T. C.
GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
MÜHENDİSLİK ve FEN BİLİMLERİ ENSTİTÜSÜ

FARKLI EĞRİ FORMLARI İÇİN ELİPTİK EĞRİ NOKTA İŞLEMLERİ YAPAN C# KÜTÜPHANESİ GELİŞTİRİLMESİ

ERGİN ÖZTÜRK

YÜKSEK LİSANS TEZİ
ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

GEBZE
2014

T.C.
GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
MÜHENDİSLİK ve FEN BİLİMLERİ ENSTİTÜSÜ

FARKLI EĞRİ FORMLARI İÇİN ELİPTİK EĞRİ NOKTA İŞLEMLERİ YAPAN C# KÜTÜPHANESİ GELİŞTİRİLMESİ

ERGİN ÖZTÜRK

YÜKSEK LİSANS TEZİ
ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMAN
YRD. DOÇ. DR. SERDAR SÜER ERDEM

GEBZE
2014

GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ	YÜKSEK LİSANS JÜRİ ONAY FORMU

GYTE Mühendislik ve Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 27.01.2014 tarih ve 2014/06 sayılı kararıyla oluşturulan jüri tarafından 05/05/2014 tarihinde tez savunma sınavı yapılan Ergin ÖZTÜRK'ün tez çalışması Elektronik Mühendisliği Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE
(TEZ DANIŞMANI) : Yrd. Doç Dr. Serdar Süer ERDEM

ÜYE : Yrd. Doç. Dr. Köksal HOCAOĞLU

ÜYE : Doç. Dr. Hacı Ali MANTAR

ONAY

GYTE Mühendislik ve Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... tarih ve ...../..... sayılı kararı.

İMZA/MÜHÜR

ÖZET

Eliptik eğri algoritmalarında verimlilik ve performans, nokta işlemlerinin hızına bağlıdır. Nokta işlemlerinde skaler çarpma operasyonu en önemli özelliktir. Bir nokta ile skaler çarpma operasyonu yapılırken, bu işlem içerisinde nokta toplama ve nokta çiftleme ardışıl işlemleri (double-and-add) bulunmaktadır. Farklı koordinat sistemlerinde bu iki işlemin gerçeklenebilmesi için ise modüler çarpmaya göre ters alma, modüler çarpma ve modüler toplama gibi işlemlere ihtiyaç vardır.

Çarpmaya göre ters alma işlemi hem işlev olarak hem de donanım olarak gerçeklenmesi yüksek maliyetlere neden olduğundan, değişkenler üzerinde dönüşümler yapılarak projektif koordinat sistemlerine geçilirse bu maliyetler azaltılabilmektedir. Projektif koordinat sistemlerinin Jacobian, Chudnovsky, Extended Jacobian, Inverted Projective ve Lopez Dahap gibi farklı formları da mevcuttur.

İkili sonlu alan veya daha yüksek güvenlik sağlayan karekteristiği büyük sonlu alanlarda eliptik eğriler üzerinde çok hızlı nokta çarpma ve nokta toplama işlemi yapabilen eliptik eğri algoritmaları geliştirilmiştir. Bunlardan teze konu olan bazı eliptik eğri formları; Jacobian eğri formu, Edwards eğri formu, Huff eğri formu, Hessian eğri formu ve bunların varyasyonlarıdır.

Bu tezde eliptik eğrilerin matematiksel temelleri incelenmiş, Microsoft .Net geliştirme ortamında Framework 4.0 tabanında BigInteger kütüphanesi kullanılarak modüler aritmetik, sonlu alan aritmetik işlemleri tanımlanmış ve eliptik eğrilerde nokta toplama, nokta çiftleme işlemleri farklı eliptik eğri formları ve farklı koordinatlarda uygulamaları gerçekleştirilerek bir eliptik eğri kütüphanesi hazırlanmıştır. Uygulamada C# programlama dili kullanılmıştır.

Anahtar kelimeler: Eliptik eğriler, eliptik eğri formları, Big integer, nokta toplama, nokta çiftleme, c# programlama.

SUMMARY

Efficiency and performance in elliptic curve algorithm depend on the speed of point operations. Scalar multiplication operation is the most important feature of point operation in ECC. There are series of point doubling and point addition operations in scalar multiplication of a point. These two operations are required operations such as modular inversion, modular multiplication, and modular addition to be implemented on different projective coordinates.

Due to the fact that the implementation of the modular inversion costs very high, the projective coordinates are used to reduce these costs. The Projective coordinate systems have many different forms such as Jacobian, Chudnovsky, Extended Jacobian, Inverted Projective and Lopez Dahap.

Elliptic curve algorithms have been developed to obtain faster point multiplications and point addition over the binary finite field or the large characteristic fields providing high level security. This master thesis concentrates on the various curve models Jacobian curves, Edwards curves, Huff curves, Hessian curves, and their variations.

In this thesis, the mathematical foundations of elliptic curve are examined. Modular arithmetic, field arithmetic operations are defined by using Big Integer library at the Microsoft .Net development environment Framework 4.0. An elliptic curve library was prepared by performing point addition and point doubling operations in different elliptic curve forms and different coordinates applications by elliptic curve. C # programming language has been used in this practice.

Keywords: Elliptic curve, alternate models of elliptic curves, Big integer, point addition, point doubling, c# programming.

TEŞEKKÜR

Bu tez çalışması sırasında beni yönlendiren, yol gösteren ve yardımlarını hiçbir zaman esirgemeyen sayın hocam Yrd. Doç Dr. Serdar Süer ERDEM'e,

Göstermiş olduğu desteklerinden dolayı sevgili eşim Feride ÖZTÜRK'e en içten teşekkürlerimi sunarım.

İÇİNDEKİLER

SİMGELER ve KISALTMALAR DİZİNİ

Simgeler ve Kısaltmalar		**Açıklamalar**
Δ	.	Diskriminant
D	:	Sabit sayı ile çarpma işlemi
E	:	Eliptik eğri
F_p	:	p asal sayılı sonlu alan
GF(p)	:	Galois cismi
M	:	Modüler çarpma işlemi (Multiplication)
P, Q, R	:	Eliptik eğri noktaları
S	:	Modüler çarpma işlemi (Squaring)
sn	:	Saniye
(x,y)	:	Koordinat ekseni üzerinde x ve y ile belirlenen nokta
Add	:	Modüler toplama işlemi (Addition)
ALP	:	Ayrık Logaritma Problemi
DES	:	Digital Encyrption Standart (Sayısal Şifreleme Standardı)
DSA	:	Digital Signature Algorithm (Dijital İmza Algoritması)
DH	:	Diffie-Hellman
EEK	:	Eliptik Eğri Kriptografi
NIST	:	National Institute of Standards and Technology (Ulusal Standartlar ve Teknoloji Enstitüsü)
RSA	:	Rivest, Shamir, Adleman
Sub	:	Modüler çıkarma işlemi (Subtraction)

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

1. GİRİŞ

Teknolojinin ilerlemesiyle ve daha fazla insanın interneti kullanmaya başlamasıyla birlikte veri şifrelemenin önemi daha belirgin bir şekilde ortaya çıkmaktadır. Bir çok ülkede araştırmacılar ve özel şirketler şifreleme üzerine yoğunlaşıp kırılması zor olan algoritmalar üretmek için ciddi kaynaklar ayırmaktadırlar. Bilgilerimizin gizliliğini sağlayan şifreleme yöntemleri, günümüzde bilgisayarlar veya elektronik sistemler üzerinde uygulanarak işlemlerimizi güven içinde yapmamızı sağlamaktadır. Bunun bir sonucu olarak da elektronik ticaret hayatımızda önem kazanarak yaygınlaşmıştır.

Haberleşen iki veya daha fazla tarafın bilgi alışverişini güvenli olarak yapmasını sağlayan metot ve kavramları inceleyen bilim dalı kriptolojidir. Kriptoloji, temelde çözülmesi zor olan matematiksel problemlere dayanan teknikler ve uygulamalar içerir. Geliştirilen şifreleme algoritmalarının kırılma zorluğunu elde edebilmek için değişik matematiksel foksiyonları üretir. Yapılan matematiksel analizler ile de kırılamaz algoritmalar üzerine çalışmalar geliştirilmektedir. Şifreleme için kullanılan açık anahtar yapısına sahip RSA, DES, 3DES gibi bir çok şifreleme algoritması, aynı anahtar uzunluğuna sahip eliptik eğri şifrelemenin sağladığı güvenliğin yarısını sağlayabilmektedir. Bu yüzden araştırmacılar eliptik eğriler cismine yönelik çalışmalarına yoğunlaşmışlardır. Bununla birlikte eliptik eğrilerin dez avantajları da vardır. Diğer şifreleme tekniklerine göre işlemci gücü ve buna bağlı sorunların varlığı, araştırmacıları bu dezavantajları ortadan kaldırmaya yönelik çalışmalara sevketmiştir.

Eliptik eğriler üzerinde 19. yüzyıldan beri çalışıla gelmiştir. 1985 yılında eliptik eğriler üzerindeki noktalar grubunu kullanarak farklı ayrık logaritma şifreleme yöntemlerinin gerçeklenebileceği ortaya konmuş [1], [2] ardından, tamsayı çarpanlarına ayırma problemine dayanan açık anahtar şifreleme yöntemlerinin eliptik eğriler üzerinde de gerçekleştirilebileceği ve eliptik eğri gerçekleştirimlerinin daha üst düzeyde güvenlik sağladığı ispatlanmıştır.

Eliptik eğri kripto sistemi (EEK), RSA kripto sistemine alternatif olarak önerilmiştir. Küçük anahtar boyutu ile gerçeklenen EEK, RSA'ye göre daha az işlem maliyeti ve daha az bellek tüketimine gereksinim duymaktadır. Aynı anahtar uzunluğu kullanıldığında EEK daha hızlı bir kripto sistemdir [3]. Anahtar boyunun

kısalığı, hızlı hesaplamalar, daha az bellek alanı gereksinimi, daha az işlem gücü ve bant genişliği verimliliği sağlar.

Eliptik eğriler üzerinde tanımlı matematiksel işlemlerin hızlı bir şekilde gerçeklenebilmesi için çeşitli metotlar geliştirildi ve halen daha geliştirilmeye devam edilmektedir. Bu aritmetik işlemleri hızlandırmak için kullanılan metotlardan biri de "alternatif eliptik eğri modelleri" kullanmaktır. Jacobi intersections, Edward eğrileri, Jacobi quartic ve Hessian eğrileri önemli eliptik eğri modellerindendir. Bu çalışmada çeşitli eliptik eğri modelleri üzerinde incelemelerde bulunup, c# programlama dilinde Biginteger yapısı kullanılarak örnek bir kripto kütüphanesi gerçekleştirilmiştir.

Bu çalışmamızın ikinci bölümünde eliptik eğri kriptografi altyapısı için gerekli matematiksel temelleri hakkında özet bilgiler verildikten sonra üçüncü bölümde eliptik eğrilerin genel özellikleri ve grup oluşturma gibi eliptik eğrilerin genel karakteristikleri incelenmiştir.

Dordüncü bölümde eliptik eğri formları ele alınarak, bu formlar üzerinde nokta toplama ve nokta çiftleme işlemleri incelenerek algoritma çözümleri ortaya konmuştur.

Beşinci bölümde algoritma tasarımları verilen eliptik eğri formlarının c# dilinde BigInteger kütüphanesi kullanılarak yazılan uygulama anlatılmıştır. Uygulamada 2.4 Ghz'lik Intel Core 2 Quad işlemcili PC üzerinde yazılım uygulaması yapılmıştır. Uygulamalarda NIST tarafından tavsiye edilen, asal ve ikili alanlarda tanımlanmış eliptik eğriler kullanılmıştır. Uygulaması yapılan eliptik eğri formlarının nokta toplama ve nokta çiftleme işlemleri gerçeklenmiştir. Elde edilen zaman değerleri, metotlar arasında karşılaştırılarak optimum olan bulunmaya çalışılmıştır.

Son bölümde ise yapılan uygulamada kullanılan algoritmaların performans değerleri incelenmiştir.

2. MATEMATİKSEL TEMELLER

Bu bölümde eliptik eğri aritmetiği ile ilgili temel kavramlar açıklanmış olup özellikle de alan kavramı ve Galois alanlar üzerinde durulmuştur.

