cauchy-Desboves formel. Beviset som vi presenterer her er inspirert av [3], men man bør legge spesielt merke til at D ikke har samme betydning i denne artikkelen. Som for kurver på Weierstrassform vil formelen for punktdobling være forskjellig fra formelen for punktaddisjon. Vi starter med punktdobling:
La $\mathbf{P} = (x_1 : y_1 : z_1)$ være et punkt på den elliptiske kurven, tangenten i punktet $\mathbf{P}$ vil da skjære kurven E i et tredje punkt $-2\mathbf{P}$. Fra likning 5.40 , med $f(x,y,z) = x^3 + y^3 + z^3 = Dxyz$, får vi:

$$\varphi = 3x_1^2 - Dy_1z_1,\ \chi = 3y_1^2 - Dx_1z_1,\ \psi = 3z_1^2 - Dx_1y_1 \tag{5.44}$$

og

$$-2\mathbf{P} = \left(\frac{\psi^3 - \chi^3}{x_1^2} : \frac{\varphi^3 - \psi^3}{y^2} : \frac{\chi^3 - \varphi^3}{z_1^3}\right) \tag{5.45}$$

$$\psi^3 - \chi^3 = (3z_1^2 - Dx_1y_1)^3 - (3y_1^2 - Dx_1z_1)^3 \tag{5.46}$$

$$= 27(z_1^6 - y_1^6) + D^3x_1^3(z_1^3 - y_1^3) - 27Dx_1y_1z_1(z_1^3 - y_1^3) \tag{5.47}$$

Ved å utnytte at $\mathbf{P}$ er på kurven kan vi gjøre følgende forenkling:

$$27(z_1^6 - y_1^6) - 27Dx_1y_1z_1(z_1^3 - y_1^3) = 27((z_1^6 - y_1^6) - (x_1^3 + y_1^3 + z_1^3)(z_1^3 - y_1^3)) \tag{5.48}$$

$$= 27(y_1^3 - z_1^3)x_1^3 \tag{5.49}$$

Ved å benytte denne forenklingen får vi:

$$\psi^3 - \chi^3 = (z_1^3 - y_1^3)(D^3 - 27)x_1^3 \tag{5.50}$$

Ved bruk av symmetri får vi:

$$\varphi^3 - \psi^3 = (x_1^3 - z_1^3)(D^3 - 27)y_1^3 \tag{5.51}$$

$$\chi^3 - \varphi^3 = (y_1^3 - x_1^3)(D^3 - 27)z_1^3 \tag{5.52}$$

Vi får nå $2\mathbf{P}$ ved å bytte om x og y, og dele på $(D^3 - 27)$:

$$2\mathbf{P} = (y_1(x_1^3 - z_1^3) : x_1(z_1^3 - y_1^3) : z_1(y_1^3 - x_1^3)) \tag{5.53}$$

La $\mathbf{P} = (x_1 : y_1 : z_1)$ og $\mathbf{Q} = (x_2 : y_2 : z_2)$ være to punkter på kurven, linjen gjennom disse punktene vil da i følge ligning 5.41 krysse kurven i et tredje punkt:

$$-(\mathbf{P} + \mathbf{Q}) = (x_1\Theta - x_2\Upsilon : y_1\Theta - y_2\Upsilon : z_1\Theta - z_2\Upsilon) \tag{5.54}$$

hvor

$$\Theta = x_1(3x_2^2 - Dy_2z_2) + y_1(3y_2^2 - Dx_2z_2) + z_1(3z_2^2 - Dx_2y_2) \tag{5.55}$$

$$\Upsilon = x_2(3x_1^2 - Dy_1z_1) + y_2(3y_1^2 - Dx_1z_1) + z_2(3z_1^2 - Dx_1y_1) \tag{5.56}$$

Når vi setter 5.55 og 5.56 inn i 5.54 får vi:

$$x_1\Theta - x_2\Upsilon = 3y_1y_2(x_1y_2 - x_2y_1) + 3z_1z_2(x_1z_2 - x_2z_1) - D(x_1^2y_2z_2 - x_2^2y_1z_1)$$

$$y_1\Theta - y_2\Upsilon = 3x_1x_2(x_2y_1 - x_1y_2) + 3z_1z_2(y_1z_2 - y_2z_1) - D(y_1^2x_2z_2 - y_2^2x_1z_1)$$

$$z_1\Theta - z_2\Upsilon = 3x_1x_2(x_2z_1 - x_1z_2) + 3y_1y_2(y_2z_1 - y_1z_2) - D(z_1^2x_2y_2 - z_2^2x_1y_1)$$

Dersom vi utnytter at $\mathbf{P}$ og $\mathbf{Q}$ er punkter på kurven, og bytter om x og y for å få $(\mathbf{P} + \mathbf{Q})$ og ikke $-(\mathbf{P} + \mathbf{Q})$, så får vi:

$$(\mathbf{P} + \mathbf{Q}) = (y_1^2x_2z_2 - y_2^2x_1z_1 : x_1^2y_2z_2 - x_2^2y_1z_1 : z_1^2x_2y_2 - z_2^2x_1y_1) \tag{5.57}$$

Bemerkning 1: Prøver man å addere to forskjellige punkter med null i samme koordinat med formelen 5.57, vil det gå galt fordi vi får et "0/0" problem. I slike tilfeller kan man bruke Desboves formel direkte. For punktdobling er problemstillingen motsatt, dersom man prøver å doble et tretorsjonspunkt med Desboves formel vil det gå galt fordi vi her får "0/0" problem, men det finnes en grenseverdi som kan brukes, denne grenseverdien får man ved å bruke formelen 5.53, som vi har utledet for punktdobling. Under utledningen ble divisjonen som skapte problemet forkortet bort.

5.6.2 Tretorsjonspunkter

Med tretorsjonspunkter mener vi punkter som tilfredsstiller ligningen $[3]\mathbf{P} = \mathcal{O}$, det vil si punkter som har den egenskapen at dersom man legger det sammen med seg selv tre ganger så får man punktet $\mathcal{O}$. Vi vil nå se spesielt på tretorsjonspunkter på Hessekurver gitt ved likning 5.28 over $\mathbb{F}_p$ hvor p er et primtall større en tre.

Vi kan skrive:

$$[3]\mathbf{P} = \mathcal{O} \iff -\mathbf{P} = [2]\mathbf{P} \tag{5.58}$$

Innsatt i likning 5.53 for dobling og likning 5.38, får vi de tre likningene, en for hver koordinat:

$$\begin{aligned}\mathbf{P} &= (x_1 : y_1 : z_1)\\ y_1 &= y_1(x_1^3 - z_1^3)\\ x_1 &= x_1(z_1^3 - y_1^3)\\ z_1 &= z_1(y_1^3 - x_1^3)\end{aligned}$$

Dette viser at $x_1 = 0,\ y_1 = 0$ eller $z_1 = 0$, dersom $z_1 = 0$ får vi at:

$$x_1^3 = 1 \tag{5.59}$$

$$y_1^3 = -1 \tag{5.60}$$

Vi er interessert i å vite hvor mange løsninger likning 5.59 har. Vi må her ta hensyn til at vi bruker projektive koordinater, og vi vil bare få en ekvivalensklasse for hver løsning av likning 5.59.

Dersom $p \equiv 2 \pmod 3$, blir $d = \gcd(3, p-1) = 1$. Setning 1 sier at man da bare har den trivielle løsningen, og vi får bare ett tretorsjonspunkt hvor $z = 0$. På grunn av symmetri mellom x, y og z får vi de tre tretorsjonspunktene:

$$\{(1 : -1 : 0), (1 : 0 : -1), (0 : 1 : -1)\} \tag{5.61}$$

Dersom $p \equiv 1 \pmod 3$, blir $d = \gcd(3, p-1) = 3$, og vi får de tre løsningene $1, \epsilon$ og ϵ^2, hvor ϵ er en primitiv tredjerot av en. Vi får nå de ni tretorsjonspunktene:

$$\begin{aligned}&\{(1 : -1 : 0), (1 : 0 : -1), (0 : 1 : -1),\\ &(0 : 1 : -\epsilon), (0 : 1 : -\epsilon^2), (1 : 0 : -\epsilon),\\ &(1 : 0 : -\epsilon^2), (1 : -\epsilon : 0), (1 : -\epsilon^2 : 0)\}\end{aligned}$$

Bemerkning 2: Det er viktig å merke seg at antall tretorsjonspunkter bare er avhengig av p, og ikke er avhengig av D. Det vil si at alle kurver over $\mathbb{F}_p$ vil ha de samme tretorsjonspunktene. Legg også merke til at tretorsjonspunktene på en Hessekurve er de punktene hvor en av koordinatene er null.

Eksempel 3: La $E : x^3 + y^3 + z^3 = 2xyz$ være en elliptisk kurve over kroppen $\mathbb{F}_p$ hvor $p = 7$. Det første vi legger merke til er at $p \equiv 1 \pmod 3$, det vil si at vi kan ha ni tretorsjonspunkter. For å finne de ni tretorsjonspunktene, trenger vi en ikke-triviell tredjerot av 7, her finnes det bare en mulighet; 2. Det vil si at $\epsilon = 2$, vi finner at $-\epsilon = 5,\ -\epsilon^2 = 3$, dersom man bruker metoden som er vist i avsnitt 5.6.2, viser det seg at punktene på den elliptiske kurven er de ni tretorsjonspunktene. Dersom man bruker Desboves formel (likning 5.41) for addisjon av to forskjellige punkter, og formelen 5.53 til å doble punkter, får man gruppeoperasjonstabell som vist i Tabell 5.1.

Ser vi på nederste rad, ser vi at elementet $(6 : 1 : 0)$ må være identitetselementet, det stemmer med det vi har funnet tidligere siden $(6 : 1 : 0) \sim (-1 : 1 : 0) \sim (1 : -1 : 0)$.

La $\mathbf{P} = (x_1 : y_1 : z_1) = (0 : 3 : 1),\ \mathbf{Q} = (x_2 : y_2 : z_2) = (0 : 5 : 1)$ være punkter på den elliptiske kurven E. La oss prøve å beregne $\mathbf{R} = (x_3 : y_3 : z_3) = [2]\mathbf{P}$ ved å bruke Desboves formel. Dette blir problematisk, siden vi må dele på x_1^2 som er null, vi kan prøve å multiplisere alle koordinatene i svaret med $x_1^2 y_1^2 z_1^2$, men

Tabell 5.1: Addisjonstabell for den elliptiske kurven $E : x^3 + y^3 + z^3 = 2xyz$ over kroppen $\mathbb{F}_7$.

+	(0:3:1)	(0:5:1)	(0:6:1)	(3:0:1)	(5:0:1)	(6:0:1)	(3:1:0)	(5:1:0)	(6:1:0)
(0:3:1)	(3:0:1)	(6:0:1)	(5:0:1)	(6:1:0)	(5:1:0)	(3:1:0)	(0:5:1)	(0:6:1)	(0:3:1)
(0:5:1)	(6:0:1)	(5:0:1)	(3:0:1)	(3:1:0)	(6:1:0)	(5:1:0)	(0:6:1)	(0:3:1)	(0:5:1)
(0:6:1)	(5:0:1)	(3:0:1)	(6:0:1)	(5:1:0)	(3:1:0)	(6:1:0)	(0:3:1)	(0:5:1)	(0:6:1)
(3:0:1)	(6:1:0)	(3:1:0)	(5:1:0)	(0:3:1)	(0:6:1)	(0:5:1)	(6:0:1)	(5:0:1)	(3:0:1)
(5:0:1)	(5:1:0)	(6:1:0)	(3:1:0)	(0:6:1)	(0:5:1)	(0:3:1)	(3:0:1)	(6:0:1)	(5:0:1)
(6:0:1)	(3:1:0)	(5:1:0)	(6:1:0)	(0:5:1)	(0:3:1)	(0:6:1)	(5:0:1)	(3:0:1)	(6:0:1)
(3:1:0)	(0:5:1)	(0:6:1)	(0:3:1)	(6:0:1)	(3:0:1)	(5:0:1)	(5:1:0)	(6:1:0)	(3:1:0)
(5:1:0)	(0:6:1)	(0:3:1)	(0:5:1)	(5:0:1)	(6:0:1)	(3:0:1)	(6:1:0)	(3:1:0)	(5:1:0)
(6:1:0)	(0:3:1)	(0:5:1)	(0:6:1)	(3:0:1)	(5:0:1)	(6:0:1)	(3:1:0)	(5:1:0)	(6:1:0)

dette fører til at to av koordinatene i svaret blir null, og gir derfor ikke et gyldig svar. Bruker vi formelen 5.53 får vi følgende:

$$\begin{aligned} x_3 &= 3(1^3 - 0^3) = 3 \\ y_3 &= 0(3^3 - 1^3) = 0 \\ z_3 &= 1(0^3 - 3^3) = 1 \end{aligned}$$

Dette svaret er korrekt.

La oss addere to forskjellige punkter $\mathbf{P}$ og $\mathbf{Q}$, ved bruk av Desboves formel:

$$\begin{aligned} \Theta &= 0 + 3(3 \cdot 5^2 - 2 \cdot 0 \cdot 1) + 1(3 \cdot 1^3 - 2 \cdot 0 \cdot 5) = 4 \\ \Upsilon &= 0 + 5(3 \cdot 3^2 - 2 \cdot 0 \cdot 1) + 1(3 \cdot 1^3 - 2 \cdot 0 \cdot 3) = 5 \\ -(\mathbf{P} + \mathbf{Q}) &= (x_1\Theta - x_2\Upsilon : y_1\Theta - y_2\Upsilon : z_1\Theta - z_2\Upsilon) \\ &= (0 \cdot 4 - 0 \cdot 5 : 3 \cdot 4 - 5 \cdot 5 : 1 \cdot 4 - 1 \cdot 5) = (0 : 1 : -1) \\ \mathbf{P} + \mathbf{Q} &= (1 : 0 : -1) = (1 : 0 : 6) \sim (6 : 0 : 1) \end{aligned}$$

Denne utregningen ga riktig svar. Til slutt vil vi vise hvilket resultat vi får om vi bruker formelen 5.57 til å addere $\mathbf{P}$ og $\mathbf{Q}$:

$$\begin{aligned} P + Q &= (x_3 : y_3 : z_3) \\ x_3 &= 3^2 \cdot 0 \cdot 1 - 5^2 \cdot 0 \cdot 1 = 0 \\ y_3 &= 0^2 \cdot 5 \cdot 1 - 0^2 \cdot 3 \cdot 1 = 0 \\ z_3 &= 1^2 \cdot 5 \cdot 0 - 1^2 \cdot 3 \cdot 0 = 0 \end{aligned}$$

Dette svaret er åpenbart galt, siden $(0 : 0 : 0)$ ikke er et lovlig punkt i projektive koordinater. Dette går galt siden begge punktene har 0 i samme koordinat, som beskrevet i Bemerkning 1.

Kapittel 6

Implementasjon av Weierstrassform

6.1 Innledning

Weierstrassform er veldokumentert og solid teknologi som er nærmest enerådende i implementasjoner av ECC i dag. Anbefalte standardkurver er på Weierstrassform, og er grundig testet for sikkerhetshull. Det finnes gode retningslinjer og metoder for å velge kryptografisk sterke kurver.

LiDIA er et C++ bibliotek for tallteori, se [15], som vi vil bruke i forbindelse med implementasjon av punktoperasjoner på elliptiske kurver. LiDIA har god støtte for elliptiske kurver på kort Weierstrassform, og vi har brukt denne funksjonaliteten til å generere noen små[1] elliptiske kurver som vi har brukt til å teste addisjons- og doblingsfunksjonene i LiDIA. Vi har også sett på LiDIAs funksjonalitet for å undersøke kurvene med hensyn til orden, isomorfi, j-invariant og diskriminant.

Vi har tatt tiden på basisoperasjonene i LiDIA for objekter av typen `gf_element`, som representerer et element i kroppen $\mathbb{F}_q$, slik at vi har et grunnlag for å vurdere hvor effektive forskjellige algoritmer som bygger på disse basisoperasjonene vil kunne være.

Vi har beskrevet hvordan man kan implementere addisjon og dobling av punkter på elliptiske kurver på kort Weierstrassform, og vi vil se på forskjellige algoritmer for punktmultiplikasjon med og uten forhåndslagring av punkter. Vi har også tatt med noen eksempler på elliptiske kurver som vi har benyttet under testing av disse algoritmene.

Kildekoden til de viktigste delene av vår implementasjon finnes i Tillegg C.

6.2 Pakker og klasser i LiDIA som vi har brukt

Matematikkbiblioteket LiDIA er inndelt i forskjellige pakker som inneholder klasser og funksjoner for forskjellige formål. Her følger en oversikt over de pakkene i LiDIA som vi har brukt mest:

[1] Med små mener vi kurver med lav orden

Base er pakken som inneholder `bigint` og andre viktige typer, grunnleggende aritmetikk og templateklasser.

FF inneholder klasser for endelige kropper, blant annet klassene `galois_field` og `gf_element`.

EC er pakken for elliptiske kurver, blant annet klassene `elliptic_curve<gf_element>` og `point<gf_element>`.

ECO er en pakke for punkttelling på elliptiske kurver. Her har vi benyttet klassen `eco_prime` med funksjonen `compute_group_order()`.

6.3 Hastighetstestmetoder

Vi utfører en operasjon mange ganger, slik at vi kommer langt over tidsoppløsningen til tidtakeren. Vi legger også vekt på ikke å ha deklarasjon av variabler inni funksjoner som kalles ofte, slik at tiden en funksjon tar er mest mulig avhengig av algoritmen. Vi bruker ikke objektorientering, siden tidsforbruket til konstruktører, virtuelle funksjonskall og lignende kan føre til at det blir vanskeligere å tolke resultatene. Vi har forsøkt så langt det er mulig å lage like forhold for algoritmene vi tester, med hensyn til bruk av variabler, referanseoverføring, bruk av `const` for parametere som ikke endres. Vi tar funksjoner ut av LiDIA og omskriver dem i stil med våre egne funksjoner for Hesseform slik at de blir mest mulig like.

Vi gjør oppmerksom på at selv om vi har tatt tiden på en del grunnoperasjoner fra LiDIA, kan man ikke uten videre forvente å addere disse for operasjonene i en funksjon og komme fram til et eksakt likt resultat i forhold til måling på utføring av funksjonen selv, siden det går med en del ekstra tid blant annet til å gå i løkke.

6.4 Hastighetstest av grunnoperasjoner i LiDIA

I dette avsnittet vil vi presentere en hastighetstest av de viktigste grunnoperasjonene i LiDIA. Med de viktigste grunnoperasjonene mener vi addisjon, subtraksjon, multiplikasjon og lignende av kroppselementer. Et punkt $\mathbf{P} = (x, y)$ i det affine planet vil bestå av to kroppselementer, et element for x og et element for y. Addisjon av punkter vil derfor implementeres ved hjelp av operasjoner på objekter av klassen `gf_element`. Det er derfor avgjørende å vite litt om tidsforbruket til disse operasjonene.

Tidsforbruket til de fleste funksjonene er nesten helt uavhengig av hvilke verdier de kalles med, men spesielt addisjon og subtraksjon er et unntak fra dette og er avhengig av hvilke verdier de kalles med.

I Tabell 6.1 følger resultatet fra tester av basisoperasjoner på klassen `gf_element`, som i LiDIA representerer et element i kroppen $\mathbb{K} = \mathbb{F}_p$. I testene har vi benyttet:

$$p = 1224753567915253525600877180059052116597297173971$$

Primtallet p har en lengde på 160 bit, som er en realistisk størrelse i forbindelse med ECC. Testene er kompilert uten optimalisering, mens LiDIA i utgangspunktet er kompilert med optimalisering.

Vi trenger to elementer a og b som testene skal utføres på, disse velges tilfeldig ved hjelp av en innebygd funksjon i LiDIA. Vi trenger i tillegg en variabel c som brukes til retur av resultatet.

Tidtakingsfunksjonen vi bruker har en oppløsning på $10ms$ som er mye mindre enn tidsforbruket til operasjonene vi skal ta tiden på, vi har derfor utført de opperasjonene som tar kortest tid, det vil si `add()`, `subtract()`, `negate()` og `multiply_by_2()` 500 000 ganger, slik at det samlede tidsforbruket kommer over $150ms$. De resterende operasjonene tar lenger tid, og vi har derfor utført dem 50 000 ganger. Vi har utført alle testene 100 ganger, og beregnet gjennomsnittet fra alle kjøringene.

Tabell 6.1: Kjøretider for operasjoner i klassen `gf_element` på elementer med 160 bit.

Operasjon	Tid	Forkortelse
add(c,a,b)	$0.46\mu s$	A
subtract(c,a,b)	$0.44\mu s$	SU
negate(c,a)	$0.52\mu s$	
a.multiply_by_2()	$0.47\mu s$	I2
a.divide_by_2()	$23.76\mu s$	M2
multiply(c,a,b)	$2.57\mu s$	M
divide(c,a,b)	$39.72\mu s$	I
invert(c,a)	$33.39\mu s$	I
square(c,a)	$2.33\mu s$	SQ
power(c,a,3)	$11.46\mu s$	-

Bemerkning 4: Om kjøretidene vil vi spesielt påpeke at

- Testene er utført på en Compaq Armada 110.
- Multiplikasjon tar minst 4 ganger så mye tid som addisjon og subtraksjon.
- Divisjon tar ca. 15 ganger mer tid enn multiplikasjon.
- Vi har testet `power(c,a,3)`, fordi vi trenger å beregne a^3 når vi skal addere punkter på Hessekurver. Det viser seg at `power(c,a,3)` tar mye lenger tid enn `square(c,a)` etterfulgt av `multiply(c,c,a)`.
- `divide_by_2()` tar mye lengre tid enn `multiply_by_2()`.
- Under kompilering av LiDIA kan man velge mellom flere mulige multipresisjonsaritmetikkbiblioteker. Vi har konfigurert LiDIA til å benytte Gnu MP, men man kan ikke utelukke at andre biblioteker ville gi forskjellige resultater.

6.5 Eksempler på elliptiske kurver

Under implementasjonen av punktaddisjon på elliptiske kurver, trenger vi noen små kurver som vi kan bruke til å teste addisjonsformelen på. Disse kurvene er

ikke interessante i forbindelse med kryptografi, men de vil være greie å bruke til å avsløre feil i implementasjonen av punktaddisjon, fordi det er mulig å kontrollregne for hånd. Vi har derfor laget noen programmer som kan skrive ut informasjon om en elliptiske kurve på Weierstrassform. Tabell 6.2 viser alle Weierstrasskurver over kroppen $\mathbb{K} = \mathbb{F}_5$.

Tabell 6.2: Informasjon om Weierstrasskurver over kroppen $\mathbb{K} = \mathbb{F}_5$.

a	b	orden	isomorf med	Δ	$j(E)$
0	0	-	-	0	-
0	1	6	$\mathbb{Z}_6$	3	0
0	2	6	$\mathbb{Z}_6$	2	0
0	3	6	$\mathbb{Z}_6$	2	0
0	4	6	$\mathbb{Z}_6$	3	0
1	0	4	$\mathbb{Z}_2 \times \mathbb{Z}_2$	1	3
1	1	9	$\mathbb{Z}_9$	4	2
1	2	4	$\mathbb{Z}_4$	3	1
1	3	4	$\mathbb{Z}_4$	3	1
1	4	9	$\mathbb{Z}_9$	4	2
2	0	2	$\mathbb{Z}_2$	3	3
2	1	7	$\mathbb{Z}_7$	1	4
2	2	-	-	0	-
2	3	-	-	0	-
2	4	7	$\mathbb{Z}_7$	1	4
3	0	10	$\mathbb{Z}_{10}$	2	3
3	1	-	-	0	-
3	2	5	$\mathbb{Z}_5$	4	4
3	3	5	$\mathbb{Z}_5$	4	4
3	4	-	-	0	-
4	0	8	$\mathbb{Z}_4 \times \mathbb{Z}_2$	4	3
4	1	8	$\mathbb{Z}_8$	2	1
4	2	3	$\mathbb{Z}_3$	1	2
4	3	3	$\mathbb{Z}_3$	1	2
4	4	8	$\mathbb{Z}_8$	2	1

Informasjonen i Tabell 6.2 er generert ved å kjøre et program vi har skrevet som bruker funksjoner i klassen `elliptic_curve<gf_element>` i LiDIA til å utføre beregningene. Klassen `elliptic_curve<gf_element>` representerer en elliptisk kurve på Weierstrassform.

Følgende funksjoner i klassen `elliptic_curve<gf_element>` er benyttet til å generere informasjonen i tabellen:

- `group_order()`
- `discriminant()`
- `j_invariant()`
- `isomorphism_type()`

Bemerkning 5: Kurver som har samme j-invariant er isomorfe over den algebraiske tillukningen $\overline{\mathbb{K}}$, og kalles tvister. Kurvene som har $\Delta = 0$ er singulære.

6.6 Punktene på den elliptiske kurven

Det finnes ikke noen funksjon i LiDIA som kan finne alle punktene på en elliptisk kurve. Men det er mulig å sjekke om det finnes et punkt på kurven med en gitt x-verdi, ved å prøve å regne ut tilsvarende y-verdi. Vi har derfor laget en funksjon som løper gjennom alle x-verdier mellom 0 og karakteristikken p, og prøver å beregne en tilsvarende y-verdi.

Dersom vi finner en $y \neq 0$, må det finnes to punkter $\mathbf{P_1} = (x, y)$, og $\mathbf{P_2} = (x, -y)$, siden $-\mathbf{P} = -(x, y) = (x, -y)$. Dersom $y = 0$, har vi funnet ett punkt på kurven, $\mathbf{P} = (x, 0)$. Til slutt legger vi til punktet $\mathcal{O}$.