2.1. Grup

Üzerinde tersi alınabilir ve birleşme özelliğine sahip iki değişkenli bir operasyonun tanımlı olduğu kümeler *grup* olarak adlandırılır. Grup, boş olmayan bir G kümesi ve onun üzerinde tanımlı bir "Δ"işleminden oluşur. Bu operasyonun aşağıdaki şartları sağlaması gereklidir;

i) Δ işlemi G'de kapalıdır.($a\Delta b \in G$)
ii) Δ işlemi,$a, b, c \in G$ için birleşme özelliği sağlar. $a\Delta(b\Delta c) = (a\Delta b)\Delta c$
iii) Δ işleminin $e \in G$ şeklinde bir birim elemanı vardır. $a\Delta e = e\Delta a = a$
iv) Δ işlemine göre G'deki her elemanın tersi vardır. $a\Delta a^{-1} = a^{-1}\Delta a = e$

Tüm bunlara ek olarak ikili işlem değişme özelliğini sağlarsa $< G, \Delta >$ grubuna değişmeli grup (Abelian Grup) denir.

v) Δ işleminin $a, b \in G$ için değişme özelliği vardır. $a\Delta b = b\Delta a$

Örneğin tamsayılar kümesi ve toplama işlemi için <Z,+>, bir değişmeli gruptur.

2.2. Halka

R boş olmayan bir küme olsun. + ve * ikili işlemleri bu küme üzerinde tanımlı olsunlar. Eğer <*R*,+,*> cebirsel yapısı aşağıdaki aksiyomları sağlıyorsa bu cebirsel yapıya *halka* adı verilir.

i) <*R*,+> değişmeli gruptur.

ii) * işleminin *R* üzerinde kapalılık ve birleşme özelliği vardır.

iii) * işleminin + işlemi üzerine sağdan ve soldan dağılma özelliği vardır.

Bir halkanın + (toplama) işlemine göre etkisiz elemanına sıfır eleman denir. Halkanın * işlemine göre etkisiz elemanı varsa bu halkaya birimli halka denir. Ayrıca * işlemine göre değişmeli ise bu halkaya birimli ve değişmeli halka denir.

2.3. Alan

F sayı kümesi üzerinde tanımlanmış olan + ve * işlemleriyle birlikte aşağıdaki aksiyomları sağlıyorsa bir "Alan" oluşturur.

i) <R,+> bir değişmeli grup olmalıdır.

ii) <R,*> bir değişmeli grup olmalıdır. Sadece + işleminin sıfır elemanı için bir ters eleman bulunmaz.

iii) <R,+,*> bir halka olmalıdır. Yani yukarıdaki her iki koşula ek olarak * işleminin + işlemi üzerine dağılma özelliği olmalıdır.

Gerçel sayılar kümesi toplama ve çarpma işlemleri ile birlikte bir "Alan" oluşturmaktadır.

2.4. Sonlu Alan

Yukarıda tanımı verilen cisimler sonlu sayıda eleman içeriyorsa bu cisme "Sonlu Alan" denir. Sonlu alanın eleman sayısı o cismin derecesidir. Aynı sonlu sayıda elemana sahip, derecesi aynı alanlar eş yapıdadırlar. Sadece elemanların gösterişleri farklıdır [4].

Bir sonlu alanın var olabilmesi için o sonlu alanın derecesinin, p ve m sırasıyla asal ve tamsayı olmak üzere, $q = p^m$ şeklinde bir asalın kuvveti olması gerekir ve bu alan F_q şeklinde gösterilir.

2.5. Galois Alanı Kavramı

p asal sayı olmak koşuluyla, eleman sayısı p olan $Z = \{0,1,2, \dots, p-1\}$ p üzerinde modülo p toplama ve modülo p çarpma islemlerinin tanımlanmasıyla p sayıda elemana sahip bir sonlu alan oluşur ve p karakteristikli $GF(p)$ olarak adlandırılır [5].

- Örnek olarak; 3, 11, 9803, 1003039 gibi asal sayılar bu alanın elemanlarıdır.

$GF(p)$ üzerinde $q = p^m$ elemanlı $GF(p^m)$ alanı oluşturulabilir. Bu yeni alana $GF(p)$ alanının genişletilmiş alanı denir. p, $GF(p^m)$ alanının karakteristiği, m ise alanın genişleme derecesi adını alır. $GF(p^m)$ alanı p^m adet elemandan oluşur ve Galois alanı olabilmesi için m. dereceden bir indirgeme değeri kullanılır. Toplama ve çarpma işlemleri sonucunda oluşan sonucun $GF(p^m)$ sonlu alanı içerisinde kalması indirgeme değeri ile sağlanır. Bir sonlu alanda elemanların ifade edilebilmesi için polinom şeklinde bir gösterim kullanılabilir. Bu hem gösterim hem de işlemlerde kolaylık sağlayacaktır.

$GF(p^m)$ alanında p karakteristiğinin 2 seçilmesiyle $GF(2^m)$ alanı oluşturulur. İkili sistemde çalısmak matematiksel işlemlerin donanım ve yazılımsal olarak gerçeklenmesini kolaylaştırır. $GF(2^m)$ sonlu alanları ikili sonlu alanlar olarak adlandırılır ve elemanları katsayıları $\{0,1\}$'den oluşan polinomlardır [6].

2.6. Polinomlar

R halkası üzerinde $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots a_1 x + a_0$ şeklinde $a_i \in R$ olmak üzere, çok sayıda polinom tanımlanabilir ve olası tüm polinomlar kümesi $R[x]$ ile gösterilir.

Bir polinomun derecesi; katsayısı sıfırdan farklı olan x 'lerin en büyük üs değeriyle belirlenir. İki polinomun dereceleri ve katsayıları aynı ise, bunlara eşit polinomlar denir.

Polinom, birden fazla polinomun çarpımı olarak yazılabiliyorsa, bunların herbirine polinomun çarpanları adı verilir. Eger bir polinom, pozitif dereceli birden

fazla polinomun çarpımı şeklinde yazılamıyorsa indirgenemez (irreducible) polinom olarak ifade edilir.

Eliptik eğri tabanlı kriptografik uygulamalarda kullanılacak polinomun, indirgenemez bir polinom olması gerekir. Aksi takdirde, çarpan polinomların tanımladığı altgruplar ve izomorfizma kullanılarak, EEK kriptosisteme atak yapılabilir.

$GF(2^m)$ alanında toplama ve çarpma işlemleri NIST'in belirlemiş olduğu aşağıdaki çizelgede verilen indirgenemez polinomlar kullanılarak yapılmalıdır.,

Tablo 2.1: NIST indirgeme polinomları.

Alan	**İndirgeme Polinomları**
F_2^{113}	$f(x) = x^{113} + x^9 + 1$
F_2^{131}	$f(x) = x^{131} + x^8 + x^3 + x^2 + 1$
F_2^{163}	$f(x) = x^{163} + x^7 + x^6 + x^3 + 1$
F_2^{193}	$f(x) = x^{193} + x^{15} + 1$
F_2^{233}	$f(x) = x^{233} + x^{74} + 1$
F_2^{239}	$f(x) = x^{239} + x^{36} + 1$
F_2^{283}	$f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$
F_2^{409}	$f(x) = x^{409} + x^{87} + 1$
F_2^{571}	$f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$

3. ELİPTİK EĞRİLERE GİRİŞ

Kriptografide eliptik eğrilerin kullanımı Diffie-Helmann ve El-Gamal algoritmalarının uyarlanması ile olmaktadır. EEK'nın temelini, eliptik eğri üzerindeki bir noktanın ayrık logaritmasını alma zorluğu oluşturmaktadır.

RSA tabanlı açık anahtarlı şifreleme yöntemlerinden hesaplama yükü anahtar büyüklüğüyle orantılı olarak artmaktadır. EEK, RSA ve Diffie-Hellman'ın önerdiği güvenlik seviyesine çok daha kısa anahtarlarla ulaşılmasını sağlar. Daha küçük boyutlu anahtar kullanımından dolayı EEK, akıllı kartlar üzerinde kolaylıkla gerçeklenebilir. Anahtar boyutu kısa olduğu için bir çok protokolde el sıkışma(hand-shake) işlemi daha hızlı bir şekilde gerçekleştirilmektedir. Bu sebeple de kablosuz ağlarda EEK kullanımı daha yaygındır. Eşdeğer güvenlik seviyeleri için NIST'in önerdiği anahtar uzunlukları [7] aşağıdaki tabloda verilmiştir.

Tablo 3.1: NIST anahtar uzunlukları.

Güvenlik(bit)	**DSA/DH**	**RSA**	**EEK**
80	1024	1024	160
112	2048	2048	224
128	3072	3072	256
192	7680	7680	384
256	15360	15360	512

3.1. Eliptik Eğri Teorisi

Eliptik eğriler elips değildirler. Bu şekilde adlandırılmalarının sebebi, bir elipsin çemberinin hesaplanması için kullanılan kübik denkliklere benzer ifadeler ile gösterilmeleridir. Bir *K* cismi ele alınırsa, *K* cismi, *R* Gerçel sayılar, *Q* Rasyonel sayılar, *C*- Kompleks sayılar veya p'nin asal bir sayı oldugunu kabul edilirse, $q = p^r$ elemanlarından olusan Fq sonlu cismi olabilir. $GF(2)$ sonlu cisminin karakteristiği 2, gerçel ve kompleks sayıların karakteristiği ise sonsuzdur. Herhangi bir K alanı üzerinde tanımlı eliptik eğrinin genel denklemi;

$$y^2 + axy + by = x^3 + cx^2 + dx + e \tag{3.1}$$

ile ifade edilir ve Weierstrass denklemi olarak bilinir. Eğer K cisminin karakteristiği $Char(K) = 2$ ise;

$$y^2 + ay = x^3 + bx + c \tag{3.2}$$

$$y^2 + xy = x^3 + 2ax + b \tag{3.3}$$

denklemlerine dönüşür. Eğer K cisminin karakteristiği $Char(K) = 3$ ise;

$$y^2 = x^3 + 2ax + bx + c \tag{3.4}$$

denklemine dönüşür. Eğer K cismi karakteristiği $Char(K) \neq 2,3$ ise ve $a = b = c = 0$ durumunda eliptik eğri denklemi:

$$E: y^2 = x^3 + ax + b \tag{3.5}$$

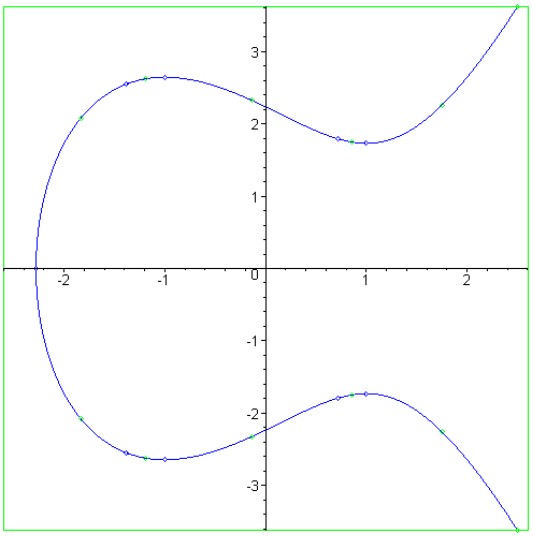
şeklindedir. Herbir denklem için Afin dönüşümler kullanılır ve bu denklemde yer alan $x, y, a, b, c, d\ ve\ e$ değerleri de K cisminin üzerinde yer almaktadır. Yukarıda belirtilen herhangi bir şartı sağlayan, K alanında yer alan ve (x, y) noktalarının kümesinden oluşan denkleme E eliptik eğri denklemi diyebiliriz [8].

Gerçek sayılar üzerinde bir eliptik eğri, karşılık gelen eğri üzerindeki noktalar ve sonsuzdaki nokta olarak adlandırılan *O* noktasının birleşiminden oluşur. Bu noktaların oluşturduğu küme, iki noktanın toplamı yine aynı eğri üzerinde bir nokta verecek şekilde tanımlanırsa bir grup oluşturulmuş olur.

Örneğin, *a*, *b*, *d* = 0, *c* = -3, *e* = 5 alınırsa, denklem $y^2 = x^3 - 3x + 5$ biçimini alır. Bu denklemin ifade ettiği eğrinin maple kodu ve çizimi aşağıda (Şekil 3.1) verilmiştir.

```
[> with(algcurves):
[> f:=y^2-x^3+3*x-5:
[> plot_real_curve(f,x,y,showArrows=false);
```

Şekil 3.1: Eliptik eğri maple kod örneği.