Vi kan benytte funksjonen `group_order()` i klassen `elliptic_curve<gf_element>` i LiDIA til å beregne antall punkter på en elliptisk kurve på kort Weierstrassform. Dermed kan vi lett se om resultatet kan være riktig ved å se om vi har funnet riktig antall punkter.

6.7 Affin punktaddisjon

Vi har brukt LiDIA sin implementasjon av affin punktaddisjon som er gjengitt i Algoritme 1. Dette er en direkte implementasjon av formelen for addisjon som står i avsnitt 5.2.3. Alle beregningene som utføres i punktaddisjonsalgoritmene utføres i LiDIA med objekter av klassen `gf_element`. Man bør legge merke til at algoritmen inneholder divisjon, som er tidkrevende i følge Tabell 6.1.

Algoritme 1 Affin punktaddisjon i kort Weierstrassform

INN: To affine punkter $\mathbf{P} = (x_1, y_1)$ og $\mathbf{Q} = (x_2, y_2)$ på samme elliptiske kurve
UT: $\mathbf{R} = \mathbf{P} + \mathbf{Q} = (x_3, y_3)$

1: $h_1 \leftarrow y_1,\ h_2 \leftarrow x_1$
2: $h_2 \leftarrow x_2 - h_2$
3: $h_2 \leftarrow 1/h_2$
4: $h_1 \leftarrow y_2 - h_1$
5: $h_3 \leftarrow h_1 \cdot h_2$
6: $h_1 \leftarrow h_3^2$
7: $h_1 \leftarrow h_1 - x_1$
8: $h_1 \leftarrow h_1 - x_2$
9: $h_2 \leftarrow x_1 - h_1$
10: $h_2 \leftarrow h_2 \cdot h_3$
11: $h_2 \leftarrow h_2 - y_1$
12: $x_3 \leftarrow h_1,\ y_3 \leftarrow h_2$

Tilsvarende har vi også brukt LiDIA sin implementasjon av punktdobling, som er gjengitt i Algoritme 2. Dette er en implementasjon av formelen for dobling som finnes i avsnitt 5.2.3 om punktdobling. Denne algoritmen inneholder også en tidkrevende divisjon av `gf_element`. Antall basisoperasjoner som kreves for å utføre punktaddisjon og punktdobling etter disse algoritmene er gjengitt

i Tabell 6.3. Addisjonsformelen må bare benyttes på forskjellige punkter, ellers gir den galt resultat.

Algoritme 2 Affin punktdobling i kort Weierstrassform

INN: Et affint punkt $\mathbf{P} = (x_1, y_1)$ på en elliptiske kurve, kurvens a
UT: $\mathbf{R} = [2]\mathbf{P} = (x_2, y_2)$

1: $h_1 \leftarrow x_1,\ h_2 \leftarrow y_1,\ h_3 \leftarrow y_1$
2: $h_2 \leftarrow 2 \cdot h_2$
3: $h_2 \leftarrow 1/h_2$
4: $h_1 \leftarrow h_1^2$
5: $h_1 \leftarrow 3 \cdot h_1$
6: $h_1 \leftarrow h_1 + a$
7: $h_3 \leftarrow h_1 \cdot h_2$
8: $h_1 \leftarrow h_3^2$
9: $h_2 \leftarrow x_1$
10: $h_2 \leftarrow 2 \cdot h_2$
11: $h_2 \leftarrow h_1 - h_2$
12: $h_1 \leftarrow x_1 - h_2$
13: $h_1 \leftarrow h_3 \cdot h_1$
14: $h_1 \leftarrow h_1 - y_1$
15: $x_2 \leftarrow h_2,\ y_2 \leftarrow h_1$

Tabell 6.3: Antall basisoperasjoner i affin Weierstrassform

Operasjon	Addisjon	Dobling
Kvadrering	1	2
Multiplikasjon	2	3
Invertering	1	1
Addisjon	0	1
Subtraksjon	6	3
Dobling	0	2

Eksempel 6: Vi vil i dette eksempelet bruke noen av programmene vi har laget til å se nærmere på kurven $E : y^2 = x^3 + 0x + 2$ over $\mathbb{K} = \mathbb{F}_7$. Kurvens orden er $\#E(\mathbb{K}) = 9$, det vil si at den består av ni punkter. Diskriminanten til kurven er 1, kurven er dermed ikke singulær. Kurvens j-invarianten er 0, og det betyr at kurven er supersingulær. Kurver som er supersingulære er generelt ansett for å være mindre egnet for bruk i kryptografi.

Tabell 6.4: Addisjonstabell for kurvene $E : y^2 = x^3 + 0x + 2$ over $\mathbb{K} = \mathbb{F}_7$.

+	(0,3)	(0,4)	(3,1)	(3,6)	(5,6)	(5,1)	(6,1)	(6,6)	O
(0,3)	(0,4)	O	(6,1)	(5,6)	(6,6)	(3,1)	(5,1)	(3,6)	(0,3)
(0,4)	O	(0,3)	(5,1)	(6,6)	(3,6)	(6,1)	(3,1)	(5,6)	(0,4)
(3,1)	(6,1)	(5,1)	(3,6)	O	(0,3)	(6,6)	(5,6)	(0,4)	(3,1)
(3,6)	(5,6)	(6,6)	O	(3,1)	(6,1)	(0,4)	(0,3)	(5,1)	(3,6)
(5,6)	(6,6)	(3,6)	(0,3)	(6,1)	(5,1)	O	(0,4)	(3,1)	(5,6)
(5,1)	(3,1)	(6,1)	(6,6)	(0,4)	O	(5,6)	(3,6)	(0,3)	(5,1)
(6,1)	(5,1)	(3,1)	(5,6)	(0,3)	(0,4)	(3,6)	(6,6)	O	(6,1)
(6,6)	(3,6)	(5,6)	(0,4)	(5,1)	(3,1)	(0,3)	O	(6,1)	(6,6)
O	(0,3)	(0,4)	(3,1)	(3,6)	(5,6)	(5,1)	(6,1)	(6,6)	O

Alle punktene har orden 3, dette betyr at alle punktene er tretorsjonspunkter. Ser vi tilbake på Hessekurven i Eksempel 3, legger vi merke til at alle punktene på denne kurven også er tretorsjonspunkter. Kurvene har også samme orden og j-invariant, og er isomorfe. I Tabell 6.4 vises en addisjonstabell for $E(\mathbb{K})$.

6.8 Punktaddisjon i Jacobisk projektive koordinater

Poenget med å bruke projektive koordinater, er at man slipper å utføre divisjon av `gf_element`.

Man kan bruker vanlige projektive koordinater, men det viser seg at det er mer effektivt å bruker Jacobiske projektive koordinater fordi dobling kan utføres med færre multiplikasjoner, og det trengs oftest flere doblinger enn addisjoner, se [16] for flere detaljer.

Vi har tatt utgangspunkt i algoritmen som er beskrevet i avsnitt 5.3 når vi har skrevet Algoritme 3 som vi bruker til å utføre punktdobling i Jacobiske projektive koordinater.

Algoritmen som brukes av LiDIA ligner svært på algoritmen vi bruker, men vår utgave er mer optimalisert med hensyn på hastighet og mindre på plass.

I LiDIAs implementasjon deler man et `gf_element` på 2 med funksjonen `divide_by_2()`. Dette er ganske tidkrevende, og vi har derfor valgt å skrive om algoritmen slik at man slipper å dele på 2:

$$(x : \frac{y}{2} : z) \sim (\alpha^2 x : \alpha^3 \frac{y}{2} : \alpha z)$$

Lar vi $\alpha = 2$, ser vi at punktet

$$(x : \frac{y}{2} : z) \sim (2^2 x : 2^2 y : 2z)$$

Vi kan altså erstatte den ene divisjonen på 2 med fem multiplikasjoner med 2, som utføres med funksjonen `multiply_by_2()` som er mye raskere en `divide_by_2()` , se Tabell 6.1. Vi har implementert begge varianter, med og uten denne endringen, og det viser seg at denne modifikasjonen har en positiv effekt på hastigheten.

Algoritme 3 Jacobisk projektiv punktaddisjon i kort Weierstrassform

INN: To punkter $\mathbf{P} = (x_1 : y_1 : z_1)$ og $\mathbf{Q} = (x_2 : y_2 : z_2)$ på samme elliptiske kurve

UT: $\mathbf{P} + \mathbf{Q} = (x_3 : y_3 : z_3)$

1: $h_1 \leftarrow x_1;\ h_2 \leftarrow x_2;\ h_4 \leftarrow y_1;\ z_3 \leftarrow z_1$
2: $h_4 \leftarrow z_2^2$
3: $h_5 \leftarrow z_1^2$
4: $h_1 \leftarrow h_1 \cdot h_4$
5: $h_2 \leftarrow h_2 \cdot h_5$
6: $h_3 \leftarrow h_1 - h_2$
7: $h_4 \leftarrow y_1 \cdot h_4$
8: $h_4 \leftarrow h_4 \cdot z_2$
9: $h_5 \leftarrow y_2 \cdot h_5$
10: $h_5 \leftarrow h_5 \cdot z_1$
11: $h_6 \leftarrow h_4 - h_5$
12: $h_2 \leftarrow h_1 + h_2$
13: $h_5 \leftarrow h_4 + h_5$
14: $z_3 \leftarrow z_1 \cdot z_2$
15: $z_3 \leftarrow z_3 \cdot h_3$
16: $h_1 \leftarrow h_3^2$
17: $h_4 \leftarrow h_2 \cdot h_1$
18: $x_3 \leftarrow h_6^2$
19: $x_3 \leftarrow x_3 - h_4$
20: $y_3 \leftarrow x_3$
21: $y_3 \leftarrow 2 \cdot y_3$
22: $y_3 \leftarrow h_4 - y_3$
23: $y_3 \leftarrow y_3 \cdot h_6$
24: $h_5 \leftarrow h_5 \cdot h_3$
25: $h_5 \leftarrow h_5 \cdot h_1$
26: $y_3 \leftarrow y_3 - h_5$
27: $y_3 \leftarrow 2^2 \cdot y_3;\ x_3 \leftarrow 2^2 \cdot x_3;\ z_3 \leftarrow 2 \cdot z_3$

6.8.1 Blandede koordinater

Dersom $z_2 = 1$, sier vi at $\mathbf{Q}$ er oppgitt i affine koordinater, i dette tilfellet kan man sløyfe linje 2,4,7,8, og linje 14 i Algoritme 3, og vi omtaler dette som blandede koordinater, se [16]. Vi tester derfor om z_2-koordinaten er forskjellig fra 1 før vi gjør disse beregningene.

Tabell 6.5 inneholder en oversikt over hvor mange forskjellige basisoperasjoner som kreves for å utføre punktdobling og punktaddisjon.

Bemerkning 7: Dersom man velger $a = -3$ kan man redusere antall kvadreringer med 2 i doblingsalgoritmen, se [13] side 60.

Algoritme 4 Jacobisk projektiv punktdobling i kort Weierstrassform

INN: Et Jacobisk projektivt punkt $\mathbf{P} = (x_1 : y_1 : z_1)$ på en elliptiske kurve, a for denne kurven

UT: $\mathbf{R} = [2]\mathbf{P} = (x_2 : y_2 : z_2)$

1: $h_1 \leftarrow x_1,\ h_2 \leftarrow y_1,\ h_3 \leftarrow z_1$
2: $h_5 \leftarrow h_3^2$
3: $h_5 \leftarrow h_5^2$
4: $h_5 \leftarrow h_5 \cdot a$
5: $h_4 \leftarrow h_1^2$
6: $h_6 \leftarrow h_4 + h_4$
7: $h_4 \leftarrow h_4 + h_6$
8: $h_4 \leftarrow h_4 + h_5$
9: $h_3 \leftarrow h_3 \cdot h_2$
10: $h_3 \leftarrow 2 \cdot h_3$
11: $h_2 \leftarrow h_2^2$
12: $h_5 \leftarrow h_1 \cdot h_2$
13: $h_5 \leftarrow 2 \cdot h_5$
14: $h_5 \leftarrow 2 \cdot h_5$
15: $h_1 \leftarrow h_4^2$
16: $h_6 \leftarrow h_5 + h_5$
17: $h_1 \leftarrow h_1 - h_6$
18: $h_2 \leftarrow h_2^2$
19: $h_2 \leftarrow 2 \cdot h_2$
20: $h_2 \leftarrow 2 \cdot h_2$
21: $h_2 \leftarrow 2 \cdot h_2$
22: $h_5 \leftarrow h_5 - h_1$
23: $h_5 \leftarrow h_5 \cdot h_4$
24: $h_2 \leftarrow h_5 - h_2$
25: $x_2 \leftarrow h_1,\ y_2 \leftarrow h_2,\ z_2 \leftarrow h_3$

Tabell 6.5: Antall basisoperasjoner i Jacobisk projektiv Weierstrassform

Operasjon	Addisjon	Addisjon med bl.	Dobling
Kvadrering	4	3	6
Multiplikasjon	12	8	4
Addisjon	2	1	4
Subtraksjon	5	5	3
Dobling	6	7	6

6.9 Punktmultiplikasjon

I dette avsnittet vil vi se på metoder for å beregne

$$\mathbf{Q} = [k]\mathbf{P} = \underbrace{\mathbf{P} + \mathbf{P} + \cdots + \mathbf{P}}_{k}$$

En algoritme som ikke vil fungere i praksis er å beregne $\mathbf{Q}$ på følgende måte:

$$\mathbf{Q} = (((\mathbf{P} + \mathbf{P}) + \mathbf{P}) + \cdots) + \mathbf{P}$$

Denne metoden vil kreve $k-2$ addisjoner og en dobling, og vil være helt ubrukelig selv for relativt små verdier av k.

6.9.1 "Right-to-left" algoritmen

Vi kan utnytte det faktum at k kan skrives på følgende måte

$$k = \sum_{i=0}^{n-1} k_i 2^i = k_0 2^0 + k_1 2^1 + \cdots + k_{n-1} k^{n-1}$$

hvor k_i betyr bit nummer i. Algoritme 5 som baserer seg på denne skrivemåten kalles for RL (Right-to-left) algoritmen.

Algoritme 5 Punktmultiplikasjon ved bruk av RL algoritmen

INN: Et punkt $\mathbf{P}$, og et heltall k med n bit
UT: $\mathbf{Q} = [k]\mathbf{P}$

```
R ← P; Q ← O
for i from 0 to n − 1 do
  if k_i = 1 then
    Q = Q + R
  end if
  R ← [2]R
end for
```

Algoritme 5 krever $n-1$ doblinger, fordi vi må beregne $[2^1]\mathbf{P}, [2^2]\mathbf{P}, \cdots, [2^{n-1}]\mathbf{P}$ og $\omega(k) - 1$ addisjoner, hvor $\omega(k)$ er Hammingvekten[2] til k.

6.9.2 "Left-to-right" algoritmen

Vi kan også skrive k på følgende måte:

$$k = 2(\cdots 2(2k_{n-1} + k_{n-2}) + \cdots) + k_0$$

Denne skrivemåten leder til en algoritme som kalles for LR(Left-to-right) algoritmen. Fordelen med denne algoritmen er at resultatet fra doblingen kan lagres i samme variabel som resultatet fra addisjonen, vi reduserer dermed minnebehovet, som i noen tilfeller kan være svært viktig.

Denne metoden krever også $n-1$ doblinger og $\omega(k) - 1$ addisjoner. Det er kjent at $n-1$ doblinger er det beste man kan oppnå, men antall addisjoner kan reduseres, og dette er tema i neste avsnitt.

[2]Med Hammingvekten til k mener vi antall siffer siffer i representasjonen som ikke er null. Det er vanlig å bruke notasjonen $\omega(k)$.

Algoritme 6 Punktmultiplikasjon ved bruk av LR algoritmen

INN: Et punkt **P**, og et heltall k med n bit
UT: $\mathbf{Q} = [k]\mathbf{P}$

```
Q ← 𝒪
for i from n − 1 to 0 do
  Q ← [2]Q
  if k_i = 1 then
    Q = Q + P
  end if
end for
```

6.9.3 Modifisert "Left-to-right" algoritme

Vi skal nå se på en metode som kan øke effektiviteten til LR metoden. Tidsforbruket til LR-algoritmen er sterkt avhengig av Hammingvekten til representasjonen av k. Ved bruk av normal binær representasjon skriver vi:

$$k = \sum_{i=0}^{n-1} k_i 2^i$$

Her representerer k_i bit nummer i, og vi har følgelig at $k_i \in \{0, 1\}$. Men vi kan også skrive k på følgende måte:

$$k = \sum_{i=0}^{n-1} k'_i 2^i,\ k'_i \in \{\overline{1}, 0, 1\}$$

hvor $\overline{1} = -1$. Denne representasjonsformen kalles ofte *Signed Digit* (SD2) representasjon. Det er klart at dette ikke er en entydig representasjonsform, siden vi kan representere samme tall på forskjellige måte. Eksempelvis vil både 1111_{SD} og $1000\overline{1}_{SD}$ representere tallet 15. Ved bruk av SD representasjon kan vi nå beregne $[15]\mathbf{P}$ ved å bruke 2 addisjoner og 4 doblinger, bruker vi vanlig binær representasjon må vi utføre 3 doblinger og 4 addisjoner. Husk at det er svært lett å beregne $-\mathbf{P}$ om man kjenner **P**.

Det kan vises at det alltid finnes en unik SD2 representasjon med lavest mulig Hammingvekt, denne representasjonsformen kalles også for nonadjacent form (NAF), fordi det kan vises at tall representert i dette formatet ikke vil inneholde to påfølgende ikke-null siffer, for bevis se [17] side 7.

Det er enkelt å beregne NAF-representasjonen ved å utnytte følgende sammenheng: $k = (3k - k)/2$. La $l = 3k$, og $l_i \in \{0, 1\}$. Bit $k'_i \in \{\overline{1}, 0, 1\}$ i NAF representasjonen beregnes på følgende måte, se [17] for flere detaljer:

$$k'_i = l_{i+1} - k_{i+1}$$

Observasjonene er implementert i Algoritme 7, som er en modifisert utgave av Algoritme 6, hvor det benyttes både subtraksjon og addisjon av punkter. Reglene som bestemmer om det skal adderes eller subtraheres er gitt i Tabell 6.6.

Algoritme 7 brukes av LiDIA, og vi har også valgt å bruke denne algoritmen i våre tester. Denne algoritmen krever maksimalt $n = |k|$ doblinger, det vil si en dobling mer enn for LR-algoritmen, og $\omega(k') - 1$ addisjoner, det vil si antall siffer i NAF representasjonen til k som ikke er null, minus 1.

Tabell 6.6: Regler for addisjon og subtraksjon.

l_{i+1}	k_{i+1}	Operasjon
0	0	-
0	1	subtrahér
1	0	addér
1	1	-

Algoritme 7 Punktmultiplikasjon ved bruk av NAF

INN: Et punkt **P**, og et heltall k med n bit
UT: $\mathbf{Q} = [k]\mathbf{P}$

$\hat{k} \leftarrow 3k$
$\mathbf{Q} \leftarrow \mathbf{P}$
if $m = 0$ **then**
  $\mathbf{Q} \leftarrow \mathcal{O}$
  return Q
end if
if $k < 0$ **then**
  $k \leftarrow -k$, $\hat{k} \leftarrow -\hat{k}$, $\mathbf{Q} \leftarrow -\mathbf{Q}$
end if
if k=1 **then**
  return Q
end if
$\mathbf{R} \leftarrow \mathcal{O}$
for i **from** $bitlen(\hat{k}) - 1$ **to** 1 **do**
  $\mathbf{R} \leftarrow [2]\mathbf{R}$
  if $k_i = 0 \wedge \hat{k}_i = 1$ **then**
    $\mathbf{R} \leftarrow \mathbf{R} + \mathbf{Q}$
  else if $k_i = 1 \wedge \hat{k}_i = 0$ **then**
    $\mathbf{R} \leftarrow \mathbf{R} - \mathbf{Q}$
  end if
end for
$\mathbf{Q} \leftarrow \mathbf{R}$
return Q

6.9.4 Forhåndslagring

Dersom punktet man opererer på er kjent på forhånd, kan man utføre doblinger med dette på forhånd slik at man under punktmultiplikasjonen bare utfører addisjoner. Slik kan man spare tid ved å ofre en del lagerplass, nærmere bestemt ett punkt for hver bit i lengden av tallet det skal multipliseres med.

Når vi benytter forhåndsberegnede punkter, bruker vi samme algoritme som nevnt over, men vi sløyfer det som har med dobling av punkter å gjøre. Punktet $[2^i]\mathbf{P}$ ligger lagret på plass i i tabellen med forhåndslagrede punkter.

Dersom man vil forhåndsberegne $[2]\mathbf{P}$, $[4]\mathbf{P}$, $[8]\mathbf{P}$, osv, kan man lagre disse i normalisert form slik at $z = 1$, dette vil gjøre beregningene mer effektive ved at man kan benytte blandede koordinater i addisjonene.

Det finnes flere andre gode algoritmer for punktmultiplikasjon med eller uten forhåndslagring. Flere av disse beskrives i kapittel 4 i [13]. Man trenger ikke å forhåndslagre alle punktene, man kan forbedre ytelsen betraktelig ved å forhåndslagre relativt få punkter.

6.10 Oppsummering

I dette kapittelet har vi studert implementasjon av addisjons- og doblingsalgoritmer for kort Weierstrassform. Vi har konsentrert oss mest om den projektive formen av disse algoritmene siden den ser ut til å være raskest ut fra våre tester av basisoperasjonene. I affin form må man utføre divisjon både under addisjon og dobling, og det er svært tidkrevende.

Vi har laget våre egne versjoner av LiDIAs Jacobisk projektive funksjoner for addisjon og dobling på kort Weierstrassform får å få en mer rettferdig sammenligning mellom kort Weierstrassform og Hesseform. Den viktigste endringen vi har gjort er at vi velger en annen representant for resultatpunktet slik at vi kan erstatte en `divide_by_2()`-operasjon med 5 doblinger, noe som gir en raskere addisjonsfunksjon.

Vi har påpekt at man kan spare mange operasjoner ved å benytte blandede koordinater under addisjon. Dette kan utnyttes i praksis ved å benytte forhåndslagrede punkter på affin form under multiplikasjon.

Kapittel 7

Implementasjon av Hesseform

7.1 Innledning

Hesseform er ikke i praktisk bruk i dag, men vi mener den har et stort potensiale. Den ser ut til å være velegnet angående hastighet, og implementasjon av addisjons- og doblingsfunksjoner er mindre kompleks enn tilsvarende for kort Weierstrassform. Andre fordeler med Hesseform er at algoritmene er godt egnet for parallell utføring. Det har også blitt vist i [3] at Hesseform kan være spesielt effektiv når man skal lage systemer som er motstandsdyktige mot sidekanalangrep.

Et problem er at de kurvene som i dag anbefales for bruk i kryptografi ikke kan skrives på Hesseform siden de har en orden som ikke er delelig på 3.

Mange konsepter i forrige kapittel er direkte overførbart til Hesseform og vil derfor ikke bli forklart her.

Vi har også sett litt på sammenhengen mellom kurver på Hesse- og kort Weierstrassform og hvordan man kan konvertere kurver mellom disse formene.

Kildekoden til de viktigste delene av vår implementasjon finnes i Tillegg C.

7.2 Oversikt over "små" Hessekurver

Som vi nevnte i kapittelet om Weierstrassform trenger vi noen kurver med få punkter, som kan brukes til å teste implementasjonen av addisjons- og multiplikasjonsformlene. LiDIA har ingen støtte for Hesseform, så vi kan ikke benytte innebygde funksjoner til å utføre disse beregningene. Vi måtte derfor lage et program som finner kurver med en gitt karakteristikk, det vil si finner en D slik at kurven ikke er singulær. Fra Teorem 4 ser vi at en kurve er singulær dersom $D^3 \equiv 27$. Under testingen av addisjons- og doblingsformlene trenger vi en kurve og noen kjente punkter på kurven. Vi trenger også en mulighet for å sjekke at svaret er et punkt på kurven. Det er også nyttig å vite noe om ordenen til kurven, vi har derfor laget et program som finner alle punktene på kurven, og oppbevarer dem i en tabell under testen. Punktene på kurven finner vi ved å prøve alle mulige verdier for x, y og z i formel 5.28.

LiDIA har støtte for å finne j-invarianten og hvilken kartesisk produktgruppe $E_{a,b}(\mathbb{K})$ er isomorf med, vi har derfor implementert tilsvarende rutiner for Hesseform. Beregningen av j-invarianten gjør vi ved hjelp av en implementasjon av formel 5.29. I implementasjonen bruker vi klassen `gf_element`, fordi alle beregninger skal utføres modulo karakteristikken til kroppen. Ordenen til små kurver beregner vi ved å telle antall punkter vi har funnet.

Teorem 5. *La E være en elliptisk kurve over en endelig kropp $\mathbb{K} = \mathbb{F}_q$ hvor p er et primtall, r er et positivt heltall og $q = p^r$, da vil følgende gjelde:*

$$E(\mathbb{K}) \simeq \mathbb{Z}_m \times \mathbb{Z}_n$$

hvor $m, n \in \mathbb{Z}^+$, og n deler m.

Bevis. Se [13]. □

Teorem 5 sier at alle elliptiske kurver over endelige kropper vil være isomorfe med en kartesisk produktgruppe, som er et produkt av maksimalt to sykliske grupper. Vi er videre interessert i å finne m og n. Setning 2 sier hvordan man kan finne m og n.

Setning 2. *Vi kan finne m og n i Teorem 5 på følgende måte:*

$$m = \max(\#\langle \mathbf{P} \rangle) \tag{7.1}$$

$$n = \#E(\mathbb{K})/m \tag{7.2}$$

hvor $\mathbf{P} \in E(\mathbb{K})$.