Şekil 3.2: Eliptik eğri örneği.

3.2. Eliptik Eğriler ve Grup Yapısı

Eliptik eğrinin derecesi *n*, eğriyi tanımlayan polinomdaki x ve y lerin derecelerinin toplamı ile belirlenir. Örneğin, $x^2y = 1$ eşitliğinin derecesi 3'tür. Homojen polinomlarda eşitliğin her iki tarafına ait derecelerin eşit olması gerekir ($x^2y = z^3$ de olduğu gibi).

Eliptik eğriler E ile sembolize edilir ve 3.dereceden bir polinom olan Weierstrass eşitliğiyle (3.1) tanımlanır. Bu denklemin çözüm kümesi nokta toplama işlemi ile bir Abelian Grup oluşturmaktadır. Oluşan bu grup kriptografik sistemde kullanılabilir [9]. Oluşturulan grubun birim elemanı y ekseni üzerinde olduğu varsayılan sonsuzdaki *O* noktasıdır.

(3.2) de verilen $a, b \in GF(p)$ değişkenleri, eliptik eğri oluşturabilmesi için $4a^3 + 27b^2 \neq 0 \; mod \; p$ koşulunu sağlamalıdır. Bu eğrinin diskriminantı ise $\Delta = -16(4a^3 + 27b^2)$ olarak tanımlanır.

Eliptik eğri üzerindeki iki noktadan geçen bir doğru, eliptiği üçüncü bir noktada keser kuralı "kiriş teğet kuralı" (chord-tangent rule) olarak anılır ve kriptografiye uygun bir cisim oluşturma için kullanılır.

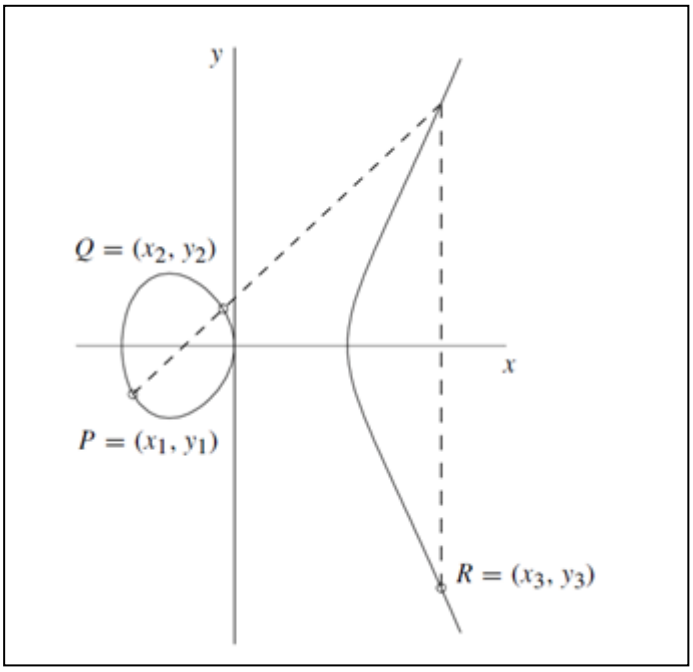
Kuralların polinom için işletilebilmesi için $\Delta(E) \neq 0$ olmalıdır. Böylece eliptik eğri tekil olmaz ve sonlu sayıda (x, y) nokta çifti ile çözümlenir. Oluşacak grup için birim eleman ve her (x, y) noktasının tersi de aynı grubun elemanı olarak bulunmalıdır.

Tekil eliptik eğrilerde, polinomu sağlayan (x, y) nokta çifti tektir. Yani grubun tek bir elemanı vardır, bu nedenle grup oluşturamaz.

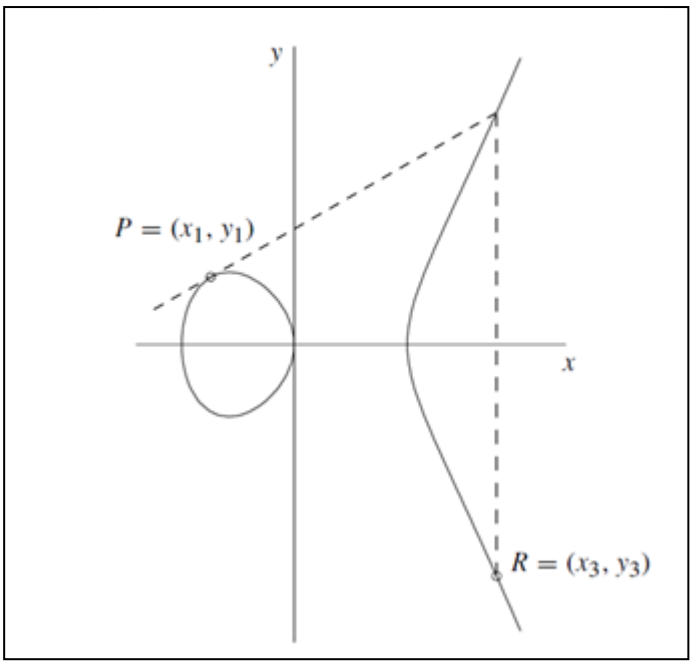
3.3. Eliptik Eğri Tabanlı Geometri

E eliptik eğrisi için tanımlanan Weierstrass denklemini sağlayan (x, y) nokta çiftlerini bulmak için eliptik eğri üzerinde nokta toplama ve nokta çarpma işlemleri gerçekleştirilir. $P(x, y)$ ve $Q(x, y)$, E eliptik eğrisi üzerinde tanımlı iki nokta olmak üzere bu noktaların toplamı aşağıda belirtilen şekilde yapılır.

P ve Q noktaları bir doğruyla birleştirilir. Bu iki noktayı birleştiren doğru eliptik eğriyi üçüncü noktada daha keser. Bu noktanın x eksenine göre simetriğinin alınmasıyla $P + Q$ değeri bulunur. $P = Q$ olduğu durumda noktaları kesen doğru P noktasından geçen eğriye teğet geçer [8].



Şekil 3.3: Geometrik nokta toplama.



Şekil 3.4: Geometrik nokta çiftleme.

Geometrik olarak verilen nokta toplama ve nokta çıkarma işlemlerini matematiksel olarak şu şekilde ifade edebiliriz:

- Eğer $P \neq Q$ ise $P + Q = R$ olur. $R(x_3, y_3)$

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \tag{3.6}$$

$$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1 \tag{3.7}$$

- Eğer $P = Q$ ise o zaman $2P = R$ olur.

$$x_3 = \left(\frac{3x_1^2 - a}{2y_1}\right)^2 - 2x_1 \tag{3.8}$$

$$y_3 = \left(\frac{3x_1^2 - a}{2y_1}\right)(x_1 - x_3) - y_1 \tag{3.9}$$

Nokta toplama ve nokta çiftleme işlemlerinde çarpma, kare alma, toplama ve ters alma işlemleri vardır. Çarpmaya göre tersini alma işlemi donanım olarak gerçekleştirmek yüksek maliyet gerektirir. Bu nedenle değişken dönüşümleri yapılarak projektif koordinatlara geçiş yapılarak maaliyet azaltılır.

3.4. Projektif Koordinatlar

Afin(doğrusal) koordinat sisteminde işlemleri yaparken ortaya çıkan işlem sayısı fazla ve ters alma işlemi gibi yüksek maliyet gerektiren adımlar olduğu için farklı bir kümeye geçerek daha az maliyetli ve daha az adımı olan işlem tanımlayabilmek için projektif koordinatlar kullanılır. Afin koordinatlarda yapılan çarpmaya göre tersini alma işlemi bir çarpma işlemine göre yaklaşık 30 kat daha fazla bir maliyete sahiptir.

Projektif (izdüşümsel) uzayda $x, y, z \in F$ olup x, y, z 'nin en az birinin sıfır olmadığı durumda bir P noktası (x, y, z) üçlüleri ile ifade edilir. Bir (x, y, z) üçlüsü için eşdeğer sınıf x, y ve z arasındaki oranlara bağlıdır. Bu nedenle (x, y, z) için eşdeğer sınıf $(x: y: z)$ ile gösterilir.

Eğer $(x: y: z)$, $z \neq 0$ olduğu bir noktaysa $(x: y: z) = ({}^{X}/_{Z} : {}^{Y}/_{Z} : 1)$ olarak ifade edilebilir. Eğer $z = 0$ ise sıfıra bölme işlemi x ve y koordinatlarının her ikisi için de ∞ değerini gösterecektir. Bu nedenle $(x: y: 0)$ noktası projektif uzayda sonsuzdaki nokta olarak tanımlanır.

Afin koordinat sisteminden projektif koordinatlara dönüşüm: $(x, y) \rightarrow (X, Y, Z) = (xZ, yZ, Z)$ şeklindedir. Projektif koordinatlardan afin koordinat sistemine dönüşüm ise $(X, Y, Z) \rightarrow (x, y) = ({}^{X}/_{Z}, {}^{Y}/_{Z})$ şeklindedir.

Projektif gösterimin biraz daha geliştirilmiş biçimi olan Jacobian ve Chudnovsky projektif gösterimleri de kullanılabilmektedir. Afin koordinat sisteminden Jacobian projektif koordinatlara dönüşüm $(x, y) \rightarrow (X, Y, Z) = (xZ^2, yZ^3, Z)$ şeklindedir. Jacobian projektif koordinatlardan afin koordinat sistemine dönüşüm ise $(X, Y, Z) \rightarrow (x, y) = ({}^{X}/_{Z^2}, {}^{Y}/_{Z^3})$ şeklindedir. Afin koordinat sisteminden Chudnovsky projektif koordinatlara dönüşüm $(x, y) \rightarrow (X, Y, Z) = (xZ^2, yZ^3, Z)$ şeklindedir. Chudnovsky projektif koordinatlardan afin koordinat sistemine dönüşüm ise $(X, Y, Z) \rightarrow (x, y) = ({}^{X}/_{Z^2}, {}^{Y}/_{Z^3})$ şeklindedir.

4. ELİPTİK EĞRİ FORMLARI

4.1. Weierstrass Eğrileri

İki boyutlu afin koordinat sisteminde F sonlu cismi üzerinde karakteristiği 2 ve 3 den farklı olan bir E eliptik eğrisi aşağıdaki gibi tanımlanır;

$$E: y^2 = x^3 + ax + b \tag{4.1}$$

$a, b \in F$ ve diskriminant $\Delta \neq 0$ olması durumunda polinomun birden fazla (x, y) çözüm kümesi olan eliptik eğriyi tanımlar. Çözüm kümesi içerisindeki tüm (x, y) çiftleri, F sonlu cismi üzerinde kiriş ve teğet kurallarına uygun bir şekilde nokta toplama işlemleri yapılarak elde edilir.

F_p cismi asal bir cisim ve $a, b \in F_p$ olup, $\Delta = -16(4a^3 + 27b^2) \neq 0$ olması durumunda eliptik eğriyi denklemini sağlayan (x, y) çiftleri, nokta toplama ve nokta çiftleme işlemlerinin ardışık olarak sonsuzdaki noktaya ulaşıncaya kadar tekrarlanması sonucu elde edilir.

Bu işlemler yapılırken öncelikle bir $P(x_1, y_1)$ noktası seçilir. Ardından sırasıyla nokta çiftleme ve nokta toplama işlemleri yapılır.

$$2P = P_1 + P_1 \tag{4.2}$$

$$3P = 2P + P_1 \tag{4.3}$$

$x = {}^{X}/_{Z}$, $y = {}^{Y}/_{Z}$ ve $Z \neq 0$ olmak koşuluyla Weierstrass eşitliğini projektif koordinatlara dönüştürdüğümüzde aşağıdaki eşitlik elde edilir.

$$E: Y^2Z = X^3 + aXZ^2 + bZ^3 \tag{4.4}$$

$Z = 0$ olması durumunda nokta sonsuzdaki bir nokta (∞) olarak adlandırılır.

Projektif koordinatlarda tanımlanmış olan eğri üzerinde seçilen iki noktanın toplama ve çiftleme işlemleri cebirsel geometri kullanarak elde edilir. Seçilen noktalar $P(x_1, y_1)$, $Q(x_2, y_2)$ olsun, $P, Q \in E$ ve , $Q \neq \infty$, $Q \neq -P$ olmak üzere

geometrik nokta toplama ve nokta çiftleme işlemleri Şekil 3.3 ve Şekil 3.4 ‘de gösterilmiştir.

(3.6), (3.7), (3.8) ve (3.9) nolu eşitliklerde ifade edilen, afin koordinat sisteminde yapılan nokta toplama ve nokta çiftleme işlemleri, ters alma (inversion) işlevi içerdiğinden, projektif koordinat sistemine geçiş yapılarak aritmetik işlemler yapılır [11].

- Projektif koordinatlarda nokta toplama işlemi:

Aşağıda Şekil 4.1’de verilen algoritma ile projektif koordinatlarda afin koordinatlara göre daha hızlı nokta toplama işlemi gerçekleştirilebilmektedir. İşlem maliyeti 12M + 2S + 7add + 1times2 + 1times3 + 1Inv2 şeklindedir. (M: Çarpma, S: Kare alma, add: Toplama-çıkarma, times2: 2 ile çarpma, times3: 3 ile çarpma, Inv: Tersini alma işlemi, D: Sabit sayı ile çarpma işlemi)

$$U_1 = X_1 Z_2;$$
$$U_2 = X_2 Z_1;$$
$$S_1 = Y_1 Z_2;$$
$$S_2 = Y_2 Z_1;$$
$$W_1 = Z_1 Z_2;$$
$$W_2 = U_1 + U_2;$$
$$P_1 = U_2 - U_1;$$
$$P_2 = P_1^2;$$
$$R_1 = S_2 - S_1;$$
$$R_2 = R_1^2;$$
$$X_3 = P_1(W_1 R_2 - W_2 P_2);$$
$$Y_3 = (R_1 * (-2W_1 R_2 + 3W_2 P_2) - P_1 P_2 (S_1 + S_2))/2;$$
$$Z_3 = W_1 P_1 P_2;$$

Şekil 4.1: Projektif koordinatlarda nokta toplama algoritması.

- Projektif koordinatlarda nokta çiftleme işlemi:

Aşağıda Şekil 4.2'de verilen algoritma ile projektif koordinatlarda afin koordinatlara göre daha hızlı nokta çiftleme işlemi gerçekleştirilebilmektedir. İşlem maliyeti 5M + 6S + 1D + 7add + 3times2 + 1times3 şeklindedir.

$U_1 = \mathrm{X}_1^2;$
$W_1 = Z_1^2;$
$S_1 = aW_1 + 3U_1;$
$S_2 = 2Y_1Z_1;$
$P_1 = S_2^2;$
$P_2 = S_2P_1;$
$R_1 = Y_1S_2;$
$R_2 = R_1^2;$
$W_2 = (X_1 + R_1)^2 - U_1 - R_2;$
$U_2 = S_1^2 - 2W_2;$
$X_3 = U_2S_2;$
$Y_3 = S_1(W_2 - U_2) - 2R_2;$
$Z_3 = P_2;$

Şekil 4.2: Projektif koordinatlarda nokta çiftleme algoritması.

4.2. Jacobian Eğrileri

Afin koordinat sisteminde tanımlı olan Weierstrass denkleminde $x = {}^{X}/_{Z^2}$, $y = {}^{Y}/_{Z^3}$ konduğunda $Z \neq 0$ olmak şartı ile aşağıdaki Jacobian eşitliğini elde ederiz [8].

$$E_j: Y^2 = X^3 + aXZ^4 + bZ^6 \tag{4.5}$$

Jacobian koordinatlarında nokta çiftleme işlemi projektif koordinatlara göre daha hızlı, toplama işlemi ise daha yavaştır.

- Jacobian eğrilerde nokta toplama işlemi:

$P(X_1, Y_1, Z_1)$ ve $Q(X_2, Y_2, Z_2)$ noktaları bu eğri üzerinde tanımlı iki nokta olmak üzere, $P + Q = R = (X_3, Y_3, Z_3)$ aşağıdaki eşitliklerle elde edilir ($P \neq \pm Q$). İşlem maliyeti: 8M + 3S + 6add + 1times2.

$U_1 = X_1 Z_2^2;$
$U_2 = X_2 Z_1^2;$
$S_1 = Y_1 Z_2^3;$
$S_2 = Y_2 Z_1^3;$
$H = U_2 - U_1;$
$r = S_2 - S_1;$
$X_3 = -H^3 - 2U_1 H^2 + r^2;$
$Y_3 = -S_1 H^3 + r(U_1 H^2 - X_3);$
$Z_3 = Z_1 Z_2 H;$

Şekil 4.3: Jacobian koordinatlarda nokta toplama eşitlikleri.

1. $\mathrm{T}_1 = \mathrm{Z}_1^2;$
2. $T_2 = T_1 Z_1;$
3. $T_1 = T_1 X_2;$
4. $T_2 = T_2 Y_2;$
5. $T_1 = T_1 - X_1;$
6. $T_2 = T_2 - Y_1;$
7. $Z_3 = Z_1 T_1;$
8. $T_3 = T_1^2;$
9. $T_4 = T_3 T_1;$
10. $T_3 = T_3 X_1;$
11. $T_1 = 2T_3;$
12. $X_3 = T_2^2;$
13. $X_3 = X_3 - T_1;$
14. $X_3 = X_3 - T_4;$
15. $T_3 = T_3 - X_3;$
16. $T_3 = T_3 T_2;$
17. $T_4 = T_4 Y_1;$
18. $Y_3 = T_3 - T_4$

Şekil 4.4: Jacobian koordinatlar için nokta toplama algoritması.

- Jacobian eğrilerde nokta çiftleme işlemi:

$P(X_1, Y_1, Z_1)$ noktası eğri üzerinde tanımlı bir nokta olmak üzere, $2P = R$ aşağıdaki eşitlikle elde edilir. İşlem maliyeti: 5M + 4S + 5add + 2times2 + 1times3.

$$S = 4X_1Y_1^2;$$
$$M = 3X_1^2 + aZ_1^4;$$
$$T = -2S + M^2;$$
$$X_3 = T;$$
$$Y_3 = -8Y_1^4 + M(S - T);$$
$$Z_3 = 2Y_1Z_1;$$

Şekil 4.5: Jacobian koordinatlarda nokta çiftleme eşitlikleri.

1. $T_1 = Z_1^2;$	10. $Y_3 = Y_3^2;$
2. $T_2 = X_1 - T_1;$	11. $Y_3 = Y_3/2;$
3. $T_1 = X_1 + T_1;$	12. $X_3 = T_2^2;$
4. $T_2 = T_2T_1;$	13. $T_1 = 2T_3;$
5. $T_2 = 3T_2;$	14. $X_3 = X_3 - T_1;$
6. $Y_3 = 2Y_1;$	15. $T_1 = T_3 - X_3;$
7. $Z_3 = Y_3Z_1;$	16. $T_1 = T_1T_2;$
8. $Y_3 = Y_3^2;$	17. $Y_3 = T_1 - Y_3;$
9. $T_3 = Y_3X_1;$	

Şekil 4.6: Jacobian koordinatlar için nokta çiftleme algoritması.

4.3. Chudnovsky Jacobian

Jacobian nokta toplama işleminin projektif koordinatlara gore daha hızlı yapılabilmesi için Chudnovsky Jacobian koordinatları kullanılır. Bu durumda bir Jacobian noktası beşli gösterim (quintuple)(X, Y, Z, Z^2, Z^3) şeklinde ifade edilir [8].

- Chudnovsky Jacobian nokta toplama işlemi:

$P(X_1, Y_1, Z_1, Z_1^2, Z_1^3)$, $Q(X_2, Y_2, Z_2, Z_2^2, Z_2^3)$ olmak üzere $P + Q = R$ şu şekilde bulunur. İşlem maliyeti: 13M + 4S + 6add.

$U_1 = X_1(Z_2^2);$
$U_2 = X_2(Z_1^2);$
$S_1 = Y_1(Z_2^3);$
$S_2 = Y_2(Z_1^3);$
$H = U_2 - U_1;$
$r = S_2 - S_1;$
$X_3 = -(U_1 + U_2)H^2 + r^2;$
$Y_3 = -S_1H^3 + r(U_1H^2 - X_3);$
$Z_3 = Z_1Z_2H;$
$Z_3^2 = Z_3^2;$
$Z_3^3 = Z_3^3;$

Şekil 4.7: Chudnovsky Jacobian koordinatlar için nokta toplama algoritması.

- Chudnovsky Jacobian nokta çiftleme işlemi:

$P(X_1, Y_1, Z_1, Z_1^2, Z_1^3)$, $Q(X_2, Y_2, Z_2, Z_2^2, Z_2^3)$ olmak üzere $2P = R$ şu şekilde bulunur. İşlem maliyeti: 3M + 6S + 1D + 4add + 7times2 + 1times3.

$S = 4X_1Y_1^2;$
$M = 3X_1^2 + a(Z_1^2)^2;$
$T = -2S + M^2;$
$X_3 = T;$
$Y_3 = -8Y_1^4 + M(S - T);$
$Z_3 = 2Y_1Z_1;$
$Z_3^2 = Z_3^2; Z_3^3 = Z_3^3;$

Şekil 4.8: Chudnovsky Jacobian koordinatlar için nokta çiftleme algoritması.

4.4. Jacobi Quartics

Jacobi quartic eliptik eğrileri sonlu F alanı içerisinde $char\,(F) \neq 2,3$ ve $k \neq 0, \pm 1$ olmak koşulu ile şu şekilde ifade edilir:

$$E_{J,k}: y^2 = k^2x^4 - (k^2+1)x^2 + 1 \\ = (1-x^2)(1-k^2x^2) \tag{4.6}$$

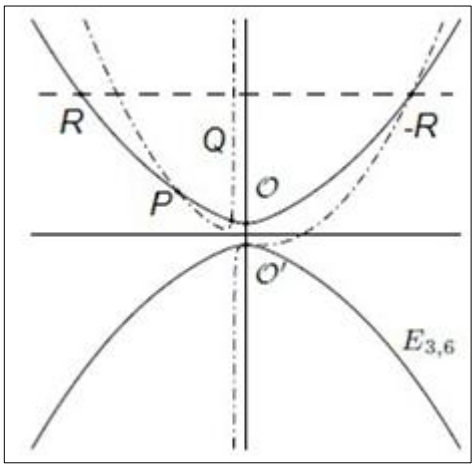
Bu tanımlanan eğri üzerinde H. Hışıl [16] bir düzenleme yaparak aşağıdaki formu elde etmiştir. ($a, d \in F, a^2 \neq 1$)

$$E_{J,k,a}: y^2 = dx^4 + 2ax^2 + 1 \tag{4.7}$$

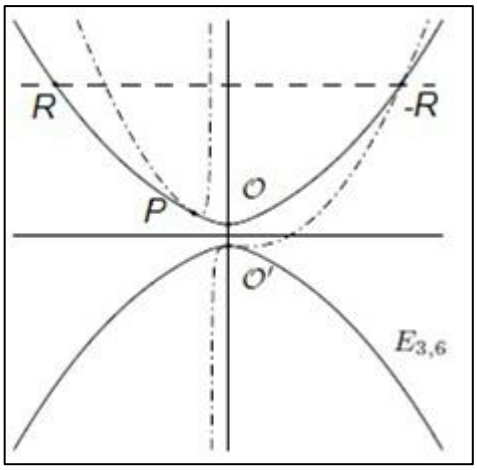
- Jacobian koordinatlarda $x = {}^{X}/_{Z}, y = {}^{Y}/_{Z^2}$ ile dönüşüm yaptığımızda:

$$Y^2 = dX^4 - 2aX^2Z^2 + Z^4 \tag{4.8}$$

Bu eğri formu üzerinde yapılan aritmetik işlemler Jacobi Intersection ile karşılaştırıldığında daha hızlı olduğu görülür [11].



Şekil 4.9: Jacobi quartics eğrisi nokta toplama geometrik gösterimi.



Şekil 4.10: Jacobi quartics eğrisi nokta çiftleme geometrik gösterimi.