Bevis. Vi har fra Teorem 5 at $E(\mathbb{F}_q) \simeq \mathbb{Z}_m \times \mathbb{Z}_n$ hvor n deler m. Vi vil nå vise at m må være ordenen til punktet $\mathbf{P}$ i $E(\mathbb{F}_q)$ med høyest orden. Av det vil det følge at $mn = \#E(\mathbb{F}_q)$, og $n = \#E(\mathbb{F}_q)/m$.

La oss anta at m er mindre enn ordenen til punktet $\mathbf{P}$. Dette er imidlertid ikke mulig da gruppen $\mathbb{Z}_m \times \mathbb{Z}_n$ ikke inneholder elementer med orden høyere enn m.

La oss anta at m er større en ordenen til punktet $\mathbf{P}$. Dette er ikke mulig da gruppen $\mathbb{Z}_m \times \mathbb{Z}_n$ inneholder minst et elementer med orden m.

Vi har nå vist at m ikke kan være mindre eller større enn ordenen til punktet $\mathbf{P}$, og setningen er bevist. □

I forbindelse med ytelsestestene er vi også interessert i å finne korresponderende kurver på Weierstrassform, dette gjør vi ved å bruke metoden som står beskrevet i Teorem 6 på side 59. Videre har vi brukt Setning 2 til å finne hvilken kartesisk produktgruppe Hessekurven er isomorf med[1]. Vi bruker en tilsvarende funksjon i LiDIA for Weierstrassform. Vi har sjekket at de korresponderende Hesse- og Weierstrasskurvene er isomorfe med samme produktgruppe, og som en ekstra test har vi også sjekket at de har samme j-invariant og ordenen.

[1] Ved bruk av Setning 2 trenger man å kjenne ordenen til alle punktene på kurven. Ordenen til punktene beregner vi ved å addere punktet med seg selv helt til man får punktet $\mathcal{O}$. Siden vi her også behandler kurver hvor $p \equiv 1 \pmod 3$ må vi bruke formel 5.41 for addisjon, og formel 5.53 til dobling.

Resultatet fra kjøringen av programmet som er beskrevet over, er oppsummert i Tabell 7.1 og Tabell 7.2. Tabell 7.1 inneholder informasjon om alle Hessekurver med karakteristikk mindre eller lik 17, og Tabell 7.2 inneholder alle Hessekurver med karakteristikk 19 og 23. Vi har bare sett på kurver over $\mathbb{K} = \mathbb{F}_p$ hvor p er et primtall. Som en tilleggsopplysning for Weierstrasskurvene har vi oppgitt diskriminanten Δ, denne gjelder bare Weierstrasskurven. Som vi ser i tabellen finnes det en kurve for de fleste mulige D-verdier, de som mangler er de singulære kurvene.

7.2.1 Strukturen

Ser vi på Tabell 7.1 og Tabell 7.2, ser vi at alle gruppene hvor $p \equiv 1 \pmod 3$ er isomorfe med kartesiske produktgrupper hvor begge faktorene er sykliske grupper med orden som er et multiplum av tre. For grupper hvor $p \equiv 2 \pmod 3$ ser vi at gruppen enten er en ren syklisk gruppe eller en produktgruppe hvor den ene faktoren er $\mathbb{Z}_2$. Vi har ikke vist flere kurver enn opp til $p = 23$ men vi har også sjekket alle kurver for $p = 29, 31$, og det viser seg at tendensen vi påpeker også stemmer for disse kurvene.

Vi ser at det er forholdsvis få forskjellige kurver over kropper hvor $p \equiv 1 \pmod 3$, og det er uavklart spørsmål om det også er slik for kurver over kropper $\mathbb{F}_p$ med høyere karakteristikk. Alle disse kurvene vil ha en orden på formen 3^2k og er dermed ikke på formen $3p$ som vi har valgt å benytte.

Vi har bare sett på kurver over $\mathbb{F}_p$, men det kunne også være interessant å undersøke kurver over $\mathbb{F}_{2^n}$.

Tabell 7.1: Alle Hessekurver med karakteristikk $5 \leq p \leq 17$.

Hesseform					Weierstrassform		
p	D	$\#E(\mathbb{F}_p)$	$j(E)$	Gruppeisomorfi	a	b	Δ
5	0	6	0	$\mathbb{Z}_6$	0	3	2
5	1	9	2	$\mathbb{Z}_9$	1	4	4
5	2	3	2	$\mathbb{Z}_3$	4	2	1
5	4	6	0	$\mathbb{Z}_6$	0	4	3
7	0	9	0	$\mathbb{Z}_3 \times \mathbb{Z}_3$	0	2	1
7	1	9	0	$\mathbb{Z}_3 \times \mathbb{Z}_3$	0	2	1
7	2	9	0	$\mathbb{Z}_3 \times \mathbb{Z}_3$	0	2	1
7	4	9	0	$\mathbb{Z}_3 \times \mathbb{Z}_3$	0	2	1
11	0	12	0	$\mathbb{Z}_{12}$	0	8	6
11	1	6	3	$\mathbb{Z}_6$	1	8	8
11	2	9	10	$\mathbb{Z}_9$	1	4	9
11	4	9	4	$\mathbb{Z}_9$	8	6	3
11	5	12	0	$\mathbb{Z}_{12}$	0	4	7
11	6	18	3	$\mathbb{Z}_{18}$	5	5	10
11	7	12	1	$\mathbb{Z}_{12}$	1	0	2
11	8	15	4	$\mathbb{Z}_{15}$	2	9	4
11	9	15	10	$\mathbb{Z}_{15}$	3	6	1
11	10	12	1	$\mathbb{Z}_6 \times \mathbb{Z}_2$	2	0	5
13	0	9	0	$\mathbb{Z}_3 \times \mathbb{Z}_3$	0	10	12
13	2	18	12	$\mathbb{Z}_6 \times \mathbb{Z}_3$	11	0	5
13	4	18	12	$\mathbb{Z}_6 \times \mathbb{Z}_3$	8	0	5
13	5	18	12	$\mathbb{Z}_6 \times \mathbb{Z}_3$	8	0	5
13	6	18	12	$\mathbb{Z}_6 \times \mathbb{Z}_3$	7	0	5
13	7	9	0	$\mathbb{Z}_3 \times \mathbb{Z}_3$	0	3	12
13	8	9	0	$\mathbb{Z}_3 \times \mathbb{Z}_3$	0	3	12
13	10	18	12	$\mathbb{Z}_6 \times \mathbb{Z}_3$	7	0	5
13	11	9	0	$\mathbb{Z}_3 \times \mathbb{Z}_3$	0	3	12
13	12	18	12	$\mathbb{Z}_6 \times \mathbb{Z}_3$	11	0	5
17	0	18	0	$\mathbb{Z}_{18}$	0	10	14
17	1	21	2	$\mathbb{Z}_{21}$	7	11	15
17	2	21	7	$\mathbb{Z}_{21}$	15	2	8
17	4	12	6	$\mathbb{Z}_{12}$	12	10	7
17	5	12	10	$\mathbb{Z}_6 \times \mathbb{Z}_2$	4	5	13
17	6	21	13	$\mathbb{Z}_{21}$	3	11	9
17	7	18	8	$\mathbb{Z}_{18}$	16	4	3
17	8	24	10	$\mathbb{Z}_{12} \times \mathbb{Z}_2$	8	9	2
17	9	24	1	$\mathbb{Z}_{24}$	4	16	11
17	10	15	7	$\mathbb{Z}_{15}$	4	6	4
17	11	18	0	$\mathbb{Z}_{18}$	0	2	6
17	12	15	13	$\mathbb{Z}_{15}$	10	9	16
17	13	24	6	$\mathbb{Z}_{24}$	10	13	12
17	14	18	8	$\mathbb{Z}_{18}$	2	5	10
17	15	15	2	$\mathbb{Z}_{15}$	14	13	1
17	16	12	1	$\mathbb{Z}_{12}$	15	6	5

Tabell 7.2: Alle Hessekurver med karakteristikk 19 og 23.

Hesseform					Weierstrassform		
p	D	$\#E(\mathbb{F}_p)$	$j(E)$	Gruppeisomorfi	a	b	Δ
19	0	27	0	$\mathbb{Z}_9 \times \mathbb{Z}_3$	0	5	11
19	1	18	1	$\mathbb{Z}_6 \times \mathbb{Z}_3$	10	8	8
19	4	18	1	$\mathbb{Z}_6 \times \mathbb{Z}_3$	13	8	8
19	5	18	1	$\mathbb{Z}_6 \times \mathbb{Z}_3$	8	18	12
19	6	18	1	$\mathbb{Z}_6 \times \mathbb{Z}_3$	10	8	8
19	7	18	1	$\mathbb{Z}_6 \times \mathbb{Z}_3$	13	8	8
19	8	18	1	$\mathbb{Z}_6 \times \mathbb{Z}_3$	3	12	18
19	9	18	1	$\mathbb{Z}_6 \times \mathbb{Z}_3$	15	8	8
19	10	27	0	$\mathbb{Z}_9 \times \mathbb{Z}_3$	0	17	1
19	11	18	1	$\mathbb{Z}_6 \times \mathbb{Z}_3$	15	8	8
19	12	18	1	$\mathbb{Z}_6 \times \mathbb{Z}_3$	14	12	18
19	13	27	0	$\mathbb{Z}_9 \times \mathbb{Z}_3$	0	17	1
19	15	27	0	$\mathbb{Z}_9 \times \mathbb{Z}_3$	0	17	1
19	16	18	1	$\mathbb{Z}_6 \times \mathbb{Z}_3$	18	18	12
19	17	18	1	$\mathbb{Z}_6 \times \mathbb{Z}_3$	12	18	12
19	18	18	1	$\mathbb{Z}_6 \times \mathbb{Z}_3$	2	12	18
23	0	24	0	$\mathbb{Z}_{24}$	0	5	10
23	1	24	3	$\mathbb{Z}_{24}$	12	0	15
23	2	24	19	$\mathbb{Z}_{12} \times \mathbb{Z}_2$	4	16	13
23	4	18	20	$\mathbb{Z}_{18}$	10	22	14
23	5	24	3	$\mathbb{Z}_{12} \times \mathbb{Z}_2$	22	0	18
23	6	18	14	$\mathbb{Z}_{18}$	10	19	20
23	7	30	20	$\mathbb{Z}_{30}$	22	18	5
23	8	15	7	$\mathbb{Z}_{15}$	6	13	16
23	9	27	12	$\mathbb{Z}_{27}$	17	22	6
23	10	18	22	$\mathbb{Z}_{18}$	10	5	19
23	11	27	21	$\mathbb{Z}_{27}$	1	3	4
23	12	30	14	$\mathbb{Z}_{30}$	21	16	21
23	13	33	7	$\mathbb{Z}_{33}$	1	11	12
23	14	27	15	$\mathbb{Z}_{27}$	2	14	8
23	15	21	21	$\mathbb{Z}_{21}$	8	10	1
23	16	18	18	$\mathbb{Z}_{18}$	18	19	7
23	17	24	0	$\mathbb{Z}_{24}$	0	3	22
23	18	21	15	$\mathbb{Z}_{21}$	6	18	9
23	19	21	12	$\mathbb{Z}_{21}$	11	13	2
23	20	30	22	$\mathbb{Z}_{30}$	5	12	11
23	21	30	18	$\mathbb{Z}_{30}$	16	12	17
23	22	24	19	$\mathbb{Z}_{12} \times \mathbb{Z}_2$	18	11	3

7.3 Punktaddisjon

For addisjon har vi kommet fram til at vi vil bruke Algoritme 8 som er en implementasjon av formel 5.57.

Dersom man utfører alle operasjonene i denne formelen uten gjenbruk av midlertidige resultater, vil dette kreve 6 kvadreringer, 12 multiplikasjoner og 3 subtraksjoner. Men vi ser at ved å gjenbruke noen mellomregninger kan vi komme ned i bare 12 multiplikasjoner og 3 subtraksjoner.

Som for Weierstrassform kan man også spare noen operasjoner på å benytte blandede koordinater. Her sparer vi 2 multiplikasjoner dersom den ene z-verdien er lik 1. Man kan da sløyfe multiplikasjonene i linje 2 og 7 i Algoritme 8.

Ifølge artikkelen [5] er det mulig å utføre disse parallelt i tre separate løp, og bare kombinere resultatene fra disse ved hjelp av tre subtraksjoner til slutt. Dette kan utnyttes på systemer hvor man har mulighet til utføre flere operasjoner parallelt. Vi har ikke implementert dette siden vi opererer med et system som ikke gir mulighet for parallell utføring av operasjoner. Addisjon av to punkter $\mathbf{P} + \mathbf{Q} = \mathbf{R}$ hvor $\mathbf{P} = (x_1 : y_1 : z_1), \mathbf{Q} = (x_2 : y_2 : z_2), \mathbf{R} = (x_3 : y_3 : z_3)$ og $\mathbf{P}, \mathbf{Q}, \mathbf{R} \in E_D(\mathbb{K})$ kan utføres i tre separate løp på følgende måte:

$$\begin{array}{lll} \lambda_1 = y_1 x_2 & \lambda_2 = x_1 y_2 & \lambda_3 = x_1 z_2 \\ \lambda_4 = z_1 x_2 & \lambda_5 = z_1 y_2 & \lambda_6 = z_2 y_1 \\ s_1 = \lambda_1 \lambda_6 & s_2 = \lambda_2 \lambda_3 & s_3 = \lambda_5 \lambda_4 \\ t_1 = \lambda_2 \lambda_5 & t_2 = \lambda_1 \lambda_4 & t_3 = \lambda_6 \lambda_3 \\ x_3 = s_1 - t_1 & y_3 = s_2 - t_2 & z_3 = s_3 - t_3 \end{array} \tag{7.3}$$

Til dobling har vi benyttet Algoritme 9 som er basert på formel 5.53.

Istedenfor å benytte funksjonen `power()` til å beregne x^3, y^3 og z^3 bruker vi en kvadrering og en multiplikasjon, siden det er raskere ifølge Tabell 6.1.

Som for addisjon kan man beregne $[2]\mathbf{P} = \mathbf{R}$, hvor $\mathbf{P} = (x_1 : y_1 : z_1), \mathbf{R} = (x_2 : y_2 : z_2)$ og $\mathbf{P}, \mathbf{R} \in E_D(\mathbb{K})$ i tre separate løp:

$$\begin{array}{lll} \lambda_1 = x_1 2 & \lambda_2 = y_1 2 & \lambda_3 = z_1 2 \\ \lambda_4 = x_1 \lambda_1 & \lambda_5 = y_1 \lambda_2 & \lambda_6 = z_1 \lambda_3 \\ \lambda_7 = \lambda_5 - \lambda_6 & \lambda_8 = \lambda_6 - \lambda_4 & \lambda_9 = \lambda_4 - \lambda_5 \\ x_2 = y_1 \lambda_8 & y_2 = x_1 \lambda_7 & z_2 = z_1 \lambda_9 \end{array} \tag{7.4}$$

Tabell 7.3 viser antall basisoperasjoner for addisjons- og doblingsalgoritmene som vi har implementert.

Bemerkning 8: Vi har benyttet normale projektive koordinater og ikke Jacobisk projektive koordinater som vi brukte for Weierstrassform.

Disse algoritmene kan bare brukes på kurver over kropper med karakteristikk $p \equiv 2 \pmod 3$. I karakteristikk $p \equiv 1 \pmod 3$ vil man kunne få galt svar hvis man regner med tretorsjonspunkter.

Både addisjon og dobling av punkter på Hesseform er uavhengig av D.

Algoritme 8 Projektiv punktaddisjon i Hesseform

INN: To punkter $\mathbf{P} = (x_1 : y_1 : z_1)$ og $\mathbf{Q} = (x_2 : y_2 : z_2)$ på samme elliptiske kurve

UT: $\mathbf{P} + \mathbf{Q} = (x_3 : y_3 : z_3)$

1: $h_1 \leftarrow x_1;\ h_2 \leftarrow y_1;\ h_3 \leftarrow z_1;\ h_4 \leftarrow x_2;\ h_5 \leftarrow y_2;\ h_6 \leftarrow z_2$
2: $h_7 \leftarrow h_1 \cdot h_6$
3: $h_1 \leftarrow h_1 \cdot h_5$
4: $h_5 \leftarrow h_3 \cdot h_5$
5: $h_3 \leftarrow h_3 \cdot h_4$
6: $h_4 \leftarrow h_2 \cdot h_4$
7: $h_2 \leftarrow h_2 \cdot h_6$
8: $h_6 \leftarrow h_2 \cdot h_7$
9: $h_2 \leftarrow h_2 \cdot h_4$
10: $h_4 \leftarrow h_3 \cdot h_4$
11: $h_3 \leftarrow h_3 \cdot h_5$
12: $h_5 \leftarrow h_1 \cdot h_5$
13: $h_1 \leftarrow h_1 \cdot h_7$
14: $y_3 \leftarrow h_1 - h_4;\ x_3 \leftarrow h_2 - h_5;\ z_3 \leftarrow h_3 - h_6$

Algoritme 9 Projektiv punktdobling i Hesseform

INN: Et projektivt punkt $\mathbf{P} = (x_1 : y_1 : z_1)$ på en elliptisk kurve

UT: $\mathbf{R} = [2]\mathbf{P} = (x_2 : y_2 : z_2)$

1: $h_1 \leftarrow x_1^2$
2: $h_1 \leftarrow h_1 \cdot x_1$
3: $h_2 \leftarrow y_1^2$
4: $h_2 \leftarrow h_2 \cdot y_1$
5: $h_3 \leftarrow z_1^2$
6: $h_3 \leftarrow h_3 \cdot z_1$
7: $h_4 \leftarrow h_3 - h_1$
8: $h_4 \leftarrow h_4 \cdot y_1$
9: $h_5 \leftarrow h_2 - h_3$
10: $h_5 \leftarrow h_5 \cdot x_1$
11: $h_6 \leftarrow h_1 - h_2$
12: $h_6 \leftarrow h_6 \cdot z_1$
13: $x_2 \leftarrow h_4,\ y_2 \leftarrow h_5,\ z_2 \leftarrow h_6$

Tabell 7.3: Antall basisoperasjoner i Hesseform

Operasjon	Addisjon	Addisjon med bl.	Dobling
Kvadrering	0	0	3
Multiplikasjon	12	10	6
Subtraksjon	3	3	3

Eksempel 9: Vi vil i dette eksempelet se på Hessekurven $E_4 : x^3+y^3+z^3 = 4xyz$ over kroppen $\mathbb{K} = \mathbb{F}_5$, vi bruker her standard projektive koordinater.

Fra Tabell 7.1 ser vi at kurven er supersingulær fordi j-invarianten er 0. Vi ser også at kurvens orden er $\#E_4(\mathbb{K}) = 6$. Kurven er også isomorf med Weierstrasskurven $E_{0,1} : y^2 = x^3 + 1$.

I Tabell 7.4 følger en addisjonstabell som er generert med vår LiDIA-baserte implementasjon av addisjonsformelen for punkter på Hessekurver.

Tabell 7.4: Addisjonstabell for kurven $E_4 : x^3 + y^3 + z^3 = 4xyz$ over kroppen $\mathbb{K} = \mathbb{F}_5$.

+	(0:4:1)	(1:4:1)	(4:0:1)	(4:1:1)	(4:4:1)	(4:1:0)
(0:4:1)	(4:1:0)	(4:4:1)	(4:1:0)	(1:4:1)	(4:1:1)	(0:4:1)
(1:4:1)	(4:4:1)	(0:4:1)	(4:1:1)	(4:1:0)	(4:0:1)	(1:4:1)
(4:0:1)	(4:1:0)	(4:1:1)	(4:1:0)	(4:4:1)	(1:4:1)	(4:0:1)
(4:1:1)	(1:4:1)	(4:1:0)	(4:4:1)	(4:0:1)	(0:4:1)	(4:1:1)
(4:4:1)	(4:1:1)	(4:0:1)	(1:4:1)	(0:4:1)	(4:1:0)	(4:4:1)
(4:1:0)	(0:4:1)	(1:4:1)	(4:0:1)	(4:1:1)	(4:4:1)	(4:1:0)

En utskrift av en slik addisjonstabell kan avsløre de fleste feilene i implementasjonen. Eksempelvis skal et punkt bare opptre en gang i samme rad eller kolonne, dette er en konsekvens av at punktene $E_4(\mathbb{K})$ er en gruppe. Videre kan vi sjekke at gruppen $E_4(\mathbb{K})$ har et identitetselement, det vil si at et punkt $\mathbf{P} \in E_4(\mathbb{K})$ som har egenskapen at $\mathbf{P} + \mathbf{Q} = \mathbf{Q}$, $\forall \mathbf{Q} \in E_4(\mathbb{K})$. I denne gruppen er punktet $(4 : 1 : 0)$ identitetselement og omtales som punktet i uendelig, som har symbolet $\mathcal{O}$.

Vi ser også at kurven har tre tretorsjonspunkter $(0 : 4 : 1)$, $(4 : 0 : 1)$ og $(4 : 1 : 0)$, det vil si punkter hvor en av koordinatene er null. Dette stemmer med avsnitt 5.6.2. Undergruppene generert av tretorsjonspunktene har orden 3, dermed må $E_4(\mathbb{K}) \simeq \mathbb{Z}_3 \times \mathbb{Z}_2$, for mer informasjon se [13] og [12].

Når man skal sjekke om tabellen gjenspeiler det faktum at $E_4(\mathbb{K})$ er lukket under punktaddisjon, bør man tenke over at punktene som står oppført i addisjonstabellen bare er representanter fra ekvivalensklasser. I dette eksempelet har vi valgt å bruke representanter som har 1 i z-koordinaten dersom $z \neq 0$ eller 1 i y-koordinaten dersom $z = 0$. Dette gir oss et entydig valg av representant, og vi vil kunne avsløre om tabellen inneholder punkter som ikke er med i $E_4(\mathbb{K})$.

7.4 Generering av tilfeldige punkter

Man har ofte behov for å finne et tilfeldig punkt på en elliptisk kurve. På kurver med lav karakteristikk er det gjennomførbart å teste alle punkter for å sjekke om de er på den elliptiske kurven. Dersom vi har en kurve over $\mathbb{F}_p$ vil vi da måtte sjekke p^2 punkter, dersom man vil finne alle punktene på kurven. Dette går greit dersom p er liten, men det er ikke gjennomførbart for store p-verdier. Dersom vi opererer med store p-verdier må vi istedenfor velge en x- eller y-verdi, sette inn i ligningen for den elliptiske kurven, og løse ut den andre variabelen.

Ifølge Hasses teorem vil $\#E(\mathbb{F}_p)$ ligge svært nær p for store p-verdier, dermed vil ca. halvparten av x-verdiene vi gi opphav til en y-verdi. Det er også kjent at

x-verdier med løsning er omtrent uniformt fordelt, se [13].

En rimelig måte å finne et tilfeldig punkt på en kurve er derfor å velge en tilfeldig x-verdi, og prøve å løse ligningen for denne x-verdien med hensyn på y. Dersom dette ikke går, velger man en ny x-verdi og prøver på nytt. Vi har sett to måter å velge en ny x-verdi på dersom den første verdien ikke gir løsning; den første løsningen er å velge en ny tilfeldig x-verdi, se [13], den andre er å øke x med 1 og prøve på nytt, se [2].

Vi har laget et program som velger en tilfeldig x-verdi og deretter sjekker om ligningen har en løsning, ved å sjekke om ligningen er et irredusibelt polynom. Dersom den ikke har løsning, øker man x med 1 og prøver på nytt. For å sjekke om ligningen har løsning, og eventuelt løse ligningen har vi benyttet LiDIA-funksjonene `prob_irred_test()` og `find_root()` som finnes i pakken `LiDIA/Fp_polynomial.h`.

7.5 Omforming av kurver

Det kan ofte være nyttig å finne en kurve med samme j-invariant på en annen form, grunnen til dette kan være at vi allerede har metoder for å undersøke kurver på den ene formen. I vårt arbeid har vi bruk LiDIA, som har innebygde metoder for å undersøke Weierstrasskurver. Vi er derfor spesielt interessert i å finne Weierstrasskurver med samme j-invariant, slik at vi kan bruke LiDIA til å undersøke egenskaper ved kurven. Vi vil derfor beskrive hvordan man kan finne slike ekvivalente kurver.

7.5.1 Fra Hesseform til Weierstrassform

Vi vil starte med å se på hvordan man kan finne en ekvivalent Weierstrasskurve til en Hessekurve.

Teorem 6. *Hessekurven gitt ved likning 5.28 er birasjonalt ekvivalent med Weierstrasskurven*

$$y^2 = x^3 - D\left(\frac{D^3}{3} + 72\right)x + 2\left(\frac{D^6}{27} - 20D^3 - 216\right) \tag{7.5}$$

under transformasjonene

$$(x_h, y_h) = \left(\eta\left(x_w + D^2\right), -1 + \eta\left(\frac{D^3}{9} - \frac{D}{3}x_w - 12\right)\right) \tag{7.6}$$

og

$$(x_w, y_w) = (-D^2 + \xi x_h, 3\xi(y_h - 1)) \tag{7.7}$$

hvor

$$\eta = \frac{54\left(D^3 - 27\right)\left(3y_w + D^3 - 3Dx_w - 108\right)}{3^6\left(x_w + D^2\right)^3 + \left(D^3 - 3Dx_w - 108\right)^3} \tag{7.8}$$

og

$$\xi = \frac{4\left(D^3 - 27\right)}{3Dx_h + 9y_h + 9} \tag{7.9}$$

Her betegner vi punkter på Weierstrasskurven med (x_w, y_w)*, og punkter på Hessekurven med* (x_h, y_h)*.*

Bevis. Beviset for dette teoremet finnes i [3] side 3, og baserer seg på Nagells reduksjon [18] side 115. Legg merke til at D ikke har samme betydning i det nevnte beviset som her. □

Beregningene som er beskrevet i Teorem 6 skal utføres i kroppen som kurven er definert over. For vår implementasjon vil dette bety at alle beregninger skal utføres med klassen `gf_element`.