- Jacobi quartics nokta toplama işlemi:

Önceki bölümde Şekil 4.9'da bahsedilen Jacobian Quartics eğrisi ile elde edilen aritmetik nokta toplama eşitlikleri ve bu eşitliklerin çözümü için kullanılan algoritma [15] aşağıda Şekil 4.12'de belirtilmiştir. Algoritma işlem maliyeti: 10M + 3S + 1D + 13add + 1times2.

$$P + Q = R = (X_3 : Y_3 : Z_3)$$
$$Z_1 \neq 0,\ Z_2 \neq 0,\ \ P \neq Q$$

$$X_3 = X_1 Y_2 Z_1 - X_2 Y_1 Z_2$$
$$Y_3 = [(Z_1 Z_2)^2 + d(X_1 X_2)^2](Y_1 Y_2 - 2aX_1 X_2 Z_1 Z_2) + 2dX_1 X_2 Z_1 Z_2 (X_1^2 Z_2^2 + Z_1^2 X_2^2)$$
$$Z_3 = (Z_1 Z_2)^2 - d(X_1 X_2)^2$$

Şekil 4.11: Jacobi quartics nokta toplama eşitlikleri.

1. $T_1 = X_1;$	13. $T_5 = T_1T_4;$	27. $T_4 = aT_6;$
2. $T_2 = Y_1;$	14. $T_1 = T_1 + T_3;$	28. $T_2 = T_2 + T_4;$
3. $T_3 = Z_1;$	15. $T_8 = T_3T_6;$	29. $T_4 = T_8^2;$
4. $T_4 = X_2;$	16. $T_4 = T_4 + T_6;$	30. $T_8 = T_5^2;$
5. $T_5 = Y_2;$	17. $T_6 = T_5T_8;$	31. $T_5 = T_4 + T_8;$
6. $T_6 = Z_2;$	18. $T_7 = T_7 - T_6$	32. $T_2 = T_2T_5;$
7. $T_7 = T_1T_3;$	20. $T_1 = T_1T_4;$	33. $T_2 = T_2 + T_3;$
8. $T_7 = T_2 + T_7;$	21. $T_1 = T_1 - T_5;$	34. $T_5 = T_4 - T_8;$
9. $T_8 = T_4T_6;$	22. $T_1 = T_1 - T_8;$	35. $X_3 = T_7;$
10. $T_8 = T_5 + T_8;$	23. $T_3 = T_1^2;$	36. $Y_3 = T_2;$
11. $T_2 = T_2T_5;$	24. $T_6 = 2T_6;$	37. $Z_3 = T_5;$
12. $T_7 = T_7T_8;$	25. $T_3 = T_3 - T_6;$	
13. $T_7 = T_7 - T_2;$	26. $T_3 = T_3T_6;$	

Şekil 4.12: Jacobi quartics nokta toplama algoritması.

- Jacobi quartics nokta çiftleme işlemi:

Şekil 4.11'de gösterilen eşitliklerde P = Q ifadesi yerine konursa, nokta Şekil 4.10'da gösterilen Jacobian Quartic eğrisi için nokta çiftleme eşitlikleri elde edilir. Bu eşitliklerin çözümü için ise kullanılan algoritma [17] aşağıda Şekil 4.13'de gösterilmiştir. Algoritma işlem maliyeti: 1M + 9S + 1D + 10add + 4times2.

$2P = R = (X_3 : Y_3 : Z_3)$

1. $T_1 = X_1^2;$	7. $T_7 = (X_1 + Z_1)^2 - T_6;$
2. $T_2 = T_1^2;$	8. $T_8 = T_7^2;$
3. $T_3 = Y_1^2;$	9. $X_3 = (Y_1 + T_7)^2 - T_3 - T_8;$
4. $T_4 = Z_1^2;$	10. $Y_3 = (T_5 + T_2)(4T_3 + 2aT_8) + T_8^2;$
5. $T_5 = T_4^2;$	11. $Z_3 = 2(T_5 - T_2);$
6. $T_6 = T_1 + T_4;$	

Şekil 4.13: Jacobi quartics nokta çiftleme algoritması.

4.5. Extended Jacobi Quartics

Karakteristiği 2'den farklı $char(F) \neq 2$ olan bir F alanında Extended Jacobi quartic eliptik eğri formu, $a, d \in F \; ve \; \Delta = 256d(a^2 - d)^2 \neq 0$ durumunda şu şekilde ifade edilir:

$$E_{J,d,a}: y^2 = dx^4 + 2ax^2 + 1 \tag{4.9}$$

- Jacobian koordinatlarda $x = {}^{X}/_{Z}, y = {}^{Y}/_{Z^2}$ ile dönüşüm yaptığımızda:

$$Y^2 = dX^4 - 2aX^2Z^2 + Z^4 \tag{4.10}$$

ifadesi elde edilir.

- Extended projektif koordinatlarda ise aşağıdaki şekilde tanımlanır:

$$X^2 - TZ = 0 \tag{4.11}$$

$$Y^2 - dT^2 - 2aX^2 - Z^2 = 0 \tag{4.12}$$

- Yukarıdaki ifadeyi $T = {}^{X^2}/_{Z}$ olmak şartı ile basitçe şu şekilde gösterebiliriz:

$$Y^2Z^2 = dX^4 + 2aX^2Z^2 + Z^4 \tag{4.13}$$

Bu durumda extended projektif koordinatlarda extended Jacobi Quartic eğrisi üzerinde P ve Q noktası $P = (X_1 : Y_1 : T_1 : Z_1)$, $Q = (X_2 : Y_2 : T_2 : Z_2)$ şeklinde ifade edilir [11].

- Extended Jacobi quartics nokta toplama işlemi:

(4.13) nolu eşitlik için P + Q = R ifadesi aşağıda Şekil 4.14'de gösterilen denklemlerle bulunur.

$$P + Q = R = (X_3 : Y_3 : T_3 : Z_3), \qquad Z_1 \neq 0,\ Z_2 \neq 0,\ \ P \neq Q$$
$$X_3 = (X_1Y_2 - Y_1X_2)(T_1Z_2 - Z_1T_2)$$
$$Y_3 = (T_1Z_2 + Z_1T_2 - 2X_1X_2)(Y_1Y_2 - 2aX_1X_2 + Z_1Z_2 + dT_1T_2) - Z_3$$
$$T_3 = (T_1Z_2 - Z_1T_2)^2 \qquad\qquad Z_3 = (X_1Y_2 - Y_1X_2)^2$$

Şekil 4.14: Extended Jacobi quartics nokta toplama eşitlikleri.

Şekil 4.14'de P ve Q noktalarının toplamı sonucu elde edilen R noktasına ait parametreler aşağıdaki algoritma ve 7M + 3S + 2D + 17add + 1times2 + 1Inv2 işlem maliyeti ile elde edilebilmektedir [16]:

1. $t_1 = T_1 + Z_1;$	11. $T_3 = T_3 - t_2;$	21. $Y_3 = Y_3 + t_2;$
2. $t_2 = dT_2;$	12. $t_2 = X_2 + Y_2;$	22. $Y_3 = t_1Y_3;$
3. $t_2 = t_2 + Z_2;$	13. $Z_3 = Z_3t_2;$	23. $X_3 = Z_3 + T_3;$
4. $t_2 = t_1t_2;$	14. $t_2 = X_1X_2;$	24. $X_3 = X_3^2;$
5. $t_2 = Z_1T_2;$	15. $Y_3 = Y_1Y_2;$	25. $Z_3 = Z_3^2;$
6. $T_3 = T_1Z_2;$	16. $Z_3 = Z_3 - t_2;$	26. $Y_3 = Y_3 - Z_3;$
7. $t_1 = t_1 - T_3;$	17. $Z_3 = Z_3 + Y_3;$	27. $X_3 = X_3 - Z_3;$
8. $t_3 = dt_2;$	18. $Y_3 = Y_3 + t_1;$	28. $T_3 = T_3^2;$
9. $t_1 = t_1 - t_3;$	19. $t_1 = 2t_2;$	29. $X_3 = X_3 - T_3;$
10. $t_3 = T_3 + t_2;$	20. $t_1 = t_3 - t_1;$	30. $X_3 = X_3/2$

Şekil 4.15: Extended Jacobi quartics nokta toplama algoritması.

- Extended Jacobi quartics nokta çiftleme işlemi:

(4.13) nolu eşitlik için $2P = R = (X_3 : Y_3 : T_3 : Z_3)$ ifadesi aşağıda gösterilen denklemlerle bulunur.

$$X_3 = X_1Y_2Z_1 + X_2Y_1Z_2$$
$$Y_3 = [(Z_1Z_2)^2 + d(X_1X_2)^2](Y_1Y_2 - 2aX_1X_2Z_1Z_2) + 2dX_1X_2Z_1Z_2(X_1^2Z_2^2 + Z_1^2X_2^2)$$
$$T_3 = (2X_1Y_1)^2$$
$$Z_3 = (Z_1Z_2)^2 - d(X_1X_2)^2$$

Şekil 4.16: Extended Jacobi quartics nokta çiftleme eşitlikleri.

Şekil 4.16'da P noktasının çiftlemesi ile elde edilen R noktasına ait parametreler aşağıdaki algoritma ve 8M + 9add + 4times2 işlem maliyeti ile elde edilebilmektedir [11]:

1. $T_3 = X_1 + Y_1;$
2. $X_3 = X_1^2;$
3. $Y_3 = Y_1^2;$
4. $Z_3 = Z_1^2;$
5. $T_3 = T_3^2;$
6. $X_3 = X_3 + Y_3;$
7. $T_3 = T_3 - X_3;$
8. $Z_3 = 2Z_3;$
9. $Z_3 = Z_3 - X_3;$
10. $X_3 = T_3 + Z_3;$
11. $T_3 = T_3^2;$
12. $Z_3 = Z_3^2;$
13. $X_3 = X_3^2;$
14. $X_3 = X_3 - T_3;$
15. $X_3 = X_3 - Z_3;$
16. $Z_3 = 2Z_3;$
17. $Y_3 = 2Y_3;$
18. $Y_3 = Y_3^2;$
19. $Y_3 = Y_3 + T_3;$
20. $Y_3 = Y_3 - Z_3;$
21. $T_3 = 2T_3;$

Şekil 4.17: Extended Jacobi quartics nokta çiftleme algoritması.

4.6. Jacobi Intersection

Liadert ve Smart [17], karakteristiği 2'den farklı olan sonlu F alanında Jacobian eğrisini $(E_{J,b})$ şu şekilde tanımlamışlardır:

$$x^2 + y^2 = 1 \tag{4.14}$$

$$bx^2 + t^2 = 1 \tag{4.15}$$

$Z \neq 0, b(1-b) \neq 0$ olmak koşulu ile projektif koordinatlarda ifade edecek olursak ($x = {}^{X}/_{Z}, y = {}^{Y}/_{Z}, t = {}^{T}/_{Z}$).

$$X^2 + Y^2 = Z^2 \tag{4.16}$$

$$bX^2 + T^2 = Z^2 \tag{4.17}$$

- Jacobi intersection nokta toplama işlemi:

$E_{J,b}$ eğrisi üzerinde bir nokta $P = (X_1 : Y_1 : T_1 : Z_1)$ şeklinde dörtlü gösterimle ifade edilir. $P + Q = R = (X_3 : Y_3 : T_3 : Z_3)$, $Z_1 \neq 0$, $Z_2 \neq 0$, $P \neq Q$ olmak üzere R noktası aşağıdaki şekilde elde edilir. Bu durumda işlem maliyeti 13M + 2S + 2D + 13add şeklinde olur.

$A = X_1 Y_1;$
$B = T_1 Z_1;$
$C = X_2 Y_2;$
$D = T_2 Z_2;$
$E = X_1 T_2;$
$F = Y_1 Z_2;$
$G = T_1 X_2;$
$H = Z_1 Y_2;$
$J = AD;$
$K = BC;$
$X_3 = (H + F)(E + G) - J - K;$
$Y_3 = (H + E)(F - G) - J + K;$
$T_3 = (B - bA)(C + D) + bJ - K;$
$Z_3 = H^2 + G^2;$

Şekil 4.18: Jacobi intersection nokta toplama algoritması.

- Jacobi intersection nokta çiftleme işlemi:

$E_{J,b}$ eğrisi üzerinde nokta çiftleme işlemi $2P = R = (X_3 : Y_3 : T_3 : Z_3)$ şeklinde ifade edildiğinde (4.16) ve (4.17)'de belirtilen eşitliklerin çözümü aşağıdaki gibi ifade edilir. İşlem maliyeti: 3M + 4S + 1D + 7add + 1times2.

$$X_3 = 2X_1Y_1T_1Z_1;$$
$$Y_3 = -T_1^2Z_1^2 - bX_1^2Y_1^2 + 2(X_1^2Y_1^2 + Y_1^4);$$
$$T_3 = T_1^2Z_1^2 - bX_1^2Y_1^2;$$
$$Z_3 = T_1^2Z_1^2 + bX_1^2Y_1^2;$$

Şekil 4.19: Jacobi intersection nokta çiftleme eşitlikleri.

$$A = X_1Y_1;$$
$$B = T_1Z_1;$$
$$C = A^2;$$
$$D = B^2;$$
$$E = Y_1^2;$$
$$X_3 = 2AB;$$
$$Y_3 = -B - bC + 2(C + Y_1^4);$$
$$T_3 = D - bC;$$
$$Z_3 = D + bC;$$

Şekil 4.20: Jacobi intersection nokta çiftleme algoritması.