Vi har også benyttet en annen omformingsmetode, som står beskrevet i [4] side 11, denne metoden gir samme a-verdi, men gir den inverse b verdien i forhold til metoden som er beskrevet i Teorem 6. Det viser seg at de to kurvene vi får har samme j-invariant, og er derfor er isomorfe over $\overline{\mathbb{K}}$ i følge Teorem 2. Grunnen til at de har samme j-invariant, er at formelen for j-invarianten til en Weierstrasskurve, formel 5.5, bare inneholder b^2, og er derfor ikke avhengig av fortegnet til b.

Bemerkning 10: Vi bemerker at omformingen forutsetter at punktene som skal omformes er oppgitt i affine koordinater, dette medfører at vi ikke kan bruke formelen til å omforme punkter som har $z = 0$. For Hessekurver vil vi spesielt bemerke at det bare skjer når vi regner med kurver over kropper av typen $\mathbb{F}_p$ hvor $p \equiv 2 \pmod 3$, som nevnt i avsnitt 5.6.2.

7.5.2 Fra Weierstrassform til Hesseform

Vi har ikke funnet noen formel som kan brukes til å regne om fra Weierstrassform til Hesseform. Vi baserer oss på å bruke j-invarianten til Hesse- og Weierstrassformen, til å finne en omregningsmetode fra Weierstrassform til Hesseform. Vi har fra Teorem 2 at to kurver er isomorfe over $\overline{\mathbb{K}}$ dersom de har samme j-invariant. Dermed kan sette opp en likning ved å sette at j-invarianten på Hesseform skal være lik j-invarianten på Weierstrassform.

Fra avsnitt 5.29 har vi at j-invarianten til en Hessekurve E_D kan beregnes med følgende formel.

$$j(E_D) = -\frac{D^3(D^3+216)^3}{-D^9+81D^6-2187D^3+19683}$$

For Weierstrasskurven $E_{a,b}$ har vi at:

$$j(E_{a,b}) = -1728\frac{(4a)^3}{\Delta}$$

Vi vil nå finne D, det vil si en Hessekurve med samme j-invariant ved å løse ligningen:

$$j(E_D) = E_{a,b}$$

med hensyn på D.

Vi setter $x = D^3$ og $k = j(E_{a,b})$, vi får da følgende likning:

$$k(81x^2 - 2187x - x^3 + 1968) + x(x+216)^3 = 0$$

Vi ekspanderer denne ligningen og løser den ved hjelp av funksjoner i klassen *Fp_polynomial* som er en del av LiDIA. For å teste om ligningen har løsning har

vi brukt medlemsfunksjonen *prob_irred_test(f)* , deretter brukte vi funksjonen *find_roots(f)* til å finne løsningen.

$$x^4 + (648 - k)x^3 + (81k + 139968)x^2 + (10077696 - 2187k)x + 19683k = 0$$

Til slutt løser vi følgende likning:

$$D^3 - x = 0$$

Dersom vi setter inn D^3 for x, ser vi at dette er en tolvtegrads likning. Dette betyr at vi kan finne maksimalt 12 Hessekurver med samme j-invariant over kroppen $\mathbb{K}$. Disse kurvene er tvister til hverandre.

Eksempel 11: Her følger et eksempel på på bruk av transformasjonen angitt i Teorem 6. Vi tar utgangspunkt i kurven $x^3 + y^3 + z^3 = 4xyz$ over $\mathbb{F}_5$ som i følge Tabell 7.1 er isomorf med Weierstrasskurven $y^2 = x^3 + 4$, opplysningen om a og b i Tabell 7.1 er beregnet med formelene som er beskrevet i Teorem 6. Vi har i forbindelse med genereringen av Tabell 7.1 og Tabell 7.2 sjekket at disse kurven er isomorfe.

I Tabell 7.5 vises vi hvordan punktene på Hessekurven $x^3 + y^3 + z^3 = 4xyz$ over $\mathbb{F}_5$, transformeres over på den birasjonalt ekvivalente Weierstrasskurven $y^2 = x^3 + 4$, ved hjelp av transformasjonen gitt i Teorem 6. Vi bemerker her at det øverste og nederste punktet i tabellen ikke lar seg transformere, fordi det oppstår et "dele på null" problem under transformasjonen.

Tabell 7.5: Tabellen viser alle punktene på kurven $x^3 + y^3 + z^3 = 4xyz$ over $\mathbb{F}_5$, og tilsvarende punkter på den birasjonalt ekvivalente Weierstrasskurven $y^2 = x^3 + 4$.

Hessepunkt	Weierstrasspunkt
(0:4:1)	-
(1:4:1)	(3:1:1)
(4:0:1)	(0:3:1)
(4:1:1)	(1:0:1)
(4:4:1)	(3:4:1)
(4:1:0)	-

Bemerkning 12: Etter at vi har utført tester i LiDIA med forskjellige små kurver fra Tabell 7.1 og Tabell 7.2, finner vi at tretorsjonspunkter som oftest ikke blir avbildet på seg selv når vi transformerer fram og tilbake. Alle andre punkter blir avbildet på seg selv. Det ser også ut til at alle andre punkter en tretorsjonspunktene lar seg transformere fra Hessekurven til Weierstrasskurven, men for tretorsjonspunktene ser det ut til at det alltid er noen som ikke lar seg transformere fra Hesseform til Weierstrassform, fordi de generer et "dele på null" problem.

7.6 Beskyttelse mot sidekanalangrep

Algoritme 8 er egentlig bare tenkt brukt til addisjon av to forskjellige punkter, men i [3] forklares og bevises det hvordan man også kan bruke denne algoritmen

både til dobling og subtraksjon. Dersom man bruker samme algoritme til å utføre alle basisoperasjonene, vil man beskytte seg bedre mot sidekanalangrep. Denne metoden senker ytelsen noe, fordi addisjon tar lengre tid en dobling, men i [3] hevder de at det likevel er 33% bedre enn det beste alternativet (Jacobi parameterization).

Teorem 7. *La punktet* $\mathbf{P} = (x, y, z)$ *være et punkt på en Hessekurve* $E(\mathbb{K})$*, da kan* $2\mathbf{P}$ *beregnes ved å addere punktene* (z, x, y) *og* (y, z, x) *med Algoritme 8.*

Bevis. Se [3] side 6. □

Denne algoritmen krever 14 addisjoner, i motsetning til den normale doblingsalgoritmen som krever 6 multiplikasjoner og 3 kvadreringer.

Subtraksjon kan utføres ved å bytte om x og y-koordinatene til det ene punktet. La punktene $\mathbf{P} = (x_1, y_1, z_1)$ og $\mathbf{Q} = (x_2, y_2, z_2)$ være to punkter på den elliptiske kurven $E(\mathbb{K})$, vi kan da beregne $\mathbf{P} - \mathbf{Q}$ ved å addere punktene (x_1, y_1, z_1) og (y_2, x_2, z_2) med Algoritme 8.

7.7 Punkter på små elliptiske kurver på Hesseform

For kurver på Hesseform benytter vi en litt annen strategi enn for Weierstrassform, her prøver vi med alle kombinasjoner av $z = 1$, og x- og y-verdier mellom 0 og kroppens karakteristikk. Vi setter disse inn i formelen for kurven og ser om de tilfredstiller denne. Hvis de gjør det, legges de til i listen med punkter. Vi gjør tilsvarende hvor vi setter $y = 1$ og $z = 0$, og prøver alle verdier av x mellom 0 og kroppens karakteristikk. Punktet $\mathcal{O}$ vil være inneholdt i denne kategorien.

En enkel test for å se om antall punkter vi har funnet kan stemme, er å se om det ligger innenfor Hasseintervallet som vi omtalte i avsnittet om Hasses Teorem 5.5.1.

7.8 Kryptografisk sterke kurver på Hesseform

LiDIA har en egen pakke **gec** som er beregnet på å generere kryptografisk sterke kurver på kort Weierstrassform, og vi trenger tilsvarende funksjonalitet for kurver på Hesseform. Vi har derfor laget et program som gjør dette.

Vi ser to muligheter for å finne kryptografisk sterke kurver på Hesseform; å først finne en kurve på kort Weierstrassform ved hjelp av gec-pakken, og deretter konvertere denne til Hesseform, eller man kan lage en algoritme for å finne en kurve på Hesseform, og deretter konvertere den til kort Weierstrassform for å teste om kurvens orden er et stort primtall multiplisert med 3. Alle kurver på Hesseform har tretorsjonspunkter, altså punkter med orden 3, men vi ønsker at de andre punktene skal ha høyest mulig orden, og når kurvens orden er $3p$, vil punktenes orden være 3, p eller $3p$. Det vil si at de genererer undergrupper med orden 3, p, eller de genererer hele gruppen. Vi vet at alle punktene untatt tretorsjonspunktene har minst orden p, og dersom p er et tilstrekkelig stort primtall, anser vi kurven for å være egnet til bruk i kryptografi siden man lett kan teste at punktene man opererer med ikke er tretorsjonspunkter.

Vi har valgt å generere en tilfeldig Hessekurve, som vi omformer til Weierstrassform, slik at vi kan bruke den innebygde funksjonen `compute_group_order()` fra pakken **eco** i LiDIA til å sjekke om ordenen oppfyller kravene våre.

Vi setter nedre og øvre grense for antall bit i lengden til karakteristikken p til 158 og 162. Vi genererer p ved å finne et tilfeldig tall med maksimal verdi lik $2^{upper\_bit\_limit}$. Dersom bitlengden er under minstekravet, forkastes p. Siden p ikke nødvendigvis er et primtall setter vi p til det neste primtallet større en p, med LiDIAs innebygde funksjon `next_prime()`. Så sjekker vi om $2 \equiv p \pmod 3$, dersom det ikke er tilfelle avbryter vi undersøkelsen. Ellers finner vi en tilfeldig D som er mellom 0 og p. Så bruker vi funksjonen vår `hesse_to_weierstass()` for å regne om D til a og b, vi oppretter så en elliptisk kurve på projektiv form med parameter a og b. Til slutt kontrollerer vi at Weierstrasskurven har orden på formen $3p$.

Selv om Weierstrass- og Hessekurvene er birasjonalt ekvivalente, er det prinsipielt ikke garantert at de har samme orden. For å kontrollere om Hessekurven har samme orden som Weierstrasskurven, vil vi multiplisere et tilfeldig punkt $\mathbf{P}$ på Hessekurven med $3p$.

Dersom $[3p]\mathbf{P} = \mathcal{O}$, vet vi at punktet har orden 3, p eller $3p$. Vi sjekker at punktet ikke har orden 3. Vi sitter nå med et punkt $\mathbf{P}$ som har orden delelig på p, vi vet dermed at p også deler ordenen til Hessekurven, siden ordenen til $\mathbf{P}$ må dele ordenen til kurven i følge Lagranges teorem. Hvis $p > 4\sqrt{3p}$, er vi garantert at $3p$ er ordenen til den elliptiske kurven E, siden Hasseintervallet ikke kan inneholde andre multipler av p enn $3p$.

I forbindelse med kurver til bruk i kryptografi vil $3 \ll p$, og kravet om at $p > 4\sqrt{3p}$ vil være oppfylt. Sansynligheten for å finne et punkt som tilhører en undergruppe med orden 3, vil også være svært liten.

Eksempler på kurver generert med denne metoden finnes i Tillegg D.

7.9 Oppsummering

Vi har sett på hvordan man kan implementere addisjons- og doblingsalgoritmer for elliptiske kurver på Hesseform på en effektiv måte. Vi ser at disse algoritmene er en god del enklere enn de tilsvarende for kort Weierstrassform. Algoritmene er symmetriske med hensyn på x, y og z, og er også mer parallelliserbare enn tilsvarende for kort Weierstrassform.

Siden LiDIA ikke har støtte for kurver på Hesseform, har vi også implementert en rekke støttefunksjoner for å finne kurver og punkter som er anvendbare til kryptografi, og for å konvertere kurver og punker mellom kort Weierstrassform og Hesseform.

Kapittel 8

Testresultater og drøfting

8.1 Grunnleggende kroppsoperasjoner i LiDIA

8.1.1 Diskusjon av testoppsett

LiDIA har ikke innebygget støtte for multipresisjonsaritmetikk[1], men benytter funksjonalitet fra et eksternt multipresisjonsaritmetikk-bibliotek. Man kan velge mellom flere forskjellige slike biblioteker, vi har valgt å bruke GNU MP, se [19], fordi det på hjemmesidene til LiDIA er indikert at LiDIA fungerer godt sammen med dette biblioteket. Vi kan ikke utelukke at utfallet av testene kunne blitt noe annerledes om vi hadde brukt LiDIA sammen med ett annet MPA-bibliotek.

Vi antar at operasjoner på punkter på elliptiske kurver hadde gått raskere om vi hadde basert implementasjonen direkte på GNU MP eller et annet multipresisjonsaritmetikk-bibliotek. Vi har likevel valgt å ikke gjøre dette, siden vi tror at dette ikke hadde ført til store endringer i forholdet mellom Hesseform og kort Weierstrassform med hensyn til tidsforbruk.

Man kunne også ha implementert hele eller deler av operasjonene i assembly, men dette ville vært et mye større og mer tidkrevende prosjekt, og vi tror gevinsten ville vært omtrent likt fordelt mellom de to formene, og dermed ikke forandret på vår konklusjon.

Vi har kompilert testprogrammene våre med og uten optimaliseringsparameteren -O2 med C++-kompilatoren g++[2], og har ikke funnet at dette påvirker resultat i noen betydelig grad, siden LiDIA-biblioteket er kompilert med optimalisering -O2, dermed er de tidkrevende funksjonene fra LiDIA og GNU MP kompilert med optimalisering.

Testen av grunnoperasjonene er utført over samme kroppen som testene av punktoperasjonene, slik at resultatene fra de forskjellige testene er sammenlignbare.

Testene er utført over to forskjellige kropper $\mathbb{F}_p$ hvor p er et primtall p_{160} med lengde 160 bit, og et annet primtall p_{240} med lengde 240 bit, dette er av en størrelsesorden som gjør den egnet for bruk i kryptografi basert på elliptiske

[1] Multipresisjonsaritmetikk består av algoritmer og datastrukturer tilpasset aritmetikk med vilkårlig store tall.

[2] Kompilatoren g++ er en del av GCC, se [20].

kurver. Testene er ellers utført som beskrevet i avsnitt 6.4.

p_{160} =1224753567915253525600877180059052116597297173971

p_{240} =1692071621110286699141341896411670096195987131713624502236260775181406103

Etter å ha utført noen innledendende tester fant vi at kjøretiden til en del av operasjonene var sterkt avhengig av hvilke kroppselementer de ble utført på, dette gjaldt spesielt addisjon og subtraksjon. Forholdet mellom den største og minste kjøretiden var for addisjon og subtraksjon 1.8. Vi har derfor utført hver test 100 ganger på forskjellige kroppselementer, og beregnet gjennomsnittsverdier av alle kjøretidene.

I Tabell 8.1 følger resultatet fra tester av basisoperasjoner på klassen `gf_element`, som i LiDIA representerer et element i kroppen $\mathbb{F}_p$.

Tabell 8.1: Kjøretider for operasjoner i klassen `gf_element` på elementer med 160 bit og 240 bit.

Operasjon	Tid(160bit)	Tid(240bit)	Forkortelse
add(c,a,b)	0.46μs	0.51μs	A
subtract(c,a,b)	0.44μs	0.51μs	SU
negate(c,a)	0.52μs	0.57μs	
a.multiply_by_2()	0.47μs	0.52μs	M2
a.divide_by_2()	23.76μs	28.34μs	I2
multiply(c,a,b)	2.57μs	5.50μs	M
divide(c,a,b)	39.72μs	63.08μs	
invert(c,a)	33.39μs	53.64μs	I
square(c,a)	2.33μs	4.84μs	SQ
power(c,a,3)	11.46μs	16.80μs	-

8.1.2 Kommentarer til resultater

I følge våre tester med 160 bits tar invertering 13 ganger så lang tid som multiplikasjon, og 11.5 ganger så lang tid med 240 bits. Det er litt mindre enn forventet, for eksempel antyder Joye og Quisquater i [3] at divisjon i kropp med karakteristikk $p > 3$ tar samme tid som 23 multiplikasjoner. I litteraturen er vanlig å bare telle multiplikasjoner, kvadreringer og divisjoner, men testene våre for 160 bits viser at en multiplikasjon tar ca. like lang tid som fem addisjoner eller subtraksjoner. Dette viser at man må være forsiktig med å bare telle antall multiplikasjoner i algoritmer som inneholder mange andre operasjoner i tillegg til multiplikasjoner og divisjoner.

I punktdoblingsformelen for Hesseform må vi opphøye et `gf_element` i tredje, av den grunn har vi testet funksjonen `power()` med verdien 3 som eksponent. Det viser seg at det er mye mer effektivt å bruke en kvadrering og en multiplikasjon, i stedet for et kall til funksjonen `power()`. Det tyder på at `power()`-funksjonen er optimalisert for større eksponenter.

Til slutt vil vi nevne at `square()` er raskere en `multiply()`, og det er mer effektivt å bruke `invert()` etterfulgt av `multiply()` enn å bruke `divide()`.

I neste avsnitt vil vi bruke målingene I Tabell 8.1 som grunnlag for å beregne teoretiske kjøretider. De beregnede kjøretidene er oppgitt i Tabell 8.2 og Tabell 8.3, og er basert på antall basisoperasjoner og målte kjøretider.

8.2 Grunnleggende punktoperasjoner

8.2.1 Valg av algoritmer for Weierstrassform

Siden vi i utgangspunktet benyttet LiDIA, var det nærliggende å undersøke om LiDIAs algoritmer kunne benyttes i testene våre. Vi fant ut at LiDIAs algoritmer stemmer, med noen få unntak, godt overens med de algoritmene som vi fant flere steder i litteraturen, blant annet i [13] og [21].

Vi kopierte LiDIAs algoritmer for addisjon og dobling av punkter i Jacobisk projektiv Weierstrassform, og omskrev dem i samme stil som Hessealgoritmene for å få mer sammenlignbare resultater.

I LiDIA sin implementasjon av punktaddisjon med Jacobiske projektive koordinater brukes funksjonen `divide_by_2()`, dette funksjonskallet kan erstattes med fem kall til funksjonen `multiply_by_2()`. Dette er en god idé, siden et funksjonskall til `divide_by_2()` tar ca. 10 ganger så lang tid som fem funksjonskall til funksjonen `multiply_by_2()`, og dermed fikk vi redusert den teoretiske kjøretiden for denne algoritmen fra 68.4 til 46.1 mikrosekunder med kurve over kropp med karakteristikk med 160 bits lengde. Vi har gjort denne forandringen i vår implementasjon av punktaddisjon med Jacobisk projektive koordinater. Dette er den mest vesentlige forandringen i forhold til implementasjonen i LiDIA. Vi tok også ut alle deklarasjoner av midlertidige variabler, og lot dem være globale. Dette hadde meget stor innvirkning på resultatet, og dette er noe av grunnen til at LiDIAs innebygde algoritmer kommer dårligere ut enn våre i testen av addisjons- og doblingsalgoritmer.

I våre versjoner er det liten forskjell mellom teoretisk kjøretid og målt tid, mens i LiDIAs innebygde versjoner er det mer forskjell. Vi tror det er særlig tre viktige årsaker til dette; opprettelse av midlertidige variabler, flere tester for å gjøre funksjonene mer generelle, og bruk av indirekte funksjonskall via funksjonspekere.

Som vi har nevnt i bemerkning 7, kan man spare 2 addisjoner i dobling for Weierstrassform dersom man har $a = -3$, se side 60 i [13], men vi har valgt å ikke innføre denne begrensningen på hva slags kurver som kan benyttes.

Koordinater

Ved å benytte projektive koordinater slipper man å benytte divisjon i addisjons- og doblingsalgoritmene, på bekostning av å øke antall multiplikasjoner. Om dette er tidsbesparende er avhengig av forholdet mellom tidsforbruket til multiplikasjon og divisjon over endelige kropper. Basert på observasjonen at multiplikasjon er mye raskere enn divisjon over endelige kropper vil det derfor som regel gi en positiv effekt å benytte projektive koordinater.

Det finnes flere varianter av projektive koordinater å velge mellom, men vi har valgt å benytte Jacobiske projektive koordinater, slik som i LiDIA.

Jacobiske koordinater er optimalisert med hensyn på dobling, se [22], og er litt dårligere for addisjon, det skiller 2 multiplikasjoner i begge tilfeller. Siden det som regel er mange flere doblinger enn addisjoner i punktmultiplikasjon, er det

som oftest best å benytte denne formen når addisjons- og doblingsalgoritmene først og fremst er beregnet på å inngå i punktmultiplikasjon. Det finnes flere metoder for å redusere antall addisjoner i punktmultiplikasjon, men det finnes ingen kjente metoder for å redusere antall doblinger, se [22].

Dersom punktet man skal multiplisere er kjent på forhånd, kan man forhåndsberegne og lagre en del punkter, og dermed forbedre hastigheten på punktmultiplikasjon. I tillegg kan man spare noen operasjoner i addisjonsformelen på å lagre punktene med $z = 1$. Dette omtales som bruk av blandede koordinater. Vi har implementert versjoner av addisjonsalgoritmene både med og uten blandede koordinater, og vi antar at kun ett av punktene har $z = 1$. I praksis er det svært usannsynlig at mer enn det forhåndslagrede punktet vil ha $z = 1$.

Når vi bruker forhåndslagring med blandede koordinater, er addisjon i Jacobiske og normale projektive koordinater like raskt.

Ved punktmultiplikasjon uten forhåndsberegninger, blir det alltid utført flest doblinger i forhold til addisjoner, derfor vil besparelsen i doblingen være større enn tapet i addisjonene, og det vil derfor lønne seg å bruke Jacobiske koordinater.

Dersom vi bruker forhåndsberegninger, vil vi lagre alle punkter med $z = 1$, og vi taper derfor ikke noe på å bruke Jacobiske koordinater.

I artikkelen [16] diskuteres det hvordan man kan effektivisere addisjons- og doblingsalgoritmene ved å bruke varierende redundante blandede koordinater, for eksempel har de brukt $(x : y : z : az^4)$. Den redundante koordinaten her gjør at man kan gjenbruke mellomregninger. Siden vi ikke har noen tilsvarende metode for Hesseform, har vi valgt å ikke implementere denne metoden.

Konklusjonen blir at det alltid lønner seg å bruke Jacobiske koordinater, med vår strategi for forhåndslagring av punkter. Basert på dette har vi kommet fram til følgende valg av algoritmer for kort Weierstrassform: For addisjon bruker vi Algoritme 3, som står på side 44. Til dobling bruker vi Algoritme 4, som står på side 45.

8.2.2 Valg av algoritmer for Hesseform

Vi har ikke sett på annet enn normale projektive koordinater for Hesseform. Vi tror ikke det er mye å vinne på å bruke andre former for projektive koordinater, fordi ligningen for Hesseform er symmetrisk med hensyn på x, y og z. Vi tror heller ikke det er noe å vinne på å bruke affine koordinater, fordi den projektive formen for addisjon og dobling i utgangspunktet tar mindre tid en en invertering.

Vi vil bemerke at man også kan bruke addisjonsformelen til dobling, dette er mindre effektivt, men man beskytter seg bedre mot sidekanalangrep, som beskrevet i [3].

En stor fordel med Hesseform er at algoritmene for punktdobling og -addisjon er parallelliserbare, med dette har vi ikke utnyttet siden det ikke er støtte for det i LiDIA.

Implementasjonen vi har brukt i testene er basert på artikkelen [3], som er den beste metoden i følge [5] og [3]. For addisjon bruker vi Algoritme 8, som står på side 57. For dobling bruker Algoritme 9, som står på side 57.

8.2.3 Resultater

Vi presenterer her reslutatet fra tester av addisjons- og doblingsfunksjonene som brukes i forbindelse med punktmultiplikasjon. Vi har utført tester av LiDIA

sine affine og Jacobisk projektive funksjoner, vår implementasjon av Jacobisk projektiv Weierstrassform, og projektiv Hesseform. Alle testene er utført over de samme kroppene som i grunnoperasjonstestene i avsnitt 8.1.

Hessekurvene vi bruker i testene har vi generert ved hjelp av metoden vi har beskrevet i avsnitt 7.8, vi har deretter funnet den korresponderende Weierstrasskurven ved hjelp av metoden som er beskrevet i avsnitt 7.5.1.

Figurene 8.1 og 8.2 viser kurven vi har brukt i testene, hver figur inneholder parametre for Hessekurven og den korresponderende Weierstrasskurven.

$$
\begin{aligned}
p &= 1224753567915253525600877180059052116597297173971 \\
\#E(\mathbb{K}) &= 3 \cdot 4082511893050845085336258395399575189669561010 71 \\
D &= 1550842421627942258257328785351007532033094402 42 \\
a &= 1808901272343108614406190635530977964674453038 76 \\
b &= 6387231065610304706782316703719324216503513898 55
\end{aligned}
$$

Figur 8.1: Figuren viser en 160 bits Hessekurve og en birasjonalt ekvivalent Weierstrasskurve.

$$
\begin{aligned}
p &= 1692071621110286699141341896411670096195987131713624502236260775181406103 \\
\#E(\mathbb{K}) &= 3 \cdot 5640238737034288997137806321372233658182706401750080706837978319881127 11 \\
D &= 7024972385738968756927999601141362972273104138207698503475582511209787 49 \\
a &= 4316434741017905318095077050734971433892552281802238768603934945328492 50 \\
b &= 9938907497500547973745707026183472285859717797758236024775615986918871 83
\end{aligned}
$$

Figur 8.2: Figuren viser en 240 bits Hessekurve og en birasjonalt ekvivalent Weierstrasskurve.