4.7. Twisted Jacobi Intersections

Jacobi intersection'ın özel bir durumu olan twisted Jacobi intersection, Feng [18] tarafından tanımlanmıştır. Karakteristiği 2'den farklı olan bir F alanı üzerinde $a, b \in F$ ve $ab(a - b) \neq 0$ olmak üzere twisted Jacobi Intersection eğrisi($E_{J,a,b}$) şu şekilde tanımlanır:

$$ax^2 + y^2 = 1 \tag{4.18}$$

$$bx^2 + t^2 = 1 \tag{4.19}$$

Projektif koordinatlarda ise $x = {}^{X}/_{Z}, y = {}^{Y}/_{Z}, t = {}^{T}/_{Z}$ ve $Z \neq 0$, $ab(a-b) \neq 0$ olmak şartı ile:

$$aX^2 + Y^2 = Z^2 \tag{4.20}$$

$$bX^2 + T^2 = Z^2 \tag{4.21}$$

olarak tanımlanır [17].

- Twisted Jacobi intersection nokta toplama işlemi:

Projektif koordinatlarda $E_{J,a,b}$ eğrisi üzerinde iki nokta $P = (X_1, Y_1, T_1, Z_1)$ ve Q= (X_2, Y_2, T_2, Z_2) olsun, $P + Q = R = (X_3 : Y_3 : T_3 : Z_3)$, $Z_1 \neq 0$, $Z_2 \neq 0$, $P \neq Q$ olması durumunda twisted intersection nokta toplamı aşağıdaki eşitlik ve algoritma ile bulunabilir.

$$\begin{aligned} X_3 &= X_1 Z_1 Y_2 T_2 + Y_1 T_1 X_2 Z_2 \\ Y_3 &= Y_1 Z_1 Y_2 Z_2 - a X_1 T_1 X_2 T_2 \\ T_3 &= T_1 Z_1 T_2 Z_2 - b X_1 Y_1 X_2 Y_2 \\ Z_3 &= Z_1^2 Y_2^2 + a X_2^2 T_1^2 \end{aligned}$$

Şekil 4.21: Twisted Jacobi intersection nokta toplama eşitlikleri.

Aşağıdaki işlem maliyeti, 13M + 2S + 5D + 13add olan algoritma ile P + Q = R noktası elde edilebilir:

1. $A = X_1Y_1;$
2. $B = T_1Z_1;$
3. $C = X_2Y_2;$
4. $D = T_2Z_2;$
5. $E = X_1T_2;$
6. $F = Y_1Z_2;$
7. $G = T_1X_2;$
8. $H = Z_1Y_2;$
9. $J = AD;$
10. $K = BC;$
11. $X_3 = (H + F)(E + G) - J - K;$
12. $Y_3 = (H + E)(F - aG) - J + aK;$
13. $Z_3 = H^2 + aG^2;$

Şekil 4.22: Twisted Jacobi intersection nokta toplama algoritması.

- Twisted Jacobi intersection nokta çiftleme işlemi:

Aşağıdaki işlem maliyeti, 3M + 4S + 1D + 7add + 1times2 olan algoritma ile $2P = R = (X_3 : Y_3 : T_3 : Z_3)$ noktası elde edilebilir:

$A = Y_1Z_1;$
$B = A^2;$
$C = X_1T_1;$
$D = C^2;$
$E = 2(T_1Z_1)^2;$
$X_3 = (A + C)^2 - B - D;$
$Y_3 = B - aD;$
$T_3 = E - B - aD;$
$Z_3 = B + aD;$

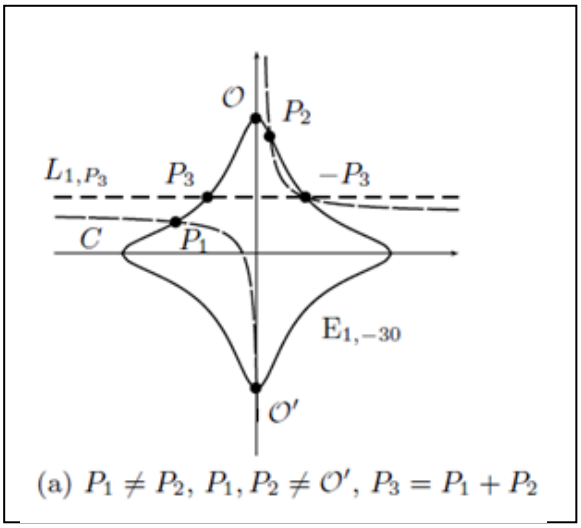
Şekil 4.23: Twisted Jacobi intersection nokta çiftleme algoritması.

4.8. Erdwards Curves

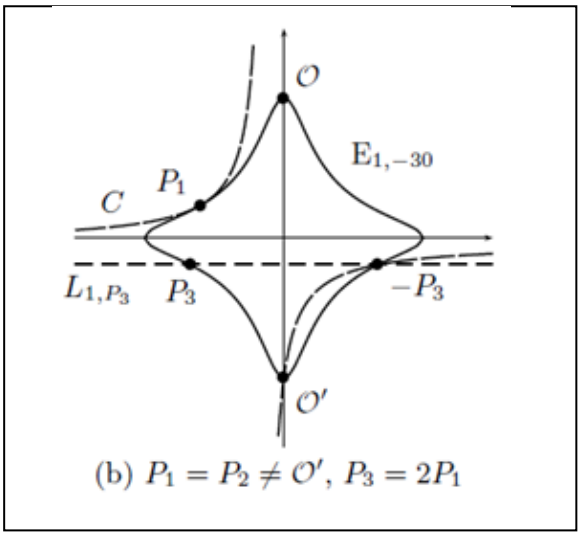
Harold Edwards [19] tarafından 2007 yılında tanımlanan eliptik eğri formudur. Karakteristiği 2'den farklı olan cisimler üzerinde tanımlanan eğri denklemi $c \in F$;

$$E_c : x^2 + y^2 = c^2(1 + x^2y^2) \tag{4.22}$$

şeklindedir.



Şekil 4.24: Edwards eğrisi nokta toplama geometrik gösterimi.



Şekil 4.25: Edwards eğrisi nokta çiftleme geometrik gösterimi.

• Edwars eğrilerde nokta toplama işlemi:

E_c eğrisi üzerince tanımlı $P(x_1, y_1)$, $Q(x_2, y_2)$ noktaları olsun, $P + Q = R$ (x_3, y_3) için aşağıdaki formül tanımlanmıştır:

$$x_3 = \frac{x_1y_2 + x_2y_1}{c(1 + x_1x_2y_1y_2)}, y_3 = \frac{y_1y_2 - x_1x_2}{c(1 - x_1x_2y_1y_2)} \tag{4.23}$$

Denklem içerisinde tersine çevirme (inverse) işlemleri yer almaktadır. Bu yüzden işlem maliyetini azaltabilmek için projektif koordinatlara dönüşüm yaparak, nokta toplama ve nokta çiftleme işlemleri gerçekleştirilir [11]. $x = {}^{X}/_{Z}$, $y = {}^{Y}/_{Z}$ $ve\ Z \neq 0$ dönüşüleri yapıldığında:

$$(X^2 + Y^2)Z^2 = c^2(Z^4 + dX^2Y^2) \tag{4.24}$$

• (4.24) nolu eşitlik için nokta toplama formülü aşağıdaki gibi elde edilir, $P(X_1 : Y_1 : Z_1)$ ve $Q(X_2 : Y_2 : Z_2)$ olmak üzere $P + Q = R(X_3 : Y_3 : Z_3)$;

$$X_3 = Z_1Z_2(Z_1^2Z_2^2 - dX_1X_2Y_1Y_2)[(X_1 + Y_1)(X_2 + Y_2) - X_1X_2 - Y_1Y_2]$$
$$Y_3 = Z_1Z_2(Z_1^2Z_2^2 + dX_1X_2Y_1Y_2)(Y_1Y_2 - X_1X_2)$$
$$Z_3 = (Z_1^2Z_2^2 - dX_1X_2Y_1Y_2)(Z_1^2Z_2^2 + dX_1X_2Y_1Y_2)$$

Şekil 4.26: Edwards eğrilerde nokta toplama eşitlikleri.

• $c = 1$ varsayılarak aşağıda tanımlı işlemleri kullanarak sonuçları 10M + 1S+ 1D +7add maaliyetle R noktasını elde edebiliriz;

• Afin koordinat dönüşümü: $x3 = X_3/Z_3,\ y3 = Y_3/Z_3$

1. $A = Z_1Z_2;$
2. $B = A^2;$
3. $C = X_1X_2;$
4. $D = Y_1Y_2;$
5. $E = dCD;$
6. $F = B - E;$
7. $G = B + E;$
8. $X_3 = AF\big((X_1 + Y_1)(X_2 + Y_2) - C - D\big);$
9. $Y_3 = AG(D - C);$
10. $Z_3 = cFG;$

Şekil 4.27: Edwards eğrileri için nokta toplama algoritması.

- Edwards eğrilerde nokta çiftleme işlemi:

Nokta toplama işlemi yapılırken, iki aynı nokta kullanıldığında nokta çiftleme formülü elde edilmiş olur.

$$x_3 = \frac{2cx_1y_1}{x_1^2 + y_1^2}, y_3 = \frac{c(y_1^2 - x_1^2)}{2c^2 - (x_1^2 + y_1^2)} \tag{4.25}$$

- Benzer şekilde projektif koordinatlara dönüşüm yapıldığında,

$$X_3 = c[(X_1 + Y_1)^2 - X_1^2 - Y_1^2][(X_1^2 + Y_1^2) - 2c^2Z_1^2]$$
$$Y_3 = c(X_1^2 + Y_1^2)(X_1^2 - Y_1^2)$$
$$Z_3 = (X_1^2 + Y_1^2)[(X_1^2 + Y_1^2) - 2c^2Z_1^2]$$

Şekil 4.28: Edwards eğrilerde nokta çiftleme eşitlikleri.

Aşağıda tanımlı işlemleri kullanarak sonuçları 3M + 4S + 3D + 5a + 1times2 maaliyetle elde edebiliriz. Afin koordinat dönüşümü: $x3 = X_3/Z_3,\ y3 = Y_3/Z_3$

$$B = ((X_1 + Y_1))^2;$$
$$C = X_1^2;$$
$$D = Y_1^2;$$
$$E = C + D;$$
$$H = (cZ_1)^2;$$
$$J = E - 2H;$$
$$X_3 = c(B - E)J;$$
$$Y_3 = cE(C - D);$$
$$Z_3 = EJ$$

Şekil 4.29: Edwards eğriler için nokta çiftleme algoritması.