For hver av de 100 testene trekker vi to tilfeldige punkter som brukes i testene av operasjonene på affin Weierstrassform, vi trekker to nye punkter som brukes til Jacobisk Weierstrassform, og to som brukes i Hesseform. Metoden som er beskrevet over benyttes både i 160 bits testene og 240 bits testene. LiDIA har en innebygget funksjon `random_point()` i klassen `elliptic_curve<gf_element>`, som vi benytter i testene av kurvene på Weierstrassform. I testene av kurvene på Hesseform benytter vi metoden som er beskrevet i avsnitt 7.4 til å generere tilfeldige punkter.

Funksjonen som vi har brukt til å generere tilfeldige punkter leverer punkter hvor $z = 1$, dermed kan vi bruke punktene som de er når vi skal teste addisjon med blandede koordinater, når vi ikke skal teste med blandede koordinater, har vi doblet punktene før vi bruker dem, slik at $z \neq 1$.

Vi har også sjekket at de innebygde funksjonene i LiDIA for punktaddisjon på Jacobisk projektiv Weierstrassform gir samme resultat som vår egen implementasjon. Resultatet fra 160 bits testene finnes i Figur 8.2, og resultatet fra 240 bits testene finnes i Tabell 8.3.

Ser vi på Tabell 8.2, legger vi merke til at punktaddisjon i affine koordinater er raskere enn addisjon i Jacobiske koordinater, men dette oppveies av at dobling i Jacobiske koordinater er mye raskere enn dobling i affine koordinater.

Tabell 8.2: Kjøretider for addisjons- og doblingsrutiner på den elliptiske kurven over den endelige kroppen med karakteristikk p_{160}.

Op.		Målt	Basisoperasjoner	Beregnet
Add.	LiDIA affin	83.91μs	2M + SQ + I + 6SU	43.5μs
	LiDIA proj.	91.88μs	12M + 4SQ +A+ 7SU+ 2M2+I2	68.4μs
	Weierstrass	52.32μs	12M + 4SQ + 2A + 5SU + 6M2	46.1μs
	Hesse	36.31μs	12M + 3SU	32.16μs
Add. bl.	LiDIA proj.	79.07μs	8M + 3SQ + A + 7SU + M2 + I2	55.32μs
	Weierstrass	38.55μs	8M + 3SQ + A + 5SU + 7M2	33.50μs
	Hesse	31.58μs	10M + 3SU	27.02μs
Dbl.	LiDIA affin	73.36μs	3M + 2SQ + I + A + 3SU + 2M2	48.48μs
	LiDIA proj.	55.26μs	4M + 6SQ + 4A + 3SU + 6M2	30.24μs
	Weierstrass	37.18μs	4M + 6SQ + 4A + 3SU + 6M2	30.24μs
	Hesse	26.1μs	6M + 3SQ + 3SU	23.73μs

Tabell 8.3: Kjøretider for addisjons- og doblingsrutiner, på den elliptiske kurven over den endelige kroppen med karakteristikk p_{240}

Op.		Målt	Basisoperasjoner	Beregnet
Add.	LiDIA affin	114.37μs	2M + SQ + I + 6SU	72.54μs
	LiDIA proj.	143.02μs	12M + 4SQ +A+ 7SU+ 2M2+I2	118.2μs
	Weierstrass	98.71μs	12M + 4SQ + 2A + 5SU + 6M2	92.05μs
	Hesse	72.27μs	12M + 3SU	67.53μs
Add. bl.	LiDIA proj.	116.24μs	8M + 3SQ + A + 7SU + M2 + I2	91.46μs
	Weierstrass	70.78μs	8M + 3SQ + A + 5SU + 7M2	64.18μs
	Hesse	61.58μs	10M + 3SU	56.53μs
Dbl.	LiDIA affin	105.31μs	3M + 2SQ + I + A + 3SU + 2M2	82.90μs
	LiDIA proj.	83.34μs	4M + 6SQ + 4A + 3SU + 6M2	57.73μs
	Weierstrass	65.08μs	4M + 6SQ + 4A + 3SU + 6M2	57.73μs
	Hesse	51.75μs	6M + 3SQ + 3SU	47.97μs

8.3 Multiplikasjon

I dette avsnittet vil vi beskrive testene vi har gjort av punktmultiplikasjonsrutiner. Vi har testet punktmultiplikasjonsfunksjonen i LiDIA for affine og Jacobisk projektive koordinater, og vår punktmultiplikasjonsfunksjon på våre egne implementasjoner av Weierstrass- og Hesseform. Våre egne implementasjoner har vi testet med og uten forhåndsberegninger. I testene med forhåndsberegninger har vi gjort alle doblingene på forhånd, slik at multiplikasjonen bare består i å addere og subtrahere.

Testene er utført på omtrent samme måte som beskrevet i forrige avsnitt, det vil si at vi trekker et tilfeldig punkt for hver testrunde, men vi utførte multiplikasjonen bare 100 ganger, fordi multiplikasjonsrutinen tar mye lenger tid enn operasjonene beskrevet i forrige avsnitt. Vi har utført tester både med og uten blandede koordinater, på samme måte som forklart i forrige avsnitt.

Som en ekstra test på at multiplikasjonsrutinen fungerer som den skal har vi sjekket at $[\#E(\mathbb{K})]\mathbf{P} = \mathcal{O}$.

Vi har tatt utgangspunkt i punktmultiplikasjonsalgoritmen som brukes i LiDIA. Det er en NAF-algoritme, som utnytter at det er nesten like enkelt å subtrahere to punkter som det er å addere to punkter på en elliptisk kurve. Denne algoritmen er beskrevet nærmere i avsnitt 6.9.3. Vi har valgt å benytte samme multiplikasjonsalgoritme både for kort Weierstrassform og Hesseform.

I de kryptografiske protokollene som punktmultiplikasjonen skal brukes i, vil man ofte beregne $[k]\mathbf{P}$ hvor k er et tilfeldig tall mellom 0 og ordenen til kurven. I våre tester har vi valgt å la k være en tilfeldig verdi mellom 0 og ordenen til kurven,$\#E(\mathbb{K})$.

LiDIA har ikke støtte for bruk av forhåndsberegninger i sine multiplikasjonsrutiner, det er grunnen til at noen av feltene i tabellen er blanke.

Siden k velges tilfeldig kan man anta at den binære representasjonen for kurven over kropp med karakteristikk på 160 bit vil inneholde $160/2$ bit som er satt til 1, og SD2 (Signed Digital Representation med Basis 2) representasjonen inneholder $160/3$ bit som er satt til 1 eller -1. Når vi multipliserer uten forhåndsberegninger, krever algoritmen vi har brukt $160 - 1$ doblinger og $160/3$ addisjoner eller subtraksjoner, siden algoritmen vi har brukt baserer seg på SD2-representasjonen til k. Dette har dannet grunnlaget for de teoretiske kjøretidene vi har oppgitt i Tabell 8.4. I disse beregningene har vi brukt de målte kjøretidene for addisjon og dobling, som står i Tabell 8.2. Tilsvarende har vi for verdiene for 240 bit i Tabell 8.5 tatt utgangspunkt i Tabell 8.3. Vi har også antatt at addisjon og subtraksjon tar like lang tid. Beregnet kjøretidene for multiplikasjon med forhåndsberegninger er bare basert på kjøretidene til addisjon med blandede koordinater.

Det finnes multiplikasjonsrutiner som bruker færre forhåndslagrede punkter men likevel er forholdsvis effektive. Hva slags algoritme for forhåndslagring av punkter man vil velge for en praktisk implementasjon, er avhengig av hvor mye minne som er tilgjengelig på det aktuelle systemet.

De målte kjøretidene for multiplikasjon er litt større (1%-6.5%) enn de beregnede. Det var minst avvik for kurvene over kropp med 240 bits karakteristikk, og det er forventet siden selve operasjonene tar lenger tid og det blir prosentvis mindre overhead med funksjonskall. Det er flere andre årsaker til avvikene. Det vil i praksis ta noe mer tid å subtrahere enn å addere, selv om vi har basert våre teoretiske beregninger på at disse operasjonene tar like lang tid. Det blir også

Tabell 8.4: Kjøretider for punktmultiplikasjon på den elliptiske kurven over den endelige kroppen med karakteristikk p_{160}

	Uten forhåndsberegning		Med forhåndsberegning	
Multiplikasjonsrutine	Målt	Beregnet	Målt	Beregnet
LiDIA affin	16.35*ms*	16.14*ms*	-	-
LiDIA Jacobisk	14.58*ms*	13.69*ms*	-	-
Weierstrass Jacobisk	8.91*ms*	8.70*ms*	2.15*ms*	2.06*ms*
Hesse	6.24*ms*	6.09*ms*	1.77*ms*	1.69*ms*

Tabell 8.5: Kjøretider for punktmultiplikasjon på den elliptiske kurven over den endelige kroppen med karakteristikk p_{240}

	Uten forhåndsberegning		Med forhåndsberegning	
Multiplikasjonsrutine	Målt	Beregnet	Målt	Beregnet
LiDIA affin	34.81*ms*	34.32*ms*	-	-
LiDIA Jacobisk	32.14*ms*	31.44*ms*	-	-
Weierstrass Jacobisk	23.72*ms*	23.40*ms*	5.79*ms*	5.66*ms*
Hesse	18.39*ms*	18.17*ms*	5.03*ms*	4.93*ms*

opprettet et objekt av typen `point<gf_element>` i multiplikasjonsalgoritmen som er innebygget i LiDIA, og dette tar også tid.

Uten forhåndsberegninger kommer Hesseform klart best ut, og er målt til å være 30% raskere enn kort Weierstrassform for 160 bit og 20% bedre for 240 bit. Med forhåndsberegninger er fortsatt Hesseform raskere, men her er marginen mindre; Hesseform er her 18% raskere enn kort Weierstrassform for 160 bit og 13% for 240 bit.

Vi ser at forholdet mellom kjøretidene til implementasjonene våre av Hesseform og kort Weierstrassform blir det samme både for de teoretiske og de målte verdiene, både med og uten forhåndsberegning.

I praktiske implementasjoner vil man som oftest ikke forhåndslagre alle punkter, og multiplikasjonsrutinen vil inneholde noen doblinger i tillegg til addisjoner og subtraksjoner. Det er ikke like mye å vinne på å bruke blandede koordinater på Hesseform som på Weierstrassform, men det er relativt stor forskjell på kjøretiden til doblingsfunksjonene. Når man forhåndslagrer færre punkter vil Hesseform derfor komme bedre ut av sammenligningen.

Det er to hovedårsaker til att punktmultiplikasjon tar mer tid for 240 bitt en for 160 bit; basis operasjonene i den underliggende kroppen tar mer tid; det må utføres flere punktaddisjoner og doblinger siden man nå skal multiplisere med et større tall.

Hovedårsaken til at forskjellen i kjøretiden blir mindre for 240 bits testene enn for 160 bits testene, er at kjøretiden til multiplikasjon av elementer i en endelig kropp øker mye raskere enn de andre kroppsoperasjonene. Dermed blir det forskjellen i antall multiplikasjoner som har mest betydning for resultatet.

Basert på dette mener vi at vi kan konkludere med at Hesseform er vel verdt å satse videre på, særlig i systemer som ikke gir muligheter for forhåndsberegning

av punkter.

8.4 Observasjoner om gruppestruktur

I forbindelse med testene genererte vi en oversikt over alle kurver på Hesseform med primtallskarakteristikk mellom 5 og 29. I den forbindelse har vi lagt merke til et par interessante fenomener.

Alle kurver vi har undersøkt over endelige kropper hvor $p \equiv 1 \pmod 3$ har strukturen $\mathbb{Z}_{3a} \times \mathbb{Z}_{3b}$. Disse har 9 tretorsjonspunkter, se [4] side 5, og vi tror dette har en sammenheng med strukturen. De krever også en spesiell variant av addisjonsalgoritmen for å fungere på alle tretorsjonspunktene. Disse punktene er lette å unngå, men det er flere problemer med dem. Et problem er at mange av disse kurvene er isomorfe, slik at det er relativt få forskjellige kurver. Det er uavklart hvordan dette vil være for kurver med høyere karakteristikk.

Vi har observert at kurvene over kropper med $p \equiv 2 \pmod 3$ enten er isomorfe med en syklisk gruppe, eller isomorfe med $\mathbb{Z}_{3a} \times \mathbb{Z}_2$.

Når vi har generert elliptiske kurver på Hesseform til testene har vi brukt følgende kriterier: kurven må være over en kropp med karakteristikk $p \equiv 2 \pmod 3$, og orden til kurven skal være $3m$ hvor m er et primtall. Med denne begrensningen unngår vi både kurvene med 9 tretorsjonspunkter, og de med punkter med orden 2.

Kapittel 9

Konklusjon

Vi har sammenlignet Hesseform og kort Weierstrassform av elliptiske kurver med hensyn til tidsforbruk for de operasjonene som benyttes i kryptografi, det vil si addisjon, dobling og multiplikasjoner av punkter på en elliptisk kurve. Testene er utført på elliptiske kurver over endelige kropper med primtallskarakteristikk.

Våre tester viser at uten forhåndsberegning av punkter er multiplikasjon i Hesseform 20%-30% raskere enn i Weierstrassform. Når man benytter forhåndsberegnede punkter, blir Hesseform 13%-18% raskere. Vi mener at dette er et godt utgangspunkt for å gjøre videre undersøkelser med tanke på bruk i kryptosystemer.

Det er vanlig å bare telle antall multiplikasjoner og divisjoner som et mål for teoretisk hastighet på algoritmer for punktaddisjon og punktdobling, mens man ser bort fra mindre tidkrevende grunnoperasjoner som addisjon, subtraksjon og lignende. I både addisjons- og doblingsalgoritmene for kort Weierstrassform er det en del slike operasjoner som hver for seg tar lite tid, men som til sammen tar mer tid enn en enkelt multiplikasjon. Dette betyr at det ikke alltid er nok å telle antall multiplikasjoner og divisjoner for å finne ut hvilke algoritmer som vil være raskest.

Algoritmene for addisjon og dobling av punkter på Hesseform vil i mange tilfeller relativt enkelt kunne tilpasses til en eksisterende implementasjon av ECC (Elliptic Curve Cryptography). Det er ikke programmeringstekniske eller ytelsesmessige hindringer for at Hesseform av elliptiske kurver kan tas i bruk i kryptosystemer i dag, men man behøver gode og veltestede kurver, og kanskje bør også erfarne kryptanalytikere foreta en grundig gjennomgang med tanke på mulige svakheter ved anvendelse av Hesseform.

En mulighet for videre arbeid er å integrere støtte for kryptografi med elliptiske kurver på Hesseform i et eksisterende kryptosystem som allerede inneholder støtte for kort Weierstrassform, for eksempel i OpenSSL.

Våre tester er begrenset til kurver over kropper med primtallskarakteristikk. Det kunne også være interessant å utføre tilsvarende tester på kurver over kropper med karakteristikk 2^n.

For at en kurve på Hesseform skal kunne benyttes i kryptografi er det viktig at den ikke har strukturelle svakheter som vil svekke sikkerheten i et system hvor den benyttes. Det kunne derfor være nyttig å klarlegge gruppestrukturen til kurver på Hesseform.

Det er viktig å vite ordenen til en elliptisk kurve som skal benyttes i kryp-

tografi. Det finnes i dag gode metoder for å beregne denne for kurver på Weierstrassform. En mulighet for videre arbeid er å se på om noen av disse metodene kan tilpasses til kurver på Hesseform.

Bibliografi

[1] Menez, van Oorschot, and Vanstone, *Handbook of Applied Cryptography.* CRC Press, 1996.

[2] M. Rosing, *Implementing Elliptic Curve Cryptography.* Manning, 1999.

[3] M. Joye and J.-J. Quisquater, "Hessian elliptic curves and side-channel attacks," in *CHES 2001.* Springer-Verlag LNCS 2162, 2001, pp. 402–410.

[4] H. R. Frium, "The group law on elliptic curves on Hesse form," 09 2001.

[5] N. P. Smart, "The Hessian form of an elliptic curve," in *CHES 2001*, Koc, Naccache, and Paar, Eds. Springer-Verlag LNCS 2162, May 2001, pp. 118–125.

[6] N. P. Smart and E. J. Westwood, "Point multiplication on ordinary elliptic curves over fields of characteristic three."

[7] Gupta, Blake-Wilson, Moeller, and Hawk, "ECC cipher suites for TLS," 2002.

[8] B. Schneier, *Applied Cryptography.* John Wiley & Sons, inc., 1996.

[9] B. Johnson, *Kryptografi.* Tapir, 2001.

[10] T. Dierks and C. Allen, "The TLS protocol," 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2246.txt

[11] E. Rescorla, "HTTP over TLS," 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2818.txt

[12] J. B. Fraleigh, *A First Course in Abstract Algebra.* Addison Wesley, 1999.

[13] Blake, Seroussi, and Smart, *Elliptic Curves in Cryptography.* Cambridge university press, 1999.

[14] M. Desboves, "Résolution, en nombres entiers et sous sa forme la plus générale, de l'eqation cubique, homogène, à trois inconnues," pp. 545–579, 1886.

[15] LiDIA-Group. (2001) LiDIA - A library for computational number theory. [Online]. Available: http://www.informatik.tu-darmstadt.de/TI/LiDIA/

[16] H. Cohen, A. Miyaji, and T. Ono, "Efficient elliptic curve exponentiation using mixed coordinates," in *Advances in Cryptology-Proceedings of ASIACRYPT'98.* Springer-Verlag LNCS 1514, 1998, pp. 51–65.

[17] M. Joye and S.-M. Yen, "Optimal left-to-right binary signed-digit recoding," *IEEE Transactions on Computers 49(7)*, pp. 740–748, 2000.

[18] I. Connell, *Elliptic Curve Handbook*, 1999.

[19] The Gnu MP home page. [Online]. Available: http://www.swox.com/gmp/

[20] GCC home page. [Online]. Available: http://gcc.gnu.org/

[21] T. Izu and T. Takagi, "A fast parallel elliptic curve multiplication resistant against side channel attacks," in *PKC 2002*. Springer-Verlag LNCS 2274, 2002, pp. 280–296.

[22] A. Miyaji, T. Ono, and H. Cohen, "Efficient elliptic curve exponentiaion," in *Advances in Cryptology-Proceedings of ICICS'97*. Springer-Verlag LNCS 1334, 1997, pp. 282–290.

[23] R. P. Grimaldi, *Discrete and Combinatorial Mathematics - An Applied Introduction*. Addison Wesley, 1998.

Tillegg A

Matematikk

A.1 Innledning

Dette kapittelet inneholder en kort gjennomgang av noen matematiske emner som er viktige for forståelsen av elliptiske kurver. Vi forutsetter kjennskap til grunnleggende diskret matematikk.

A.2 Gruppeteori

Vi starter med definisjonen på en gruppe, deretter vil vi forklare litt mer om hva som menes med grupper, og gi noen eksempler på grupper.

Definisjon 4. *Dersom G er en mengde og $\circ$ er en binær operasjon på G, så er $(G, \circ)$ en gruppe, dersom følgende er tilfredsstilt.*

1. $\forall\ a, b \in G,\ a \circ b \in G$. *($G$ er lukket under $\circ$)*
2. $\forall\ a, b, c \in G,\ a \circ (b \circ c) = (a \circ b) \circ c$. *(Assosiativitetsegenskap)*
3. $\exists\ e \in G$ *slik at* $a \circ e = e \circ a = a, \forall a \in G$. *(Identitetselement)*
4. $\forall\ a \in G,\ \exists b \in G$ *slik at* $a \circ b = b \circ a = e$. *(Inverselement)*

En gruppe består av en mengde elementer og en lukket binær operasjon, som også kalles gruppeoperasjonen. Dersom man kan vise at en mengde elementer og en binær operasjon danner en gruppe, det vil si at alle reglene i definisjon 4 er tilfredsstilt, så vil alle regler som gjelder grupper også gjelde for denne mengden under gruppeoperasjonen. På denne måten er det utviklet mye matematisk teori som ikke er bundet opp til en spesiell anvendelse, men som er mer generell og gjelder for grupper, dette er grunnen til at grupper hører til under abstrakt algebra, se [23] side 701.

Eksempel 13: Et enkelt eksempel på en gruppe er de hele tallene under addisjon $G = (\mathbb{Z}, +)$. Vi vil kort gå gjennom regelene i definisjonen av grupper.

1. Gruppen G er *lukket* fordi summen av to heltall er et nytt heltall.
2. Gruppen G arver også asossiativitetsegenskaper fra + operatoren.

3. Her er 0 *identitetselementet* fordi $0 + a = a + 0 = a$.

4. For alle $a \in \mathbb{Z}$ så finnes det et *inverselement* $-a \in \mathbb{Z}$.

Eksempel 14: Et enkelt eksempel på noe som ikke er en gruppe er de hele positive tallene under addisjon $G = (\mathbb{Z}^+, +)$. Dette ikke er en gruppe fordi det ikke finnes et identitetselement.

En annen viktig egenskap ved grupper, er om de er abelske eller ikke.

Definisjon 5. *Dersom $a \circ b = b \circ a$ for alle $a, b \in G$, så sies gruppen G å være kommutativ eller abelsk.*

Gruppen i eksempel 13 er *abelsk* fordi addisjon av heltall er *kommutativ*. Eksempler på grupper som ikke er abelske er grupper av matriser under multiplikasjon, se [23] side 702.

Definisjon 6. *Med **ordenen** til en en gruppe G mener vi antall elementer i gruppen G, og vi skriver $|G|$. Dersom en gruppe har uendelig mange elementer sier vi at gruppen har uendelig orden.*

Det er vanlig å sløyfe gruppeoperasjonstegnet, slik at $a \circ b$ blir ab.
Her er noen viktige egenskaper som gjelder alle grupper:

Teorem 8. *For en gruppe G gjelder følgende:*

1. *identitetselementet til G er entydig.*

2. *inverselementet for hvert element i G er entydig.*

3. *hvis $a, b, c \in G$ og $ab = ac$, så er $b = c$(Venstre kanselleringslov).*

4. *hvis $a, b, c \in G$ og $ba = ca$, så er $b = c$(Høyre kanselleringslov).*

Bevis. Se teorem 16.1 i [23] □

Det viser seg ofte at noen av elementene i en gruppe danner en ny gruppe under den samme gruppeoperasjonen. Dette er er et meget viktig fenomen og kalles for en *undergruppe*. Alle grupper som ikke bare består av identitetselementet har to *trivielle undergrupper*, gruppen som dannes av identitetselementet alene, og gruppen selv.

Definisjon 7. *La G være en gruppe og $\emptyset \neq H \subseteq G$. Dersom H er en gruppe under den binære operasjonen til G, så sier vi at H er en undergruppe av G.*

Eksempel 15: Dersom vi ser på gruppen fra eksempel 13, så har den mange undergrupper, en av dem er $H = (2\mathbb{Z}, +)$, hvor $2\mathbb{Z} = \{2n | n \in Z\} = \{0, \pm 2, \pm 4, \pm 6 \cdots\}$.

Teorem 9. *Hvis H er en delmengde av G, så er H en undergruppe av G hvis og bare hvis*

1. *for alle $a, b \in H$, $ab \in H$*

2. *for alle $a \in H$, $a^{-1} \in H$*

Bevis. Se teorem 16.2 i [23] □

Teorem 9 gjør det enklere å vise at en gruppe er en undergruppe, det utnytter at assosiativiteten arves fra G. Dersom vi skal bruke Teorem 9 til å vise at $H = 2\mathbb{Z}$ fra Eksempel 15 er en undergruppe av $G = \mathbb{Z}$, så må vi vise at punkt 1 og 2 i Teorem 9 er oppfylt. Vi starter med å vise 1, det vil si at gruppen H er lukket: Dersom $a, b \in H$, så kan vi skrive $a = 2n, b = 2m,\ m, n \in G$. Vi får nå at $c = ab = 2n + 2m = 2(n + m),\ (n + m) \in G \Rightarrow 2(n + m) = c \in H$. Videre må vi vise 2, at alle elementer i H har et inverselement i H: $a^{-1} = -a = -2n = 2(-n),\ (-n) \in \mathbb{Z} \Rightarrow 2(-n) \in H$. Dersom en undergruppe er en *endelig gruppe*, det vil si at den inneholder et endelig antall elementer, så er det enda enklere å vise at den er en undergruppe.

Teorem 10. *Hvis G er en gruppe og $\emptyset \neq H \subseteq G$, og H er endelig, så er H en undergruppe av G hvis og bare hvis H er lukket under binæroperasjonen til G.*

Bevis. Se teorem 16.3 i [23] □

A.2.1 Addisjon modulo n

I dette avsnittet skal vi se på grupper av typen $(\mathbb{Z}_n, +)$ som er meget viktige innen gruppeteori. Først trenger vi en definisjon av *kongruens*.

Definisjon 8. *La $n \in \mathbb{Z}^+,\ n > 1$. For $a, b \in \mathbb{Z}$, så sier vi at a er kongruent til b modulo n, og vi skriver: $a \equiv b \pmod{n}$, hvis $n|(a - b)$, eller $a = b + kn$, hvor $k \in \mathbb{Z}$.*

Dersom vi tar utgangspunkt i de hele tallene $\mathbb{Z}$ så danner de en gruppe under addisjon, og hvor 0 er identitetselement. Den inverse til et tall $a \in \mathbb{Z}$ er $-a \in \mathbb{Z}$. Vi vil nå se på mengden av alle multipler av n. La n være et positivt heltall $n \in \mathbb{Z}^+$:

$$n\mathbf{Z} = \{0, \pm n, \pm 2n, \ldots\}$$

$n\mathbf{Z}$ er en undergruppe av $\mathbb{Z}$, og er opphavet til en *ekvivalensrelasjon* i $\mathbb{Z}$. To tall er ekvivalente hvis differansen mellom dem er delelig med n, og vi skriver:

$$a \backsim b \pmod{n}$$

Mer generelt kan vi si at to elementer $a, b \in \mathbb{Z}$ er ekvivalente dersom $a^{-1}b \in n\mathbb{Z}$ siden a^{-1} tilsvarer $-a$, så kan vi også skrive $(b - a) \in n\mathbb{Z}$, som er det samme som at n deler differansen mellom a og b. Denne ekvivalensrelasjonen deler $\mathbb{Z}$ inn i n ekvivalensklasser:

$$\begin{aligned} n\mathbb{Z} &= \{0, \pm n, \pm 2n, \ldots\} \\ 1 + n\mathbb{Z} &= \{1, 1 \pm n, 1 \pm 2n, \ldots\} \\ &\vdots \\ n - 1 + n\mathbb{Z} &= \{n - 1, n - 1 \pm n, n - 1 \pm 2n, \ldots\} \end{aligned}$$

De er vanlig å skrive $[k]$ istedenfor $k + n\mathbb{Z}$, og mengden av de n ekvivalensklassene betegnes $\mathbb{Z}_n = \{[0], [1], \ldots, [n - 1]\}$. Det er også vanlig å droppe paranteser og skrive 0 istedenfor $[0]$. Siden vi her behandler en ekvivalensrelasjon, tilhører alle elementer i $\mathbb{Z}$ en og bare en ekvivalensklasse, se [23] kapittel 7.1 for definisjon av ekvivalensrelasjon.

Vi vil nå definere en lukket binær operasjon på $\mathbb{Z}_n$ så vi kan danne en gruppe. Vi definerer addisjon av ekvivalensklasser ved å addere representanter fra ekvivalensklassene, slik at svaret blir den ekvivalensklassen som resultat fra addisjonen tilhører. Vi skriver addisjonen av ekvivalensklassene $[a], [b] \in \mathbb{Z}_n$på følgende måte:

$$[a] + [b] = [a + b]$$

Problemet med denne addisjonen er at to ulike representanter kan identifisere samme ekvivalensklasse, slik at $[a] = [b]$ selv om $a \neq b$. For at addisjonen skal være veldefinert, så må addisjonen gi samme resultat uavhengig av hvilken representant fra ekvivalensklassene som velges. Vi skal nå vise at addisjon av ekvivalensklasser er uavhengig av valg av representant: Først observerer vi at $[a] = [c] \Rightarrow a = c + sn$, hvor $s \in \mathbb{Z}$, og $[b] = [d] \Rightarrow b = d + tn$, hvor $t \in \mathbb{Z}$. Dermed følger det at

$$a + b = (c + sn) + (d + tn) = (c + d) + (s + t)n$$

dermed får vi at $(a + b) \equiv (c + d) \pmod{n}$ og $[a + b] = [c + d]$.

Teorem 11. *Dersom $n \in \mathbb{Z}^+$, så er $\mathbb{Z}_n$ en abelsk gruppe under den lukkede binæroperasjonen som er definert over, hvor $[0]$ er identitetselement.*

A.2.2 Sykliske grupper

Sykliske grupper er grupper som inneholder minst ett element som genererer hele gruppen dersom man utfører gruppeoperasjonen på dette elementet gjentatte ganger. Elementet som brukes til å generere gruppen kalles en generator.

Definisjon 9. *En gruppe G sies å være syklisk dersom det finnes et element $g \in G$, slik at for hver $a \in G$, $a = g^n$, hvor $n \in \mathbb{Z}$.*

I forbindelse med behandling av sykliske grupper vil vi nevne en spesiell type sykliske undergrupper. Gitt en gruppe G med et element $a \in G$, så sier vi at $\langle a \rangle = \{a^k | k \in \mathbb{Z}\} = \{a, a^2, a^3, \ldots\}$ er undergruppen generert av a, se [23] side 710.

Definisjon 10. *Hvis G er en gruppe og $a \in G$, så er ordenen til a (skrives $\mathcal{O}(a)$) $|\langle a \rangle|$.*

Videre kan det vises, se [23] side 712 for bevis, at alle undergrupper av sykliske grupper selv også er sykliske.

Eksempel 16: Vi vil nå se på gruppen $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$. Fra Tabell A.1 ser vi

Tabell A.1: Addisjonstabell for gruppen $(\mathbb{Z}_5, +)$.

$+_5$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

at gruppe $\mathbb{Z}_5$ er abelsk, fordi tabellen er symmetrisk om diagonalen som starter

i øverste venstre hjørne. Videre er vi interessert i å se om gruppen er syklisk, og vi ser på undergruppene $\langle a \rangle, a \in \mathbb{Z}_5$.

$$\langle 0 \rangle = \{0\}$$
$$\langle 1 \rangle = \{1, 2, 3, 4, 0\}$$
$$\langle 2 \rangle = \{2, 4, 1, 3, 0\}$$
$$\langle 3 \rangle = \{3, 1, 4, 2, 0\}$$
$$\langle 4 \rangle = \{4, 3, 2, 1, 0\}$$

Vi ser at gruppen er syklisk fordi den blir generert av alle elementene unntatt identitetselementet 0.

Det neste teoremet om syklisk grupper er svært viktig, og sier oss at det bare finnes en syklisk gruppe $\mathbb{Z}_n$ av orden n, alle andre sykliske grupper har samme struktur (er isomorfe), og det finnes en bijektiv avbildning som avbilder hvert element i den ene gruppen på et og bare et element i den andre gruppen. Det finnes også bare en syklisk gruppe $\mathbb{Z}$ med uendelig orden, alle andre sykliske grupper med uendelig orden er isomorfe med denne gruppen.

Teorem 12. *La G være en syklisk gruppe med generator a. Hvis ordenen til G er uendelig, så er G isomorf med $(\mathbb{Z}, +)$. Hvis G har orden n, så er G isomorf til $(\mathbb{Z}_n, +_n)$.*

Bevis. Se teorem 16.7 i [23] □

Det neste teoremet kan brukes til å finne størrelsen til en undergruppe av en syklisk gruppe.

Teorem 13. *La G være en syklisk gruppe med n elementer som er generert av a. La $b \in G$ og $b = a^s$, g genererer da en undergruppe H av G som inneholder n/d elementer, hvor d er den største felles divisor til n og s. Videre er $\langle a^s \rangle = \langle a^t \rangle$ hvis og bare hvis $gcd(s, n) = gcd(t, n)$.*

Bevis. Se teorem 6.14 i [12] □

Eksempel 17: La oss se på gruppen $\mathbb{Z}_{24}$, vi bruker addidativ notasjon og ser på undergruppen $\langle 10 \rangle$ generert av $a = 10$. Vi vil nå bruke Teorem 13 til å beregne $|\langle 10 \rangle|$. Vi må beregne største felles divisor til $12 = 2^2 \cdot 3$ og $10 = 2 \cdot 5$, som er 2. Størrelsen på undergruppen $\langle 10 \rangle$ blir $24/2 = 12$. Dersom vi skriver ut alle elementene i $\langle 10 \rangle$, så ser vi at dette stemmer.

$$\langle 10 \rangle = \{10, 20, 6, 16, 2, 12, 22, 8, 18, 4, 14, 0\}$$

A.2.3 Lagranges teorem

Dersom man ser på ordenen til en gruppe og ordenen til en undergruppe, så viser det seg at ordenen til undergruppen alltid deler ordenen til gruppen. Dette er sant for alle grupper og undergrupper, og er bevist av den franske matematikeren Lagrange, og kalles Lagranges teorem.

Et sentralt begrepet i beviset for Lagranges teorem er *restklasser*, som defineres slik:

Definisjon 11. *Hvis H er en undergruppe av G, for hver element $a \in G$, så kaller vi mengde $aH = \{ah | h \in H\}$ for en venstre restklasse til H i G. Mengden $Ha = \{ha | h \in H\}$ kalles for en høyre restklasse til H i G.*

For abelske grupper er høyre og venstre restklasser like, på grunn av kommutativitet.

Teorem 14. *(Lagranges teorem) Hvis G er en endelig gruppe med orden n, hvor H er en undergruppe av G med orden m, så vil $m|n$.*

Bevis. Se teorem 16.9 i [23] □

Som en interessant konsekvens av Lagranges teorem følger at hvis en gruppe G har primtallsorden, så er gruppen syklisk. Et primtall har ingen faktorer, dermed kan gruppen ikke ha andre undergrupper enn de trivielle. Dersom vi ser på gruppen $H = \langle a \rangle$ generert av et element $a \in G$, så må a enten generere hele gruppen G eller gruppen $\{e\}$ som består av bare identitetselementet. Siden $a \in \langle a \rangle$, så vil alle andre elementer enn identitetselementet generere hele gruppen, det vil si at alle elementer bortsett fra identitetselementet er generatorer for G, se [12] side 120 og [23] side 713.

Eksempel 18: La oss se på gruppen Z_6, vi bruker addidativ notasjon og ser på undergruppen $H = \{0, 3\}$. Vi vil se hvordan gruppen Z_6 kan deles inn i restklasser av undergruppen H.

$$
\begin{aligned}
0 + H &= \{0 + 0, 0 + 3\} = \{0, 3\} \\
1 + H &= \{1 + 0, 1 + 3\} = \{1, 4\} \\
2 + H &= \{2 + 0, 2 + 3\} = \{2, 5\} \\
3 + H &= \{3 + 0, 3 + 3\} = \{3, 0\} \\
4 + H &= \{4 + 0, 4 + 3\} = \{4, 1\} \\
5 + H &= \{5 + 0, 5 + 3\} = \{5, 2\}
\end{aligned}
$$

Vi ser at det dannes tre restklasser $\{0, 3\}$, $\{1, 4\}$, $\{2, 5\}$ med to elementer, det er ingen felles elementer i restklassene. Vi ser også at alle elementene i en restklasse er opphav til restklassen de selv er med i. Dette stemmer med Laranges teorem, som sier at ordenen til undergruppen som er 3 må dele ordene til gruppen selv, som er 6. Vi ser at dette stemmer, siden $3|6$.

A.2.4 Fermats teorem

I dette avsnittet skal vi se på grupper av typen $(\mathbb{Z}_p^*, \cdot)$ hvor p er et primtall, med $\mathbb{Z}_p^*$ mener vi de positive tallene $1, 2, \cdots, p - 1$.

Teorem 15. *Dersom $a \in \mathbb{Z}$ og p er et primtall som ikke deler a, så $p|a^{p-1} - 1$, dermed blir $a^{p-1} \equiv 1 \pmod{p}$, for $a \not\equiv 0 \pmod{p}$.*

Bevis. La G være en gruppe med orden n, og $a \in G$. Gruppen $\langle a \rangle$ må ha en orden som deler n, dette kan vi vise ved å bruke Lagranges teorem, som sier at ordenen til en undergruppe må dele ordenen til gruppen selv, siden $\langle a \rangle$ er en undergruppe av G, har vi at $m = |\langle a \rangle| = kn$, hvor $k \in \mathbb{Z}^+$. Av dette følger det at $a^m = 1$, og vi får at $a^n = a^{mn} = (a^m)^n = 1^n = 1$. Vi ser nå igjen på gruppen $\mathbb{Z}_p^*$ som

har orden $p-1$, la $a \in \mathbb{Z}_p^*$, vi får dermed at $a^{(p-1)} = 1$, eller $a^{(p-1)} \equiv 1 \pmod{p}$, og videre at $a^p \equiv a^{(p-1)}a \equiv 1a \equiv a \pmod{p}$. For alle $a \in Z$, hvis $p \mid a$ så $a = kp$ og $a \equiv kp \equiv 0 \pmod{p}$, hvis $p \mid a$ så har vi at $a \equiv b \pmod{p}$, hvor $1 \leq b \leq (p-1)$, og $a^p \equiv b^p \equiv b \equiv a \pmod{p}$. □

A.2.5 Strukturteorem for abelske grupper

I dette avsnittet vil vi vise hvordan vi kan bygge nye grupper av de sykliske gruppene $\mathbb{Z}_n$. Teorien i dette avsnittet gjelder bare abelske grupper. Før vi starter må vi definere *kartesisk produkt* av mengder:

Definisjon 12. ***Kartesisk produkt av mengdene*** $S_1, S_2, \cdots, S_n$ *er mengden av alle ordnede n-tupler* $(a_1, a_2, \cdots, a_n)$*, hvor* $a_i \in S_i$ *for* $i = 1, 2, \cdots, n$*. Det kartesiske produktet skrives enten*

$$S_1 \times S_2 \times \cdots \times S_n$$

eller

$$\prod_{i=1}^{n} S_i.$$

La $G_1, G_2, \cdots, G_n$ være grupper, fra disse gruppene danner vi en ny gruppe $\prod_{i=1}^{n} G_i$, hvor gruppeoperasjonen er komponentvis multiplikasjon. Her gjør vi en forekling i notasjonen og bruker samme notasjon på gruppen som vi bruker på mengden med elementer i gruppen.

Teorem 16. *La* $G_1, G_2, \cdots, G_n$ *være grupper. For* $(a_1, a_2, \cdots, a_n), (b_1, b_2, \cdots, b_n) \in \prod_{i=1}^{n} G_i$*, definerer vi produktet av to elementer* $(a_1, a_2, \cdots, a_n)(b_1, b_2, \cdots, b_n)$ *til å være* $(a_1b_1, a_2b_2, \cdots, a_nb_n)$*. Da er* $\prod_{i=1}^{n} G_i$ *en gruppe, som vi betegner det* direkte produkt til gruppene G_i *under denne binære operasjonen.*

Bevis. Se teorem 11.2 i [12] □

Eksempel 19: Dersom vi definerer en direkte produktgruppe mellom $\mathbb{Z}_2$ og $\mathbb{Z}_3$, får vi gruppen $\mathbb{Z}_2 \times \mathbb{Z}_3$. Gruppen har følgende elementer:
$\mathbb{Z}_2 \times \mathbb{Z}_3 = \{(0,0), (0,1), (0,2), (1,0), (1,1), (1,2)\}$. Vi vil nå vise at gruppen er syklisk,

Tabell A.2: Addisjonstabell for gruppen $(\mathbb{Z}_2 \times \mathbb{Z}_3, +)$

+	(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)
(0,0)	(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)
(0,1)	(0,1)	(0,2)	(0,0)	(1,1)	(1,2)	(1,0)
(0,2)	(0,2)	(0,0)	(0,1)	(1,2)	(1,0)	(1,1)
(1,0)	(1,0)	(1,1)	(1,2)	(0,0)	(0,1)	(0,2)
(1,1)	(1,1)	(1,2)	(1,0)	(0,1)	(0,2)	(0,0)
(1,2)	(1,2)	(1,0)	(1,1)	(0,2)	(0,0)	(0,1)

ved å vise at elementet $(1,1)$ genererer hele gruppen, vi skriver gruppeoperasjonen addidativt, fordi gruppeoperasjonen her er addisjon.

$$\langle(1,1)\rangle = \{1(1,1), 2(1,1), 3(1,1), 4(1,1), 5(1,1), 6(1,1)\}$$
$$\langle(1,1)\rangle = \{(1,1), (0,2), (1,0), (0,1), (1,2), (0,0)\}$$

Vi vet nå fra Teorem 12 at denne gruppen må være isomorf med gruppen $\mathbb{Z}_6$, siden det bare finnes en gruppestruktur med seks elementer som er syklisk.

Teorem 17. *Gruppen $\mathbb{Z}_m \times \mathbb{Z}_n$ er syklisk og isomorf med $\mathbb{Z}_{mn}$ hvis og bare hvis m og n er relative primtall.*

Bevis. Se teorem 11.5 i [12] □

Vi kunne brukt Teorem 17 direkte til å vise at gruppen i Eksempel 16 er syklisk og isomorf med den sykliske gruppen $\mathbb{Z}_6$. Det neste korollaret er bare en generalisering av teorem 17, som gjelder kartesisk produkt mellom flere grupper.

Korollar 1. *Gruppen $\prod_{i=1}^{n} \mathbb{Z}_{m_i}$ er syklisk og isomorf til gruppen $\mathbb{Z}_{m_1 m_2 \cdots m_n}$ hvis og bare hvis tallene m_i for $i = 1, 2, \cdots, n$ er relative primtall.*

Bevis. Se korollar 11.6 i [12] □

Vi vil nå presentere et teorem som gjør oss i stand til å finne ordenen til et element i en direkte produktgruppe, dette teoremet tilsvarer Teorem 13, som gjelder sykliske grupper av typen $\mathbb{Z}_n$.

Teorem 18. *La $(a_1, a_2, \cdots, a_n) \in \prod_{i=1}^{n} G_i$. Hvis a_i er av endelig orden r_i i G_i, så er ordenen til $(a_1, a_2, \cdots, a_n)$ i $\prod_{i=1}^{n} G_i$ lik det minste felles multippel av alle r_i.*

Bevis. Se teorem 11.9 i [12] □

Det kan vises at alle abelske grupper kan skrives som et direkte produkt av sykliske grupper med orden p^{n_i}, hvor p_i er primtall, og $n_i \in \mathbb{Z}^+$. Dette gir oss full oversikt over hvilke abelske grupper som finnes for en gitt orden. Dette er uttrykt i følgende teorem:

Teorem 19. *Enhver endelig generert abelsk gruppe G er isomorf til et direkte produkt av sykliske grupper på formen:*

$$\mathbb{Z}_{(p_1)^{r_1}} \times \mathbb{Z}_{(p_2)^{r_2}} \times \cdots \times \mathbb{Z}_{(p_n)^{r_n}} \times \mathbb{Z} \times \mathbb{Z} \times \cdots \times \mathbb{Z}$$

Hvor p_i er primtall, men ikke nødvendigvis forskjellige, og r_i er positive hele tall. Det direkte produktet er unikt, bortsett fra mulige omstokkinger av faktorene.

Bevis. Se teorem 11.12 i [12] □

Dette teoremet sier at enhver abelsk gruppe av en gitt orden er isomorf med en direkte produktgruppe av sykliske grupper. Dette gir oss full oversikt over hvilke abelske grupper som finnes for en gitt orden.

Eksempel 20: Som eksempel på bruk av strukturteoremet, vil vi se på hvilke abelske grupper som finnes med orden 36. Vi starter med å faktorisere 36 i primtall $36 = 2^2 3^2$, så ser vi på hvordan vi kan skrive 36 som et produkt av $p_i^{n_i}$ hvor p_i er et primtall, og n_i er et helt positivt tall.

$$36 = 2 \cdot 2 \cdot 3 \cdot 3 \tag{A.1}$$
$$36 = 2 \cdot 2 \cdot 9 \tag{A.2}$$
$$36 = 4 \cdot 3 \cdot 3 \tag{A.3}$$
$$36 = 4 \cdot 9 \tag{A.4}$$

Vi ser at det er fire måter å faktorisere 36 på som tilfredsstiller kravet i strukturteoremet, det vil si at det finnes fire gruppestrukturer for abelske grupper

av orden 36. Alle andre abelsk grupper av orden 36 er isomorf med en av disse direkte produkt gruppene.

$$\mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_3 \times \mathbb{Z}_3 \tag{A.5}$$
$$\mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_9 \tag{A.6}$$
$$\mathbb{Z}_4 \times \mathbb{Z}_3 \times \mathbb{Z}_3 \tag{A.7}$$
$$\mathbb{Z}_4 \times \mathbb{Z}_9 \tag{A.8}$$

En av disse gruppene må være isomorf med gruppen $\mathbb{Z}_{36}$, ved å bruke Korollar 1, så ser vi at det må være gruppen $\mathbb{Z}_4 \times \mathbb{Z}_9$ siden det er den eneste gruppen hvor ordenen til de sykliske gruppene er relative primtall.

A.2.6 Homomorfi og isomorfi

I dette avsnittet skal vi se på sammenhenger mellom grupper. Vi skal se på avbildninger mellom grupper som bevarer strukturen. Dersom man vet alt om strukturen til en av gruppene, og man kan finne en slik avbildning på en annen gruppe, så vet man straks ganske mye om denne gruppen. Dersom man kan finne en isomorfi mellom to grupper, så vet man at disse gruppene har helt lik struktur, og egentlig er samme gruppe. En isomorfi krever at det er et en-til-en forhold mellom elementene i de to gruppene, det vil si at avbildningen er en-til-en og på.

Definisjon 13. *(**Homomorfi**) En avbildning ϕ av en gruppe G på en gruppe G' er en homomorfi hvis:*

$$\phi(ab) = \phi(a)\phi(b)$$

for alle $a, b \in G$.

Det finnes alltid en homomorfi $\phi : G \to G'$ mellom to grupper, den *trivielle homomorfien* definert ved $\phi(g) = e'$ for alle $g \in G$, og som avbilder alle elementene i den ene gruppen over på identitetselementet i den andre gruppen.

Teorem 20. *La ϕ være en homomorfi som avbilder gruppen G på gruppen G'.*

1. *Hvis e er identitet i G, så er $e' = \phi(e)$ identitet i G'.*
2. *Hvis $a \in G$, så er $\phi(a^{-1}) = \phi^{-1}(a)$.*
3. *Hvis H er en undergruppe av G, så er $\phi[H]$ en undergruppe av G'.*
4. *Hvis K' er en undergruppe av G' så er $\phi^{-1}[K']$ en undergruppe av G.*

Bevis. Se teorem 13.12 i [12] □

Punkt 3 og 4 i Teorem 20 er spesielt nyttige, og sier at dersom vi har en undergruppe så vil også avbildningen på den andre gruppen være en undergruppe, dersom funksjonen er en homomorfi.

Et viktig begrep er *kjernen* til en homomorfi, det vil si alle elementene som avbildes på identitetselementet.

Definisjon 14. ***(Kjernen)** La $\phi : G \to G'$ være en homomorfi. Undergruppen $\phi^{-1}(\{e'\}) = \{x \in G \mid \phi(x) = e'\}$ er kjernen til ϕ, og skrives $Ker(\phi)$.*

Fra punkt 3 og 4 i Teorem 20 ser vi at kjernen til en homomorfi er en undergruppe.

Definisjon 15. *En undergruppe H av G er normal hvis høyre-restklassene og venstre-restklassene er sammenfallende, det vil si at $gH = Hg$ for alle $g \in G$.*

Alle undergrupper av abelske grupper er normale, men det motsatte er ikke alltid tilfelle.

Definisjon 16. ***(Isomorfi)*** *Hvis ϕ er en bijektiv avbildning (en-til-en og på) av en gruppe G på en annen gruppe G', og*

$$\phi(ab) = \phi(a)\phi(b)$$

*for alle $a, b \in G$, så er ϕ en **isomorfi**, og G og G' er **isomorfe** og vi skriver $G \approx G'$.*

Se [12] side 161.

A.2.7 Kvotienter av abelske grupper

I dette avsnittet vil vi se på gruppen som dannes av restklasser, hvor gruppeoperasjonen er venstre multiplikasjon av representanter. Resultatet fra en slik multiplikasjon er den restklassen som inneholder resultatet fra multiplikasjonen av representantene.

Teorem 21. *La H være en undergruppe av G. Venstre restklassemultiplikasjon er da veldefinert ved ligningen:*

$$(a)H(b)H = (ab)H$$

hvis og bare hvis H er en normal undergruppe av G.

Bevis. Se teorem 14.4 i [12] □

Korollar 2. *La H være en undergruppe av G hvor venstre og høyre restklasser er sammenfallende. Da danner restklassene til H en gruppe G/H, under den binære operasjonen $(aH)(bH) = (ab)H$.*

Bevis. Se korollar 14.5 i [12] □

Undergrupper av abelske grupper er alltid normale, og vil alltid kunne danne en kvotientgruppe.

Definisjon 17. ***(Kvotientgruppe)*** *Gruppen G/H i korollar 2 er en kvotientgruppe (eller en faktorgruppe) av G modulo H.*

Eksempel 21: Vi vil nå se på gruppen $\mathbb{Z}$, og på den normale undergruppen $6\mathbb{Z}$, denne undergruppen er normal fordi $\mathbb{Z}$ er abelsk. Vi får følgende restklasser:

$$\begin{aligned}
[0] &= 0 + 6\mathbb{Z} &&= \{\cdots, -12, -6, 0, 6, 12, \cdots\} \\
[1] &= 1 + 6\mathbb{Z} &&= \{\cdots, -11, -5, 1, 7, 13, \cdots\} \\
[2] &= 2 + 6\mathbb{Z} &&= \{\cdots, -10, -4, 2, 8, 14, \cdots\} \\
[3] &= 3 + 6\mathbb{Z} &&= \{\cdots, -9, -3, 3, 9, 15, \cdots\} \\
[4] &= 4 + 6\mathbb{Z} &&= \{\cdots, -8, -2, 4, 10, 16, \cdots\} \\
[5] &= 5 + 6\mathbb{Z} &&= \{\cdots, -7, -1, 5, 11, 17, \cdots\}
\end{aligned}$$

Disse danner kvotientgruppen $\mathbb{Z}/6\mathbb{Z} = \{[0], [1], [2], [3], [4], [5]\}$, som er isomorf til $\mathbb{Z}_6$. Videre vil vi vise hvordan vi adderer to elementer $[3]$ og $[4]$, $3 + 4 = 7$. Siden 7 tilhører restklassen $[1]$, så blir svaret $[1]$, og vi får at $[3] + [4] = [1]$. Vi skal få samme svar om vi velger to andre elementer i de samme restklassene og legger dem sammen, vi velger to nye elementer $[-9]$ og $[16]$ fra de samme restklassene og ser at $-9 + 16 = 7$ som er i samme restklasse som 1.