4.9. Twisted Edwards Eğrileri

Twisted Edwards eğrileri karakteristiği 2'den farklı olan sonlu bir F cismi üzerinde $a, d \epsilon F$ olmak üzere

$$ax^2 + y^2 = 1 + dx^2y^2 \tag{4.26}$$

şeklinde ifade edilir. Twisted Edwards eğrileri $E_{E,a,d}$ şeklinde ifade edilir. $a = 1$ olması durumunda bütün twisted Edwards eğrileri bir Erdwards eğrisini ifade eder. $P(x_1, y_1)$ ve $Q(x_2, y_2)$ olarak verilen iki noktanın toplamı $P + Q = R(x_3, y_3)$ şeklinde gösterilir ve aşağıdaki eşitlikler elde edilir.

$$x_3 = \frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, y_3 = \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2} \tag{4.27}$$

P noktası twisted Edwards koordinatlarda $P(X_1 : Y_1 : Z_1)$ şeklinde ifade edilir ve $x_1 = {}^{X_1}/_{Z_1}$, $y_1 = {}^{Y_1}/_{Z_1}$, $Z_1 \neq 0$' dır. Bu durumda eğri denklemi:

$$a\left(\frac{X}{Z}\right)^2 + \left(\frac{Y}{Z}\right)^2 = 1 + d\left(\frac{X}{Z}\right)^2\left(\frac{Y}{Z}\right)^2 \tag{4.28}$$

olur ve projektif koordinatlarda twisted Edwards eğrisi aşağıdaki formda tanımlanır [20]:

$$(aX^2 + Y^2)Z^2 = Z^4 + dX^2Y^2 \tag{4.29}$$

- Twisted Edwards eğrilerde nokta toplama işlemi:

Nokta toplama formülünü elde edebilmek için $P(x_1, y_1)$ ve $Q(x_2, y_2)$ noktalarını projektif koordinatlardaki eşiti olan nokta formlarını twisted Edwards eğri koordinatlarında yerine konulup eşitlikler sadeleştirildiğinde, aşağıdaki sonuçlar elde edilir.

$$X_3 = Z_1Z_2(Z_1^2Z_2^2 - dX_1X_2Y_1Y_2)(X_1Y_2 + X_2Y_1)$$
$$Y_3 = Z_1Z_2(Z_1^2Z_2^2 + dX_1X_2Y_1Y_2)(Y_1Y_2 - aX_1X_2)$$
$$Z_3 = (Z_1^2Z_2^2 + dX_1X_2Y_1Y_2)(Z_1^2Z_2^2 - dX_1X_2Y_1Y_2)$$

Şekil 4.30: Twisted Edwards eğrilerde nokta toplama eşitlikleri.

Aşağıdaki Şekil 4.31'de verilen işlem maliyeti 10M + 1S + 2D + 7add olan algoritma ile de $R(X_3, Y_3, Z_3)$ noktası elde edilir.

$A = Z_1Z_2;$
$B = A^2;$
$C = X_1X_2;$
$D = Y_1Y_2;$
$E = dC - D;$
$F = B - E;$
$G = B + E;$
$X_3 = AF[(X_1 + Y_1)(X_2 + Y_2) - C - D];$
$Y_3 = AG(D - aC);$
$Z_3 = FG;$

Şekil 4.31: Twisted Edwards eğrileri için nokta toplama algoritması.

- Twisted Edwards eğrilerde nokta çiftleme işlemi:

Nokta toplama işleminde elde edilen eşitliklerde $Q(X_2 : Y_2 : Z_2)$ noktası yerine $P(X_1 : Y_1 : Z_1)$ konulduğunda nokta çiftleme formülü elde edilir.

$$\begin{aligned} X_3 &= Z_1^2(Z_1^4 - dX_1^2Y_1^2)[(X_1 + Y_1)^2 - X_1^2 - Y_1^2] \\ Y_3 &= Z_1^2(Z_1^4 + dX_1^2Y_1^2)(Y_1^2 - aX_1^2) \\ Z_3 &= (Z_1^4 + dX_1^2Y_1^2)(Z_1^4 - dX_1^2Y_1^2) \end{aligned} \quad (4.30)$$

$Z_1^4 - dX_1^2Y_1^2$ ifadesi yerine $-Z_1^2(aX_1^2 + Y_1^2 - 2Z_1^2)$, $Z_1^4 + dX_1^2Y_1^2$ ifadesi yerine de $Z_1^2(aX_1^2 + Y_1^2)$ ifadesini yazdığımızda,

$$\begin{aligned} X_3 &= (aX_1^2 + Y_1^2 - 2Z_1^2)[(X_1 + Y_1)^2 - X_1^2 - Y_1^2] \\ Y_3 &= (aX_1^2 + Y_1^2)(aX_1^2 - Y_1^2) \\ Z_3 &= (aX_1^2 + Y_1^2)(aX_1^2 - Y_1^2 - 2Z_1^2) \end{aligned} \tag{4.31}$$

sonuçlarını elde ederiz. Bu durumda aşağıda verilen işlem maliyeti 3M + 4S + 1D + 7add + 1times2 olan algoritma ile de çözüme ulaşabiliriz.

$B = (X_1 + Y_1)^2;$
$C = X_1^2;$
$D = Y_1^2;$
$E = aC;$
$F = E + D;$
$H = Z_1^2;$
$J = F - 2H;$
$X_3 = (B - C - D)J;$
$Y_3 = F(E - D);$
$Z_3 = FJ$

Şekil 4.32: Twisted Edwards eğrileri için nokta çiftleme algoritması.

4.10. Inverted Twisted Edwards Eğrileri

Gösterim olarak Inverted Edwards koordinatları $(X:Y:Z)$ şeklinde ifade edildiğinde, $P(x, y)$ noktası projektif koordinatlarda $P(Z/X, Z/Y)$ şeklinde ifade edilir. $XYZ \neq 0$ koşuluyla,

$E_c : x^2 + y^2 = c^2(1 + x^2y^2)$ Edwards eğrisinde, $x = Z/X$, $y = Z/Y$ dönüşümleri yapıldığında Inverted Edwards eğrisi aşağıdaki şekilde elde edilir [20]:

$$Z^2(X^2 + Y^2) = X^2Y^2 + dZ^4 \tag{4.32}$$

- Inverted twisted Edwards eğrilerde nokta toplama işlemi:

Nokta toplama formülünü elde etmek için Edwards koordinatlarda x_3 ve y_3 eşitliklerinde $x = {}^{Z}/_{X}, y = {}^{Z}/_{Y}$ konularak elde edilir. Bu durumda iki nokta toplamı denklemi, $P(X_1 : Y_1 : Z_1)$ ve $Q(X_2 : Y_2 : Z_2)$ olmak üzere $P + Q = R = (X_3 : Y_3 : Z_3)$

$$X_3 = (X_1X_2 - Y_1Y_2)(X_1X_2Y_1Y_2 + dZ_1^2Z_2^2)$$
$$Y_3 = (X_2Y_1 + X_1Y_2)(X_1X_2Y_1Y_2 - dZ_1^2Z_2^2)$$
$$Z_3 = (X_1X_2 - Y_1Y_2)(X_2Y_1 + X_1Y_2)Z_1Z_2$$

Şekil 4.33: Inverted twisted Edwards eğrilerde nokta toplama eşitlikleri.

eşitlikleri elde edilir ve aşağıdaki algoritmayı kullanarak 9M + 1S + 2D + 7add maliyetle eşitlikleri çözebiliriz.

$A = Z_1Z_2;$
$B = dA^2;$
$C = X_1X_2;$
$D = Y_1Y_2;$
$E = CD;$
$H = C - aD;$
$I = (X_1 + Y_1)(X_2 + Y_2) - C - D;$
$X_3 = c(E + B)H;$
$Y_3 = c(E - B)I;$
$Z_3 = AHI$

Şekil 4.34: Inverted twisted Edwards eğriler için nokta toplama algoritması.

- Inverted twisted Edwards eğrilerde nokta çiftleme işlemi:

Nokta çiftleme formülünü elde etmek için Edwards koordinatlarda $x_3\ ve\ y_3$ eşitliklerinde $x = {}^{Z}/_{X}, y = {}^{Z}/_{Y}$ konularak elde edilir. Bu durumda nokta çiftleme denklemi, $P(X_1 : Y_1 : Z_1)$ ve $2P = R$ olmak üzere,

$$X_3 = (X_1^2 Y_1^2 + dZ_1^4)\,(X_1^2 - Y_1^2)$$
$$Y_3 = 2X_1 Y_1 (X_1^2 Y_1^2 - dZ_1^4)$$
$$Z_3 = 2X_1 Y_1 Z_1^2 (X_1^2 - Y_1^2)$$

Şekil 4.35: Inverted twisted Edwards eğrilerde nokta çiftleme eşitlikleri.

eşitlikleri elde edilir. Aşağıda verilen algoritma ile çözüme ulaşılır. Bu durumda işlem maliyeti 3M + 4S + 2D + 6add + 1times2 olmaktadır.

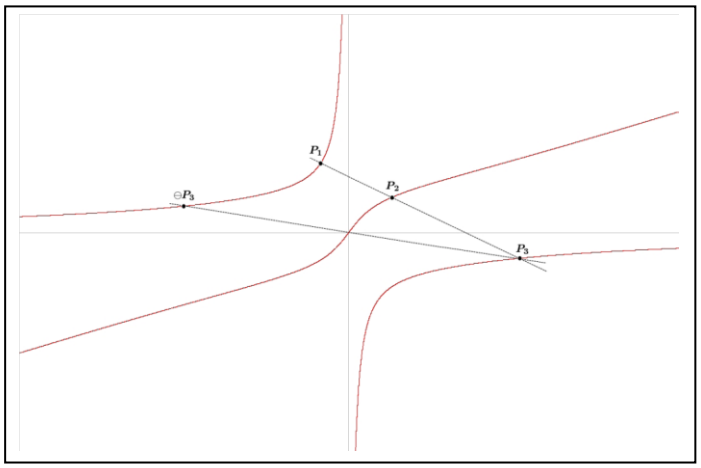
$$A = X_1^2;$$
$$B = Y_1^2;$$
$$U = aB;$$
$$C = A + U;$$
$$D = A - U;$$
$$E = (X_1 + Y_1)^2 - A - B;$$
$$X_3 = CD;$$
$$Y_3 = E(C - 2dZ_1^2);$$
$$Z_3 = DE$$

Şekil 4.36: Inverted twisted Edwards eğriler için nokta çiftleme algoritması.

4.11. Huff Eğrileri

1948 yılında Joye, Tibouchi ve Vergnaud tarafından geliştirilen bir eliptik eğri modelidir. Karakteristiği 2'den farklı olan F alanı için, $a, b \in F$ ve $a \neq b$ olan Huff eliptik eğri modeli aşağıdaki eşitlik ile tanımlanır [21]:

$$E_{a,b} = ax(y^2 - 1) = by(x^2 - 1) \tag{4.33}$$



Şekil 4.37: Huff eğrisi geometrik gösterimi.

İki noktanın toplamı formülü afin koordinatlarda $ab(a - b) \neq 0$ olmak şartıyla şu şekilde ortaya konmuştur. $P(x_1, y_1)\ ve\ Q(x_2, y_2)$ bu eğri üzerinde seçilen iki nokta olsun;

- $P + Q = R = (x_3, y_3)$
- $x_1 x_2 \neq \mp 1, y_1 y_2 \neq \mp 1$

$$\begin{aligned}(x_1, y_1) + (x_2, y_2) \\ = \left(\frac{(x_1 + x_2)(1 + x_1 x_2)}{(1 + x_1 x_2)(1 - y_1 y_2)}, \frac{(y_1 + y_2)(1 + x_1 x_2)}{(1 - x_1 x_2)(1 + y_1 y_2)}\right)\end{aligned} \tag{4.34}$$

Huff eğrilerinin projektif koordinatlarda gösterimi $x = {}^{X}/_{Z}, y = {}^{Y}/_{Z}$ olmak üzere,

$$aX(Y^2 - Z^2) = bY(X^2 - Z^2) \tag{4.35}$$

şeklinde tanımlanmıştır $(a, b \neq 0, a^2 - b^2 \neq 0)$.

- Huff eğrilerde nokta toplama işlemi:

Bu eğri üzerinde tanımlı olmak üzere $P(X_1:Y_1:Z_1)$ ve $Q(X_1:Y_1:Z_1)$ şeklinde iki noktanın toplamı; $P+Q=R=(X_3:Y_3:Z_3)$ olur. $(X_1Y_1Z_1 \neq 0)$

$$X_3 = (bX_1X_2 - Z_1Z_2)(bX_1X_2 + Z_1Z_2)(Z_1Z_2 - aY_1Y_2)$$
$$Y_3 = b(X_1Z_2 + X_2Z_1)(bX_1X_2 + Z_1Z_2)(Y_1Z_2 + Y_2Z_1)$$
$$Z_3 = b(X_1Z_2 + X_2Z_1)(bX_1X_2 - Z_1Z_2)(aY_1Y_2 + Z_1Z_2)$$

Şekil 4.38: Huff eğrilerde nokta toplama eşitlikleri.

Aşağıda ifade edilen algoritma [22] ile $R(X_3:Y_3:Z_3)$ noktasını bulabiliriz. İşlem maliyeti: 11M + 3D + 12add.