A.3 Ringer, modulær aritmetikk

I kapittelet om grupper så vi på en mengde hvor det var definert en binær operasjon. I dette kapittelet skal vi se på en ny struktur som består av en mengde hvor det er definert to binære operasjoner, henholdsvis addisjon og multiplikasjon, denne strukturen kalles en **ring**. Helt fra barneskolen har vi lært om multiplikasjon og addisjon, og har derfor en intuitiv forståelse for hva en ring er siden den definerer addisjon og multiplikasjon. Selv om en ringstruktur er mer intuitiv enn gruppe, så er den matematisk sett mer komplisert enn gruppe fordi den består av to binære operasjoner, i motsetning til en gruppe som bare har definert en binær operasjon.

Definisjon 18. *En* ring $(A, +, \cdot)$ *er en mengde med to binære operasjoner (multiplikasjon og addisjon), slik at for alle* $a, b, c \in A$. *Vi vil bruke* z *som symbol for addidativ identitetselement.*

1. $(A, +)$ *er en abelsk gruppe under addisjon, hvor* z *er identitetselement.*

2. *Multiplikasjon er assosiativ og distributiv over addisjon, det vil si at*

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

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

I denne definisjonen bør vil legge merke til noen detaljer, addisjonsoperasjonen er kommutativ, det vil si at $a + b = b + a$ for alle $a, b \in A$, fordi gruppen $(A, +)$ er abelsk. Det er ikke et krav at multiplikasjon skal være kommutativ, og det er ikke nødvendigvis sant at $ab = ba, a, b \in A$. Videre finnes det for alle elementer $a \in A$ et addidativt inverselement $-a \in A$ fordi $(A, +)$ er en gruppe. Det eksisterer alltid et addidativt identitetselement som betegnes z, fordi $(A, +)$ er en gruppe, men det samme gjelder ikke for multiplikasjon.

Vi vil som oftest droppe multiplikasjonstegnet og skrive ab i stedet for $a \cdot b$. Vi vil også skrive A når vi mener $(A, +, \cdot)$. Vi vil også omtale addidativt identitetselement z som null.

Definisjon 19. *Hvis* a *og* b *er to elementer i* R *slik at* $ab = 0$, *så er* a *og* b ***nulldivisorer**.*

I følgende avsnitt ser vi på ringer hvor det stilles tilleggskrav i forhold til kravene i definisjon av ringer.

Definisjon 20. *La* $(R, +, \cdot)$ *være en ring*

1. *Hvis* $ab = ba$ *for alle* $a, b \in R$, *så sier vi at* R *er en* ***kommutativ ring**.*

2. *Ringen R sies å ikke ha* ***ekte divisorer av null*** *hvis for alle $a, b \in R$, $ab = 0 \Rightarrow a = 0$ eller $b = 0$.*

3. *Hvis et element $u \in R$ oppfører seg slik at $u \neq 0$ og $au = ua = a$ for alle $a \in R$, så kaller vi u* ***multiplikativ enhet*** *for R. og R kalles en* ***ring med multiplikativ enhet****.*

En kommutativ ring er en ring hvor også multiplikasjon er kommutativ, det vil si at både multiplikasjon og addisjon er kommutativ. En ring kan ha divisorer av null, det vil si at to elementer multiplisert sammen gir det addidative identitetselementet, dette avviker fra våre tidligere erfaringer med multiplikasjon av heltall. En ring med multiplikativt identitetselementet er en ring som inneholder et element som oppfører seg som tallet 1 gjør det i vanlig heltallsmultiplikasjon.

Teorem 22. *Dersom $n \in \mathbb{Z}^+$, $n > 0$, så er Z_n en kommutativ ring med multiplikativ identitet 1 og addidativ identitet 0, under addisjon modulo n.*

Bevis. Se teorem 14.12 i [12] □

A.3.1 Integritetsområder og kropper

En av de viktigste egenskapene ved vårt vanlige tallsystem er at dersom produktet av to tall er null, så er minst en av faktorene null. Vi bruker denne regelen til å løse ligninger av typen $x^2 - 5x + 6 = 0$, dersom vi faktoriserer finner vi svaret må være 2 eller 3:

$$x^2 - 5x + 6 = (x - 2)(x - 3) = 0$$

Problemet er at denne regelen ikke uten videre kan brukes til å løse ligninger i ringer, fordi det i utgangspunktet ikke er noen regel som sier at hvis produktet av to elementer er null, så er en av de to elementene null. Vi vil vise et eksempel for å illustrere problemet med ekte divisorer av null.

Eksempel 22: (Fra [12]) Vi vil prøve å løse ligningen $x^2 - 5x + 6 = 0$ i $\mathbb{Z}_{12}$. Vi kan faktorisere $x^2 - 5x + 6 = 0$ slik at vi får $(x - 2)(x - 3) = 0$, dette gir oss løsningene 2 og 3, problemet er at dette ikke er de eneste løsningene. Grunnen er at $\mathbb{Z}_{12}$ har ekte divisorer av null, og dermed er det ikke bare $0a = 0$ og $a0 = 0$ for $a \in \mathbb{Z}_{12}$, men også

$$\begin{aligned}(2)(6) = (3)(4) = (4)(3) = (3)(8) = (8)(3)\\ = (4)(6) = (6)(4) = (4)(9) = (9)(4) = (6)(6)\\ = (6)(8) = (8)(6) = (6)(10) = (10)(6) = (8)(9) = (9)(8) = 0\end{aligned}$$

Vi får dermed at 6 og 11 også er løsninger på ligningen siden :

$$(6 - 2)(6 - 3) = (4)(3) = 0, \quad (11 - 2)(11 - 3) = (9)(8) = 0$$

Dersom vi ser på elementene i $\mathbb{Z}_{12}$ som er **divisorer av 0** $\{2, 3, 4, 6, 8, 9, 10\}$, så er ingen av dem relative primtall til 12, dette bekreftes i Teorem 23.

Teorem 23. *I ringen $\mathbb{Z}_n$, er det de elementene som er ulik null, og som ikke er relative primtall til n, som er divisorer av null.*

Bevis. Se teorem 19.3 i [23] □

Ringen $\mathbb{Z}_p$ har ingen divisorer av null dersom p er et primtall, fordi alle elementene i $\mathbb{Z}_p$ er relative primtall med p.

Korollar 3. *Hvis p er et primtall, så har $\mathbb{Z}_p$ ingen ekte divisorer av null.*

Bevis. Se korollar 19.4 i [23] □

Alle elementer som har en multiplikativ invers kalles en multiplikativ enhet.

Definisjon 21. *La R være en ring med multiplikativ enhet 1. Hvis $a \in R$ og det eksisterer en $b \in R$ slik at $ab = ba = 1$, så sier vi at b er den multiplikative enhet til a, og a kalles en enhet i R.*

Ringer som ikke har ekte divisorer av null, er svært viktige og har fått navnet integritetsområde. Dersom en ring er et integritetsområde, gjelder fortsatt regelen fra det vanlige tallsystemet, som sier at dersom produktet av to elementer er null, så må minst ett av elementene være lik null.

Definisjon 22. *La R være en kommutativ ring med multiplikativ enhet. R kalles et* integritetsområde*, hvis R ikke har noen ekte divisorer av null.*

Et enda strengere krav til en ring, er at alle elementer untatt null skal være en enhet, det vil si at de har en multiplikativ invers, en slik ring kalles en *kropp*. En kropp er en ring hvor vi kan utføre addisjon, multiplikasjon, subtraksjon og i tillegg **divisjon**.

Definisjon 23. *La R være en kommutativ ring med multiplikativ enhet. R kalles en **kropp** dersom alle elementer i R untatt null er en enhet.*

Neste teorem sier at alle ringer av typen $\mathbb{Z}_p$, hvor p er et primtall er en kropp, dersom p ikke er et primtall så er ikke p en kropp. Det vil si at alle kropper av denne typen vil ha primtallsorden, senere i dette kapittelet skal vi se på kropper som ikke behøver å ha primtallsorden.

Teorem 24. *$\mathbb{Z}_n$ er en kropp hvis og bare hvis n er et primtall.*

Bevis. Se korollar 19.12 i [12] □

A.3.2 Underringer og idealer

For grupper har vi definert undergrupper av en gruppe som en delmengde av en gruppe som selv er en gruppe. Vi har tilsvarende for ringer, en delmengde av en ring som selv er en ring, under samme binære operasjon, kalles en underring.

Definisjon 24. *La $(R, +, \cdot)$ være en ring, en delmengde $S \neq \emptyset$ av R kalles en underring av R dersom S under addisjon og multiplikasjon (som er definert for R) er begrenset til S.*

Under behandlingen av grupper så vi på restklassen som dannes av kvotienten G/H mellom to grupper G og H, restklassene danner en gruppe dersom H er en normal undergruppe av G. For ringer har vi et tilsvarende krav, dersom restklassene som dannes av kvotienten R/H mellom to ringer R og H skal danne en ring så må H være et ideal i R.

Definisjon 25. *En delmengde $I \neq \emptyset$ i R kalles et ideal av R, hvis $a - b \in I$ og $ar, ra \in I$ for alle $a, b \in I$, og alle $r \in R$.*

Definisjon 26. *La A og B være to ringer. En avbildning $f : A \to B$ kalles en* ***ringhomomorfi*** *hvis for alle $a, b \in A$.*

1. $f(x+y) = f(x) + f(y)$
2. $f(x \cdot y) = f(x) \cdot f(y)$

La R være en ring, det finnes da i noen tilfeller et tall n slik at $na = 0$ for alle $a \in R$. Det vil si at hvis vi legger et element sammen med seg selv n ganger så får vi alltid null. Det minste tallet som har denne egenskapen kaller vi karakteristikken til R. Eksempelvis vil karakteristikken til $\mathbb{Z}_n$ være n.

Definisjon 27. *Hvis det for en ring R finnes et positivt tall n slik at $n \cdot a = 0$ for alle $a \in R$, hvor $n \cdot a$ betyr $\underbrace{a + a + \cdots + a}_{n}$, så er det minste positive heltallet som oppfører seg slik* ***karakteristikken*** *til R. Dersom det ikke finnes et slik tall, sier vi at ringen R har* ***karakteristikk 0****.*

A.4 Endelige kropper

A.4.1 Polynomringer

I dette avsnittet vil vi se på hvordan vi kan konstruere ringer av polynomer. Med utgangspunkt i en ring R kan vi konstruere en ring $R[x]$, som er ringen av alle polynomer med koeffisienter i R under polynomaddisjon og polynommultiplikasjon.

Definisjon 28. *La R være en ring. Et uttrykk på formen $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, hvor $a_i \in R$ for alle $0 \leq i \leq n$, kalles et polynom i den ubestemte variabelen x med koeffisienter i R. Dersom a_n ikke er additiv identitet i R, så kalles a_n den ledende koeffisienten i $f(x)$, og vi sier at $f(x)$ har graden n. Graden til et polynom er den høyeste eksponenten av x i polynomet. Leddet $a_0 x^0$ kalles konstantleddet til $f(x)$. Vi bruker notasjonen $R[x]$ for å representere mengden av alle polynomer i x med koeffisienter i R.*

Eksempel 23: Polynomet $f(x) = x^2 - 2$ har graden 2, og koeffisientene er $a_0 = -2$, $a_1 = 0$, $a_2 = 1$. Polynomet $f(x)$ er et polynom med koeffisienter i $\mathbb{Z}$.

Et **konstant polynom** er et polynom hvor alle koeffisientene bortsett fra a_0 er lik 0.

Teorem 25. *Mengden $R[x]$ av alle polynomer av den ubestemte variabelen x med koeffisienter i ringen R, er en ring under polynomaddisjon og polynommultiplikasjon. Dersom R er kommutativ, så er også $R[x]$ kommutativ, og hvis R har multiplikativt identitetselement $1 \neq 0$, så er også 1 multiplikativt identitetselement i $R[x]$. $R[x]$ er et integritetsområde dersom R er et integritetsområde.*

Bevis. Se teorem 17.1 i [23] □

Definisjon 29. *La R være en ring med multiplikativt identitetselement u, og la $f(x) \in R[x]$, hvor graden til $f(x)$ er større en 1. Hvor $r \in R$ og $f(x) = z$, så sier vi at r er en rot i polynomet $f(x)$.*

Definisjon 30. *La F være en kropp. For $f(x), g(x) \in F[x]$, hvor $f(x)$ ikke er nullpolynomet, så sier vi at $f(x)$ er en divisor (eller faktor) i $g(x)$ hvis det eksisterer et polynom $h(x) \in F[x]$ slik at $f(x)h(x) = g(x)$. Vi sier at $f(x)$ deler $g(x)$, og at $g(x)$ er et multippel av $f(x)$.*

Eksempel 24: Dersom $f(x) = x^2 - 3x + 2 \in \mathbb{Z}_7[x]$, så er $(x-1), (x-2) \in Z_7[x]$ faktorer i $f(x)$.

Eksempel 25: Dersom $f(x) = x^2 - 2 \in \mathbb{Z}_7[x]$, så er ikke $(x - \sqrt{2}), (x + \sqrt{2})$ faktorer i $f(x)$, fordi $(x - \sqrt{2}), (x + \sqrt{2}) \notin \mathbb{Z}_7[x]$.

Fra tidligere matematikk-kurs har vi lært om polynomdivisjon, hvor polynomet har koeffisienter i $\mathbb{R}$. Vi starter med to polynomer $f(x), (g(x) \in \mathbb{R}[x]$ hvor graden til $g(x)$ er større eller lik graden til $f(x)$. Vi kan da utføre polynomdivisjonen hvor vi deler $g(x)$ på $f(x)$. Vi vil da få en kvotient $q(x) \in \mathbb{R}[x]$ og en rest $r(x) \in \mathbb{R}[x]$, hvor $r(x) = 0$ eller graden til $r(x)$ er lavere en graden til $f(x)$. Det følger da at $g(x) = q(x)f(x) + r(x)$. Det viser seg at denne teknikken også fungerer om vi bytte ut de reelle tallene med en kropp.

Teorem 26. ***(Divisjonsalgoritmen for** $F[x]$**)*** *La*

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$$

og

$$g(x) = b_m x^m + b_{m-1} x^{m-1} + \cdots + b_0$$

være to elementer i $R[x]$, hvor a_n, b_m er ulik null i R, og $m > 0$. Det finnes da unike polynomer $q(x)$ og $r(x)$ i $R[x]$ slik at $f(x) = g(x)q(x) + r(x)$, hvor $r(x) = 0$ eller graden til $r(x)$ er lavere en graden til $g(x)$.

Bevis. Se teorem 17.3 i [23] □

Teorem 27. ***(Faktorteorem)*** *Et element $a \in R$ er et nullpunkt i $f(x) \in F[x]$ hvis og bare hvis $x - a$ er en faktor i $f(x) \in F[x]$.*

Bevis. Se korollar 23.3 i [12] □

Korollar 4. *Et polynom $f(x) \in F[x]$ med grad n kan maksimalt ha n nullpunkter i kroppen F.*

Bevis. Se korollar 23.5 i [12] □

A.4.2 Irredusible polynomer

I tallteori er primtall et meget sentralt begrep, det vil si tall som ikke kan faktoriseres. Vi skal nå se på polynomer som har tilnærmet samme egenskaper blant polynomer som primtall har blant de hele tallene. Slike polynomer kalles for irredusible polynomer eller primpolynomer, og kan ikke faktoriseres over den kroppen de er definert over.

Definisjon 31. *Et polynom $f(x) \in F[x]$ som ikke er konstant, er* irredusibelt over F *eller* irredusibelt *i $F[x]$ hvis $f(x)$ ikke kan uttrykkes som et produkt $g(x)h(x)$ av to polynomer $g(x)$ og $h(x)$ i $F[x]$, begge med lavere grad enn $f(x)$. Hvis $f(x) \in F[x]$ ikke er konstant og ikke er irredusibelt over F, så er $f(x)$* redusibelt over F.

Det er veldig viktig å forstå at et polynom kan være irredusibelt over en kropp, og redusibelt over en annen kropp.

Teorem 28. *La $f(x) \in F[x]$, og la $f(x)$ være av graden 2 eller 3. Da er $f(x)$ irredusibelt over F hvis og bare hvis det ikke har nullpunkter i F.*

Bevis. Se teorem 23.10 i [12] □

Definisjon 32. *Et polynom $f(x) \in F[x]$ kalles monic hvis den ledende koeffisienten er 1, multiplikativt identitetselement i F.*

Vi kan definere største felles divisor eller gcd mellom to polynomer, på tilnærmet samme måte som det er definert for de hele tallene. Det finnes også en versjon av Euklids algoritme for polynomer, men den har vi ikke tatt med her.

Definisjon 33. *Dersom $f(x), g(x) \in F[x]$, så er $h(x) \in F[x]$ største felles divisor til $f(x)$ og $g(x)$ hvis:*

1. *$h(x)$ deler både $f(x)$ og $g(x)$.*
2. *$k(x) \in F[x]$ deler både $f(x)$ og $g(x)$, så deler $k(x)$ også $h(x)$.*

A.4.3 Kroppsutvidelser

Med en kroppsutvidelse mener vi at en kropp E er en utvidelse av en kropp F, dersom alle elementer i E er elementer i F. Eksempler på kroppsutvidelser er $\mathbb{C}$ som er en utvidelse av $\mathbb{R}$, som igjen er en utvidelse av $\mathbb{Q}$.

Definisjon 34. *En kropp E er en kroppsutvidelse av kroppen F dersom $F \subseteq E$.*

I kapittelet om grupper forklarte vi hvordan de hele tallene kan deles opp i ekvivalensklasser ved hjelp av en ekvivalensrelasjon, som sier at to tall er ekvivalente eller kongruente modulo n dersom differansen mellom de to tallene er et multippel av n. Mengden av av disse ekvivalensklassene som også er rest klasser, danner en ring under addisjon og multiplikasjon av representanter. Ringer av denne typen har vi omtalt som $\mathbb{Z}_n$. For polynomer kan man gjøre tilsvarende, og dele $F[x]$ opp i ekvivalensklasser, ved hjelp av en ekvivalensrelasjon som sier at to polynomer er ekvivalente eller kongruente modulo $s(x) \in F[x]$ dersom differansen mellom to elementer er et polynom $t(x) \in F[x]$ multiplisert med $s(x)$.

Teorem 29. *La $s(x) \in F[x], s(x) \neq 0$. Definer en relasjon $\mathscr{R}$ på $F[x]$ ved at $f(x)\mathscr{R}g(x)$ hvis $f(x) - g(x) = t(x)s(x)$ hvor $t(x) \in F[x]$, det vil si at $s(x)$ deler $f(x) - g(x)$. Da er $\mathscr{R}$ en ekvivalensrelasjon på $F[x]$.*

Bevis. Se teorem 17.10 i [23] □

Teorem 30. *La $s(x)$ være et polynom ulikt null i $F[x]$.*

1. *Ekvivalensklassene i $F[x]$ indusert av kongruens modulo $s(x)$ relasjonen, danner en kommutativ ring med multiplikativt identitetselement under de lukkede binær operasjonene*

$$[f(x)] + [g(x)] = [f(x) + g(x)], \quad [f(x)][g(x)] = [f(x)g(x)] = [r(x)]$$

hvor $r(x)$ er resten som oppstår når $f(x)g(x)$ deles med $s(x)$. Denne ringen skrives $F[x]/(s(x))$.

2. *Hvis* $s(x)$ *er ikke reduserbar i* $F[x]$*, så er* $F[x]/(s(x))$ *en kropp.*

3. *Hvis* $|F| = q$ *og graden til* $s(x)$ *er* n*, så inneholder* $F[x]/(s(x))$ q^n *elementer.*

Bevis. Se teorem 17.11 i [23] □

Eksempel 26: I dette eksempelet vil vi se på kroppen $GF(3^3)$ som har 9 elementer, denne kroppen kan konstrueres som en kvotientkropp mellom $Z_3[x]$ og kroppen som genereres av et irredusibelt tredjegradspolynom med koeffisienter fra $\mathbb{Z}_3$.

Polynomet $f(x) = x^2 + x + 2$ er irredusibelt i følge Teorem 28 fordi:

$$\begin{aligned} f(0) &= 2 \pmod 3 \\ f(1) &= 1 + 1 + 2 = 4 \equiv 1 \pmod 3 \\ f(2) &= 4 + 2 + 2 = 8 \equiv 2 \pmod 3 \end{aligned}$$

$$\mathbb{Z}_3[x]/\left\langle x^2 + x + 2 \right\rangle \simeq GF(3^3)$$

Vi har nå en ring hvor elementene er restklasser modulo $x^2 + x + 2$, dette gir følgende restklasser:

$$\begin{aligned} 0 &+ \left\langle x^2 + x + 2 \right\rangle \\ 1 &+ \left\langle x^2 + x + 2 \right\rangle \\ 2 &+ \left\langle x^2 + x + 2 \right\rangle \\ x &+ \left\langle x^2 + x + 2 \right\rangle \\ x + 1 &+ \left\langle x^2 + x + 2 \right\rangle \\ x + 2 &+ \left\langle x^2 + x + 2 \right\rangle \\ 2x &+ \left\langle x^2 + x + 2 \right\rangle \\ 2x + 1 &+ \left\langle x^2 + x + 2 \right\rangle \\ 2x + 2 &+ \left\langle x^2 + x + 2 \right\rangle \end{aligned}$$

Addisjon og multiplikasjon i denne kroppen foregår ved å regne med elementer fra restklassene:

$$(x + 1)(x + 2) \equiv x^2 + 3x + 2 \equiv x^2 + 3x + 2 - (x^2 + x + 2) \equiv 2x \pmod{x^2 + x + 2}$$

I avsnittet om idealer nevnte vi at idealer har samme rolle blant ringer som normale undergrupper har blant grupper. Vi skal nå se litt nærmere på hva vi mener med dette. I kapittelet om grupper så vi på kvotientgrupper, som dannes mellom en gruppe og en normal undergruppe; tilsvarende vil en ring og en underring danne en kvotientring dersom underringen er et ideal.

Dersom vi har en kropp R og et ideal $I \subseteq R$, så vil kvotienten R/I danne en ny kropp som består av restklassene av typen $a + I$ eller $[a]$, hvor $a \in R$. Vi sier at $a \in R$ er kongruent med $b \in R$ modulo I, og skrives $a \equiv b \pmod I$ dersom $a - b \in I$.

Tabell A.3: Multiplikasjon modulo $x^2 + x + 2$

$\cdot_3$	0	1	2	x	$x+1$	$x+2$	$2x$	$2x+1$	$2x+2$
0	0	0	0	0	0	0	0	0	0
1	0	1	2	x	$x+1$	$x+2$	$2x$	$2x+1$	$2x+2$
2	0	2	1	$2x$	$2x+2$	$2x+1$	x	$x+2$	$x+1$
x	0	x	$2x$	$2x+1$	1	$x+1$	$x+2$	$2x+2$	2
$x+1$	0	$x+1$	$2x+2$	1	$x+2$	$2x$	2	x	$2x+1$
$x+2$	0	$x+2$	$2x+1$	$x+1$	$2x$	2	$2x+2$	1	x
$2x$	0	$2x$	x	$x+2$	2	$2x+2$	$2x+1$	$x+1$	1
$2x+1$	0	$2x+1$	$x+2$	$2x+2$	x	1	$x+1$	2	$2x$
$2x+2$	0	$2x+2$	$x+1$	2	$2x+1$	x	1	$2x$	$x+2$

Tabell A.4: Addisjon modulo $x^2 + x + 2$

$+_3$	0	1	2	x	$x+1$	$x+2$	$2x$	$2x+1$	$2x+2$
0	0	1	2	x	$x+1$	$x+2$	$2x$	$2x+1$	$2x+2$
1	1	2	0	$x+1$	$x+2$	x	$2x+1$	$2x+2$	$2x$
2	2	0	1	$x+2$	x	$x+1$	$2x+2$	$2x$	$2x+1$
x	x	$x+1$	$x+2$	$2x$	$2x+1$	$2x+2$	0	1	2
$x+1$	$x+1$	$x+2$	x	$2x+1$	$2x+2$	$2x$	1	2	0
$x+2$	$x+2$	x	$x+1$	$2x+2$	$2x$	$2x+1$	2	0	1
$2x$	$2x$	$2x+1$	$2x+2$	0	1	2	x	$x+1$	$x+2$
$2x+1$	$2x+1$	$2x+2$	$2x$	1	2	0	$x+1$	$x+2$	x
$2x+2$	$2x+2$	$2x$	$2x+1$	2	0	1	$x+2$	x	$x+1$

A.4.4 Galoiskropper

I dette avsnittet vil vi se at det alltid finnes en kropp med orden p^n, hvor p er et primtall og $n \in \mathbb{Z}^+$. Det første teoremet sier at alle endelige kropper må ha en orden som kan skrives på denne måten.

Teorem 31. *En endelig kropp F har orden p^n hvor p er et primtall og $n \in \mathbb{Z}^+$.*

Bevis. Se teorem 17.13 i [23]. □

Videre har vi et teorem som sier at det finnes en kropp med størrelse p^n, for alle primtall p og positive hele tall $n \in \mathbb{Z}^+$.

Teorem 32. *En endelig kropp $GF(p^n)$ med p^n elementer eksisterer for hver p^n, hvor p er et primtall og $n \in \mathbb{Z}^+$.*

Bevis. Se teorem 33.10 i [12]. □

Til slutt har vi et teorem som sier at det bare finnes en kropp med orden p^n, alle andre kropper med samme størrelse er isomorfe med denne kroppen. Kropper av orden p^n kalles ofte Galoiskropp av orden p^n, og vi skriver $GF(p^n)$.

Teorem 33. *La p være et primtall og la $n \in \mathbb{Z}^+$. Hvis E og E' er kropper av orden p^n, så er $E \simeq E'$.*

Bevis. Se teorem 33.12 i [12]. □

Tillegg B

Kompilering og installasjon av LiDIA

Vi begynte arbeidet med LiDIA med å laste ned kildekoden fra LiDIAs hjemmeside, i vårt tilfelle versjon LiDIA-2.1pre7.tar.gz. Vi meldte oss også på to LiDIA-relaterte mailinglister for å holde oss oppdatert på utviklingen. Ifølge dokumentasjonen er det enklest å få LiDIA til å virke i Unix-lignende operativsystemer, så vi valgte å installere Linux.

Etter å ha tatt backup av systemet, installerte vi Mandrake 9.0 på ledig plass på Windows-partisjonen, slik at vi slapp å repartisjonere harddisken manuelt. Vi benyttet her vanlig og ikke ekspert-modus for installasjon. Når vi fikk velge pakker å installere, krysset vi av for å få utviklingsverktøy, c-kompilator etc. Vi valgte å få velge individuelle pakker, og la der til tetex for å få latex-støtte, samt libgmp-devel som er headerfiler og statisk bibliotek som kreves av LiDIA. Tilslutt opprettet jeg en vanlig bruker i tillegg til root (systemadministrator-funksjonen i Unix).

Vi pakket ut LiDIAs kildekode i hjemmekatalogen med kommandoen

```
tar zxvf LiDIA-2.1pre7.tar.gz
```

Vi får da et katalogtre som vi går til roten av med kommandoen

```
cd LiDIA-2.1pre7
```

Deretter tilpasser vi kompileringsinstillingene til systemet med *configure*

```
./configure
```

Nå genereres Makefile og diverse andre filer som bestemmer hvordan LiDIA skal kompileres. Vi fant senere ut at det må gjøres en liten tilpasning for at systemet skal fungere; man må redigere filen include/LiDIA/path.h som ble generert av *configure*, slik at stiene til databaser brukt av punkttellingspakken eco blir riktige. I path.h defineres 5 stier, alle begynner med NONE/..., og dette skal byttes ut med /usr/local/... dersom man installerer LiDIA i /usr/local slik som er default.

Deretter kjører man kommandoene

```
make (kompilere) su (bli root) make install (default i /usr/local)
Evt. make dvi (for dokumentasjon) make examples
```

Tilslutt måtte vi legge */usr/local/lib* til i filen */etc/ld.so.conf* og deretter kjøre kommandoen *ldconfig* for at programfilene som genereres skal finne delte biblioteker fra LiDIA.

Tillegg C

Kildekode

C.1 Innledning

Her er utdrag av kildekoden til vår implementasjon av operasjoner på elliptiske kurver og på punkter på disse.

C.2 Funksjoner

```
galois_field F;
//hjelpevariabler for hesseformoperasjoner
gf_element x3, y3, z3, hx3,hy3,hz3;
//hjelpevariabler for projektive weierstrassoperasjoner
gf_element h1, h2, h3, h4, h5, h6, h7, hh,h8, h9;
int i;
bigint m, m3;
bool mix, zz1,zz2;

// simple_point inneholder i tillegg til punktinformasjon (x,y,z)
// en peker til et simple_point til bruk i lenkede lister
typedef struct simple_point {

    gf_element x,y,z;
    simple_point *next;
};

simple_point ssh,h, * Q, * P;

base_vector< gf_element> __weierstrass_to_hesse(const gf_element &a4, const
gf_element &a6) {
    base_vector< gf_element> D(0,EXPAND);
    int n = 0;
    galois_field F=a4.get_field();
    gf_element jw(F), delta(F);
    bigint p=a4.get_field().characteristic();
    delta=-16*(4*a4*a4*a4+27*a6*a6);
```

```
    jw=-1728*64*a4*a4*a4/delta;
    bigint jjww=jw.polynomial_rep()[0];
    Fp_polynomial f;
    base_vector< bigint> koff(5, FIXED);
    koff[4] = 1;
    koff[3] = 648-jjww;
    koff[2] = 81*jjww+139968;
    koff[1] = 10077696-2187*jjww;
    koff[0] = 19683*jjww;
    f.assign (koff, p);
    if(prob_irred_test(f))
        return D;
    base_vector<bigint> roots=find_roots(f);
    if(roots.get_size()<1)
        return D;
    int ant_roots = (int)roots.get_size();
    Fp_polynomial g;
    base_vector< bigint> koff_g(4, FIXED);
    koff_g[3] = 1;
    koff_g[2] = 0;
    koff_g[1] = 0;
    for (int i=0;i<ant_roots;i++) {
        koff_g[0] = -roots[i];
        g.assign (koff_g, p);
        if(prob_irred_test(g))
            continue;
        base_vector<bigint> g_roots=find_roots(g);
        for (int j=0;j<g_roots.get_size();j++) {
            gf_element tmp_d(F),tmp_a4(F),tmp_a6(F);
            tmp_d.assign(g_roots[j]);
            D.insert_at(tmp_d, 0);
        }
    }
    return D;
}

void twist_weierstrass_to_hesse(gf_element &D, const gf_element &a4, const
gf_element &a6) {
    base_vector< gf_element> DV = _weierstrass_to_hesse(a4, a6);
    base_vector< gf_element> DD(0,EXPAND);
    for(int i = 0; i < DV.size(); i ++) {
        gf_element tmp_d(a4.get_field()), tmp_a4(a4.get_field()),
tmp_a6(a4.get_field());
        tmp_d.assign(DV[i]);
        hesse_to_weierstrass(tmp_d,tmp_a4,tmp_a6);
        if(!(a4==tmp_a4 && a6==tmp_a6)) {
            DD.insert_at(tmp_d, 0);
        }
    }
}
```

```
bool weierstrass_to_hesse(gf_element &D, const gf_element &a4, const
gf_element &a6) {
    base_vector< gf_element> DV = __weierstrass_to_hesse(a4, a6);
    for(int i = 0; i < DV.size(); i ++) {
        gf_element tmp_d(a4.get_field()), tmp_a4(a4.get_field()),
tmp_a6(a4.get_field());
        tmp_d.assign(DV[i]);
        hesse_to_weierstrass(tmp_d,tmp_a4,tmp_a6);
        if(a4==tmp_a4 && a6==tmp_a6) {
            D.assign(tmp_d);
            return true;
        }
    }
}

bool random_hesse_point(simple_point &res, const gf_element &gfD) {
    bigint p = gfD.get_field().characteristic();
    bigint D = gfD.polynomial_rep()[0], x,xx,y;
    res.x.assign_zero(p);
    res.y.assign_zero(p);
    res.z.assign_zero(p);
    Fp_polynomial f;
    base_vector< bigint> koff(4, FIXED);
    xx.randomize (p);
    do {
        x = xx;
        koff[3] = 1;
        koff[2] = 0;
        koff[1] = - D*x;
        koff[0] = 1 + x*x*x;
        f.assign ( koff, p);
        xx++;
        remainder(xx,xx,p);

    } while(prob_irred_test ( f));

    y = find_root ( f);

    res.x.assign(x);
    res.y.assign(y);
    res.z.assign(1);
    remainder(xx,x*x*x + y*y*y + 1 - D*x*y, p);

    if(!xx.is_zero()) {
        return false;
    } else return true;

}
```

```
// antar følgende likning X∧3 + y∧3 + z∧3 = 3Dxyz
simple_point * find_points(const gf_element &D) {
    const galois_field &F = D.get_field();
    const bigint p = F.characteristic();
    simple_point *root , *lp;
    lp = root = NULL;
    // Leter først opp alle punkter hvor z = 1
    gf_element x(F),y(F),z(F), vs(F), hs(F);
    int n = 0;
    z.assign(1);
    for(int i = 0; i < p; i++) {
        x.assign(i);
        for(int j = 0; j < p; j++) {
            y.assign(j);
            vs = x*x*x + y*y*y + z*z*z;
            hs = D*x*y*z;
            if(vs == hs) {
                if(root == NULL) {
                    lp = root = new simple_point(F);
                    root→next = NULL;
                } else {
                    lp→next = new simple_point(F);
                    lp = lp→next;
                    lp→next = NULL;
                }
                lp→x = x;
                lp→y = y;
                lp→z = z;
                n++;
            }
        }
    }

    // Leter opp alle punkter hvor z = 0 og y = 1
    z.assign_zero(F);
    y.assign(1);
    for(int j = 0; j < p; j++) {
        x.assign(j);
        vs = x*x*x + y*y*y + z*z*z;
        hs = D*x*y*z;
        if(vs == hs) {
            if(root == NULL) {
                lp = root = new simple_point(F);
                root→next = NULL;
            } else {
                lp→next = new simple_point(F);
                lp = lp→next;
                lp→next = NULL;
            }
            lp→x = x;
```

```
            lp→y = y;
            lp→z = z;
            n++;
            }
        }
    return root;
}

//konverterer fra D til a4 og a6 (Hesse til kort Weierstrassform)
void hesse_to_weierstrass(const gf_element &D, gf_element &a, gf_element
&b) {
    gf_element g2, g3 ;
    const galois_field &F = D.get_field();
    g2.assign_zero(F);
    g3.assign_zero(F);
    g2 = D*D*D*D/12 + D * 18;
    g3 = 27 - D*D*D*D*D*D / 216 + D*D*D*5 / 2;
    a = - g2/4;
    b = -g3/4;
}

bool simple_equal(const simple_point &p1, const simple_point &p2) {
    if(p1.x==p2.x && p1.y==p2.y && p1.z==p2.z)
        return true;
    return false;
}

void simple_assign(simple_point &res, const simple_point &p1) {
    res.x=p1.x;
    res.y=p1.y;
    res.z=p1.z;
}

void simple_assign_zero_hesse(simple_point &p1,galois_field &F) {
    p1.x.assign_zero(F);
    p1.y.assign_zero(F);
    p1.z.assign_zero(F);
    p1.x.assign_one();
    p1.y.assign(-1);
    p1.z.assign_zero();
}

void simple_negate_hesse(simple_point &res, const simple_point &p1) {
    hx3=p1.x;
    hy3=p1.y;
    res.x=hy3;
    res.y=hx3;
    res.z=p1.z;
}
```

```
void simple_add_hesse(simple_point &res, const simple_point &p1, const
simple_point &p2) {
    if (p1.z.is_zero()) {
        simple_assign(res, p2);
        return;
    }
    if (p2.z.is_zero()) {
        simple_assign(res, p1);
        return;
    }
        multiply(h1,p1.x,p2.y);
    if (p1.z!=1) {
        multiply(h5,p1.z,p2.y);
        multiply(h3,p1.z,p2.x);
    } else {
        h5=p2.y;
        h3=p2.x;
    }
    multiply(h4,p1.y,p2.x);
    if (p2.z!=1) {
        multiply(h2,p1.y,p2.z);
        multiply(h7,p1.x,p2.z);
    } else {
        h2=p1.y;
        h7=p1.x;
    }
    multiply(h6,h2,h7);
    multiply(h2,h2,h4);
    multiply(h4,h3,h4);
    multiply(h3,h3,h5);
    multiply(h5,h1,h5);
    multiply(h1,h1,h7);
    subtract(res.y,h1,h4);
    subtract(res.x,h2,h5);
    subtract(res.z,h3,h6);
    return;
}

void simple_subtract_hesse(simple_point &res, const simple_point &p1, const
simple_point &p2) {
    simple_negate_hesse(ssh,p2);
    simple_add_hesse(res,p1,ssh);
}

// 3 square + 6 multiply + 3 sub
void simple_double_hesse(simple_point &res, const simple_point &p1) {
    square(hx3,p1.x);
    multiply(hx3,hx3,p1.x);
    square(hy3,p1.y);
    multiply(hy3,hy3,p1.y);
```

```
    square(hz3,p1.z);
    multiply(hz3,hz3,p1.z);
    subtract(x3,hz3,hx3);
    multiply(x3,x3,p1.y);
    subtract(y3,hy3,hz3);
    multiply(y3,y3,p1.x);
    subtract(z3,hx3,hy3);
    multiply(z3,z3,p1.z);
    res.x=x3;
    res.y=y3;
    res.z=z3;
}

//her bruker vi samme algoritme som LiDIA bruker for projektive
// koordinater i kort Weierstrassform, modifisert for aa passe til hesseform
void simple_multiply_hesse(simple_point &res, const bigint mult, const
simple_point &p1) {
    F=p1.x.get_field();
    m=mult;
    m3=3*m;
    simple_assign(h,p1);

    if (m.is_zero()) {
        simple_assign_zero_hesse(res,F);
        return;
    }
    if (m.is_negative()) {
        m.negate();
        m3.negate();
        simple_negate_hesse(h, h);
    }
    if (m.is_one()) {
        simple_assign(res,h);
        return;
    }
    simple_assign_zero_hesse(res,F);
    for (i = m3.bit_length() - 1; i > 0; i--) {
        simple_double_hesse(res, res);
        if (m.bit(i) == 0 && m3.bit(i) == 1)
            simple_add_hesse(res, res, h);
        else if (m.bit(i) == 1 && m3.bit(i) == 0)
            simple_subtract_hesse(res, res, h);
    }
}


bool h_equ(const simple_point &Denne, const simple_point &DenAndre) {
        // Dersom z != 0 bruker
        if(!Denne.z.is_zero()) {
            //Tilfelle hvor z1 != 0 og z2 = 0
```

```
            if(DenAndre.z.is_zero()) return false;
            // Tilfelle hvor z1 != 0 og z2 != 0
            if(DenAndre.x * Denne.z == DenAndre.z * Denne.x &&
            DenAndre.y *Denne. z == DenAndre.z * Denne.y) return true;
            else return false;
        } else {
            // Tilfell hvor z1 = 0 og z2 != 0
            if(!DenAndre.z.is_zero()) return false;
            // Tilfelle hvor z1 = z2 = 0
            if(DenAndre.y * Denne.x == DenAndre.x * Denne.y) return true;
            return false;
        }

    }

void proj2Affine(simple_point &p ) {
    if(p.z.is_zero()) return;
    p.x = p.x/(p.z);
    p.y= p.y/(p.z);
    p.z = 1;
}

void PReCalcHesseMult(const simple_point &p1, int bit) {
    P = new simple_point[bit];
    P[0] =p1;
    proj2Affine(P[0] );
    for(i = 1; i < bit; i++){
        simple_double_hesse(P[i],P[i-1]);
        proj2Affine(P[i] );
    }
}

//her bruker vi samme algoritme som LiDIA bruker for projektive koordinater
// i kort Weierstrassform, modifisert for aa passe til hesseform
void simple_pre_multiply_hesse(simple_point &res, const bigint mult) {
    simple_assign(h,P[0]);
    F= h.x.get_field();
    m=mult;
    m3=3*m;
    if (m.is_zero()) {
        simple_assign_zero_hesse(res,F);
        return;
    }

    if (m.is_negative()) {
        m.negate();
        m3.negate();
        simple_negate_hesse(h, h);
    }
    if (m.is_one()) {
```

```
        simple_assign(res,h);
        return;
    }
    simple_assign_zero_hesse(res,F);
    int n = m3.bit_length();
    for (i = 0; i < (n-1); i++) {
        if (m.bit(i+1) == 0 && m3.bit(i+1) == 1)
            simple_add_hesse(res, res, P[i]);
        else if (m.bit(i+1) == 1 && m3.bit(i+1) == 0)
            simple_subtract_hesse(res, res, P[i]);
    }
}

void simple_add_proj(simple_point &p3, const simple_point &p1, const
simple_point &p2, const gf_element &a4) {
    if(p2.z.is_one()) mix = true;
    else mix = false;
    if(p2.z.is_zero()) {
        p3.x=p1.x;
        p3.y=p1.y;
        p3.z=p1.z;
        return;
    }
    if(p1.z.is_zero()) {
        p3.x=p2.x;
        p3.y=p2.y;
        p3.z=p2.z;
        return;
    }
    h1 = p1.x;
    h2 = p2.x;
    h4 = p1.y;
    p3.z = p1.z;
    if(!mix)
        square(h4, p2.z ); //z2^2 Mix SQ
    square(h5, p1.z ); // z1^2 SQ
    if(!mix)
        multiply(h1,h1,h4); // x1*z2^2 Mix M
    multiply(h2,h2,h5); // x2*z1^2 M
    subtract(h3, h1,h2); // h1 - h2 SU
    if(!mix) {
        multiply(h4, p1.y, h4 ); // y1*z2^2 Mix M
        multiply(h4,h4,p2.z); // y1*z2^3 Mix M
    }
    multiply(h5,p2.y,h5); // y2*z1^2 M
    multiply(h5,h5,p1.z); // y2*z1^3 M
    subtract(h6,h4,h5); // h4 - h5 SU
    add(h2,h1,h2); // h1 + h2 A
    add(h5, h4,h5); // h4 + h5 A
    if(!mix)
```

```
        multiply(p3.z, p1.z,p2.z); // z1*z2 Mix M
    multiply(p3.z, p3.z,h3); // z1*z2*h3 M
    square(h1, h3); // h3^2 SQ
    multiply(h4, h2,h1); // h7*h3^2 M
    square( p3.x, h6 ); // h6^2 SQ
    subtract(p3.x, p3.x, h4 ); // h6^2 - h7*h3^2 SU
    p3.y = p3.x;
    p3.y.multiply_by_2(); // 2*x3 M2
    subtract(p3.y, h4,p3.y ); // h7*h3^2 - 2*x3 SU
    multiply(p3.y,p3.y,h6 ); // h9*h6 M
    multiply(h5,h5,h3 ); // h8*h3 M
    multiply(h5,h5,h1 ); // h8*h3^3 M
    subtract(p3.y, p3.y, h5); // h9*h6 - h8*h3^3 SU
    p3.y.multiply_by_2(); // M2
    p3.y.multiply_by_2(); // M2
    p3.x.multiply_by_2(); // M2
    p3.x.multiply_by_2(); // M2
    p3.z.multiply_by_2(); // M2
    // 12M + 4SQ + 2A + 5SU + 6M2
}

// 6 square + 4 multiply + 4 add + 3 subtract + 6 multiply_by_2
void simple_double_proj(simple_point &res, const simple_point &p1, const
gf_element &a4) {
    if (p1.x==0 && p1.y==1 && p1.z==0);
        simple_assign(res,p1);
    h1=p1.x;
    h2=p1.y;
    h3=p1.z;
    square(h5, h3);
    square(h5, h5);
    multiply(h5, a4, h5);
    square(h4, h1);
    add(hh, h4, h4);
    add(h4, hh, h4);
    add(h4, h4, h5);
    multiply(h3, h2, h3);
    h3.multiply_by_2();
    square(h2, h2);
    multiply(h5, h1, h2);
    h5.multiply_by_2();
    h5.multiply_by_2();
    square(h1, h4);
    add(hh, h5, h5);
    subtract(h1, h1, hh);
    square(h2, h2);
    h2.multiply_by_2();
    h2.multiply_by_2();
    h2.multiply_by_2();
    subtract(h5, h5, h1);
```

```
    multiply(h5, h4, h5);
    subtract(h2, h5, h2);
    res.x=h1;
    res.y=h2;
    res.z=h3;
}

void simple_subtract_proj(simple_point &res, const simple_point &p1, const
simple_point &p2, const gf_element &a4) {
    simple_negate_proj(ssh, p2);
    simple_add_proj(res, p1, ssh, a4);
}

void simple_negate_proj(simple_point &res, const simple_point &p1) {
    res.x=p1.x;
    negate(res.y,p1.y);
    res.z=p1.z;
}

void simple_multiply_proj(simple_point &res, const bigint mult, const
simple_point &p1, const gf_element &a4) {
    m=mult;
    m3=3*m;
    simple_assign(h,p1);
    if (m.is_zero()) {
        res.x=0;
        res.y=1;
        res.z=0;
        return;
    }
    if (m.is_negative()) {
        m.negate();
        m3.negate();
        simple_negate_proj(h, h);
    }
    if (m.is_one()) {
        simple_assign(res,h);
        return;
    }
    res.x=0;
    res.y=1;
    res.z=0;
    for (i = m3.bit_length() - 1; i > 0; i--) {
        simple_double_proj(res, res, a4);
        if (m.bit(i) == 0 && m3.bit(i) == 1)
            simple_add_proj(res, res, h, a4);
        else if (m.bit(i) == 1 && m3.bit(i) == 0)
            simple_subtract_proj(res, res, h, a4);
    }
}
```

```
void Jacobian2Affine(simple_point &p ) {
    if(p.z.is_zero()) return;
    p.x = p.x/(p.z*p.z);
    p.y= p.y/(p.z*p.z*p.z);
    p.z = 1;
}

void PReCalcWeierstrassMult(const simple_point &p1, int bit, const
gf_element &a4) {
    Q = new simple_point[bit];
    Q[0] =p1;
    Jacobian2Affine(Q[0]);
    for(i = 1; i < bit; i++){
        simple_double_proj(Q[i],Q[i-1],a4);
        Jacobian2Affine(Q[i]);
    }
}

//her bruker vi samme algoritme som LiDIA bruker for projektive koordinater
// i kort Weierstrassform, modifisert for aa passe til hesseform
void simple_pre_multiply_weierstrass(simple_point &res, const bigint mult ,
const gf_element &a4) {
    simple_assign(h,Q[0]);
    F=h.x.get_field();
    m=mult;
    m3=3*m;
    if (m.is_zero()) {
        res.x=0;
        res.y=1;
        res.z=0;
        return;
    }
    if (m.is_negative()) {
        m.negate();
        m3.negate();
        simple_negate_proj(h, h);
    }
    if (m.is_one()) {
        simple_assign(res,h);
        return;
    }
    int n = m3.bit_length();
    res.x=0;
    res.y=1;
    res.z=0;
    for (i = 0; i < (n-1); i++) {
        if (m.bit(i+1) == 0 && m3.bit(i+1) == 1)
            simple_add_proj(res, res, Q[i],a4);
        else if (m.bit(i+1) == 1 && m3.bit(i+1) == 0)
```

```
            simple_subtract_proj(res, res, Q[i],a4);
    }
}
```

Tillegg D

Eksempler på elliptiske kurver

D.1 Innledning

Her er noen eksempler på elliptiske kurver, de oppgis med karakteristikk p for kroppen, n som er bitlengden til p, D-parameter for Hesseform, orden til kurven$\#E(F_p)$, og a- og b-parametere til birasjonalt ekvivalente kurver på kort Weierstrassform.

Kurvene er generert som beskrevet i avsnitt 7.8, alle har orden 3 ganger et primtall.

D.2 Kurver

Kurve 1:

$$
\begin{aligned}
n &= 160 \\
\#E(\mathbb{F}_p) &= 3 \cdot 408251189305084508533625839539957518966956101071 \\
p &= 1224753567915253525600877180059052116597297173971 \\
D &= 155084242162794225825732878535100753203309440242 \\
a &= 180890127234310861440619063553097796467445303876 \\
b &= 638723106561030470678231670371932421650351389855
\end{aligned}
$$

Kurve 2:

$$
\begin{aligned}
n &= 162 \\
\#E(\mathbb{F}_p) &= 3 \cdot 162378588177467866043420331205760706302230354 7623 \\
p &= 4871357645324035981302607454822983909742787176879 \\
D &= 2751348526123464356756459439581152254411380715462 \\
a &= 1404584638382299303272727156390801290408142289418 \\
b &= 1475313448485476636474334759640105873560319054604
\end{aligned}
$$

Kurve 3:

$$n = 162$$
$$\#E(\mathbb{F}_p) = 3 \cdot 1434059863474691732434459203386481366862736770031$$
$$p = 430217959042407519730337471243332918110298285924 7$$
$$D = 1423073835129527915483328481964884039120615974314$$
$$a = 859575705654501397198281906706239017894510845632$$
$$b = 140225064883545229860749446992680010465048619838 0$$

Kurve 4:

$$n = 171$$
$$\#E(\mathbb{F}_p) = 3 \cdot 498859288848208660730142879303425025572313088603367$$
$$p = 149657786654462598219042865780615286795236620513592 1$$
$$D = 833264864988720759488315794143041007542981047044402$$
$$a = 466549275347861062401042732147059358455915026263949$$
$$b = 121380497061522943707792873288640913281531978298446 0$$

Kurve 5:

$$n = 171$$
$$\#E(\mathbb{F}_p) = 3 \cdot 498859598290454901346223002113069814523218692612761$$
$$p = 149657879487136470403866897973138917238783844160094 7$$
$$D = 124309164872415644924619214023065032610374660209785 2$$
$$a = 215675680992052149973424008629697396906654760539748$$
$$b = 468063830405003170086486879685397863187636506906923$$

Kurve 6:

$$n = 171$$
$$\#E(\mathbb{F}_p) = 3 \cdot 498859556685843202654502261453342398598066999326477$$
$$p = 149657867005752960796350677902229260295924945861950 7$$
$$D = 463321042146659535905506886599134396317636269486436$$
$$a = 202240723435918001127374986030149602398016650729186$$
$$b = 199438923365881611901958740509542294500230224256659$$

Kurve 7:

$$n = 240$$
$$\#E(\mathbb{F}_p) = 3 \cdot 56402387370342889971378063213722336581827064017500807068379783198811271 1$$
$$p = 16920716211102866991413418964116700961959871317136245022362607751814061 03$$
$$D = 70249723857389687569279996011413629722731041382076985034755825112097874 9$$
$$a = 43164347410179053180950770507349714338925522818022387686039349453284925 0$$
$$b = 99389074975005479737457070261834722858597177977582360247756159869188718 3$$