$A = X_1X_2;$
$B = Y_1Y_2;$
$D = Z_1Z_2;$
$E = bA;$
$F = aB;$
$G = (X_1 + Z_1)(X_2 + Z_2) - A - D;$
$H = (Y_1 + Z_1)(Y_2 + Z_2) - B - D;$
$X_3 = \left(1/b\right)(E + D)(E - D)(D - F);$
$Y_3 = GH(E + D);$
$Z_3 = G(E - D)(F + D);$

Şekil 4.39: Huff eğrilerde nokta toplama algoritması.

- Huff eğrilerde nokta çiftleme işlemi:

$X(aY^2 - Z^2) = Y(bX^2 - Z^2)$, eğrisi üzerinde nokta çiftleme işlemi için $2P = R = (X_3:Y_3:Z_3)$, işlem maliyeti: 6M + 5S + 2D + 10add.

$A = X_1^2;$
$B = Y_1^2;$
$D = Z_1^2;$
$E = bA;$
$F = aB;$
$G = (X_1 + Z_1)^2 - A - C;$
$H = (Y_1 + Z_1)^2 - B - C;$
$X_3 = (D - C)(D - C)(C - E);$
$Y_3 = FG(C + D);$
$Z_3 = F(D - C)(C + E);$

Şekil 4.40: Huff eğrilerde nokta çiftleme algoritması.

4.12. Lopez Dahab Eğrileri

1998 yılında Julio Lopez ve Ricardo Dahab tarafından karakteristiği 2 olan($char\ (F) = 2$) alanlar için geliştirilmiştir. Lopez-Dahab projektif koordinatları genellikle ikili alan (binary fields) üzerinde elpitik eğri uygulamaları gerçekleştirebilmek için tercih edilir. İkili sonlu alanda pratik uygulamalarda daha iyi bir performans elde edebilmek için bu koordinatlar kullanılır. $P(X_1, Y_1, Z_1)$ ve $Q(X_2, Y_2, Z_2)$ olmak üzere iki projektif nokta alındığında, $Z \neq 0$ olmamak şartıyla $x = {X}/{Z}$, $y = {Y}/{Z^2}$ aşağıdaki eşitlikte yerine konursa,

$$y^2 + xy = x^3 + ax^2 + b \tag{4.36}$$

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4 \tag{4.37}$$

eşitlikleri elde edilir [25].

- Lopez Dahab nokta toplama işlemi:

Lopez Dahab projektif koordinatlarda eliptik eğri noktalarının toplam ifadesi şu şekildedir: $(X_1, Y_1, Z_1) + (X_2, Y_2, Z_2) = (X_3, Y_3, Z_3)$. (4.37)'de belirtilen eğri

denklemi ve aşağıdaki algoritma ile 13M + 4S + 9add maliyetle nokta toplamı elde edilebilir:

1. $A = X_1 Z_2$;
2. $B = X_2 Z_1$;
3. $C = A^2$;
4. $D = B^2$;
5. $E = A + B$;
6. $F = C + D$;
7. $G = Y_1 Z_2^2$;
8. $H = Y_2 Z_1^2$;
9. $I = G + H$;
10. $J = IE$;
11. $Z_3 = F Z_1 Z_2$;
12. $X_3 = A(H + D) + B(C + G)$;
13. $Y_3 = (AJ + FG)F + (J + Z_3)X_3$;

Şekil 4.41: Lopez Dahab eğrilerde nokta toplama algoritması.

- Lopez Dahab nokta çiftleme işlemi:

Lopez Dahab projektif koordinatlarda eliptik eğri nokta çiftleme işlemi $2(X_1, Y_1, Z_1) = (X_3, Y_3, Z_3)$ şeklinde gösterecek olursak, iş maliyeti 4M + 5S + 1C + 5add olan algoritma ile R noktası elde edilebilmektedir.

$A = X_1 Z_1; Z_3 = A^2;$
$B = X_1^2; X_3 = C^2 + D + a^2 Z_3;$
$C = B + Y_1; Y_3 = (Z_3 + D)X_3 + B^2 Z_3$
$D = AC;$

Şekil 4.42: Lopez Dahab eğrilerde nokta çiftleme algoritması.

4.13. Hessian Eğrileri

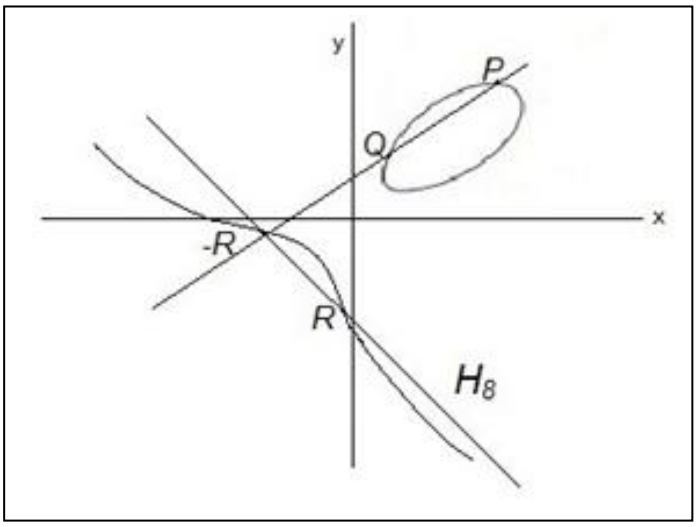
Hessian eliptik eğrileri Joye ve Quisquater tarafından ortaya çıkarılmıştır. Karakteristiği 2 ve 3'den farklı ($char(F) \neq 2,3$) sonlu bir F alanı üzerinde eğri denklemi:

$$H_d: x^3 + y^3 + 1 = 3dxy \tag{4.38}$$

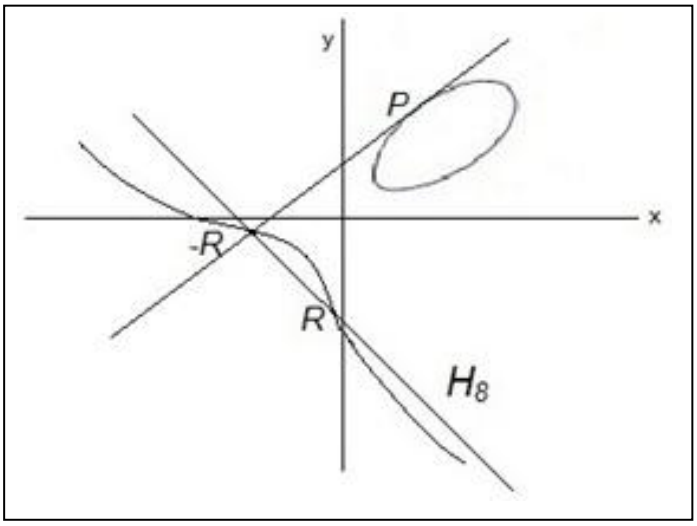
- Projektif koordinatlarda $d \in F$, $d^3 \neq 1$ olmak koşuluyla,

$$H_d: X^3 + Y^3 + Z^3 = 3dXYZ \tag{4.39}$$

olarak tanımlanmıştır [23].



Şekil 4.43: Hessian eğriler nokta toplama geometrik gösterimi.



Şekil 4.44: Hessian eğriler nokta çiftleme geometrik gösterimi.

- Hessian eğrilerde nokta toplama işlemi:

Bu eğri üzerinde tanımlı olmak üzere $P(X_1:Y_1:Z_1)$ ve $Q(X_1:Y_1:Z_1)$ şeklinde iki noktanın toplamı; $P+Q=R=(X_3:Y_3:Z_3)$ olur ve aşağıda verilen eşitliklerle elde edilebilir.

$$X_3 = Y_1^2 X_2 Z_2 - Y_2^2 X_1 Z_1$$
$$Y_3 = X_1^2 Y_2 Z_2 - X_2^2 Y_1 Z_1$$
$$Z_3 = Z_1^2 Y_2 X_2 - Z_2^2 Y_1 X_1$$

Şekil 4.45: Hessian eğriler nokta toplama eşitlikleri.

Aşağıdaki algoritma ile $(X_3:Y_3:Z_3)$ noktasını 12M + 3add maaliyetle hesaplayabiliriz.

1. $T_1 = X_1;$	14. $T_2 = T_2 T_6;$
2. $T_2 = Y_1;$	15. $T_6 = T_2 T_7;$
3. $T_3 = Z_1;$	16. $T_2 = T_2 T_4;$
4. $T_4 = X_2;$	17. $T_4 = T_3 T_4;$
5. $T_5 = Y_2;$	18. $T_3 = T_3 T_5;$
6. $T_6 = Z_2;$	19. $T_5 = T_1 T_5;$
7. $T_7 = T_1 T_6;$	20. $T_1 = T_1 T_7;$
8. $T_1 = T_1 T_5;$	21. $T_1 = T_1 - T_4;$
9. $T_5 = T_3 T_5;$	22. $T_2 = T_2 - T_5;$
10. $T_3 = T_3 T_4;$	23. $T_3 = T_3 - T_6;$
11. $T_4 = T_2 T_4;$	24. $X_3 = T_2;$
12. $T_2 = T_2 T_6;$	25. $Y_3 = T_1;$
13. $T_6 = T_2 T_7;$	26. $Z_3 = T_3;$.

Şekil 4.46: Hessian eğri için nokta toplama algoritması.

- Hessian eğrilerde nokta çiftleme işlemi:

İki tane P noktasını topladığımızda $2P = R = (X_3 : Y_3 : Z_3)$'yi elde edilir ve aşağıdaki şekilde gösterilebilir:

$$X_3 = Y_1(Z_1^3 - X_1^3)$$
$$Y_3 = X_1(Y_1^3 - Z_1^3)$$
$$Z_3 = Z_1(X_1^3 - Y_1^3)$$

Şekil 4.47: Hessian eğrilerde nokta çiftleme eşitlikleri.

İşlem maliyeti 7M + 1S + 8add olan aşağıdaki algoritma ile $R(X_3 : Y_3 : Z_3)$ noktasını hesaplayabiliriz.

1. $T_1 = X_1^2;$
2. $T_2 = X_1 + Y_1;$
3. $T_2 = Y_1 T_2;$
4. $T_3 = Z_1 + X_1;$
5. $T_3 = Z_1 T_3;$
6. $T_3 = T_1 + T_3;$
7. $T_2 = T_1 + T_2;$
8. $T_1 = X_1 - Y_1;$
9. $T_1 = T_2 T_1;$
10. $Z_3 = T_1 Z_1;$
11. $T_2 = Z_1 - X_1;$
12. $T_2 = T_3 T_2;$
13. $X_3 = T_2 Y_1;$
14. $T_3 = -(T_1 + T_2);$
15. $Y_3 = T_3 X_1;$

Şekil 4.48: Hessian eğrilerde nokta çiftleme algoritması.

4.14. Twisted Hessian Eğrileri

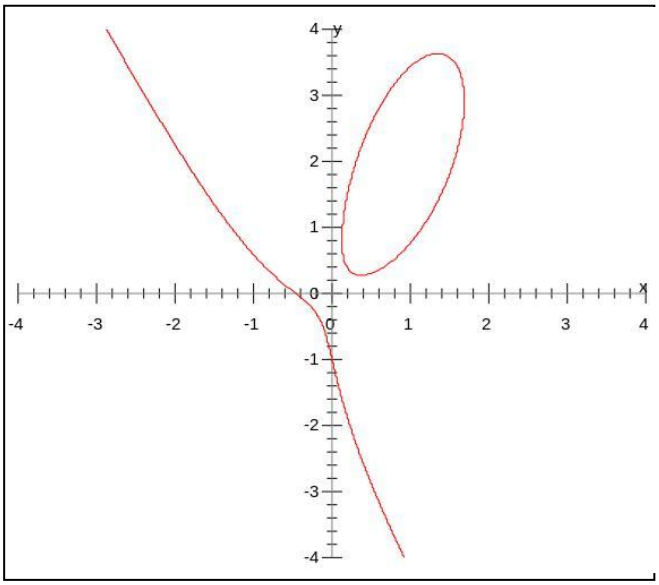
Twisted Hessian eliptik eğrileri Berstein, Kohel ve Lange [24] tarafından karakteristiği 2 ve 3'den farklı $(char(F) \neq 2,3)$ sonlu bir F alanı üzerinde eğri denklemi şu şekilde tanımlanmıştır:

- $a, d \in F, a \neq 0, d^3 \neq 27c$

$$H_{a,d}: ax^3 + y^3 + 1 = dxy \tag{4.40}$$

- Projektif koordinatlarda $x = {}^{X}/_{Z}, y = {}^{Y}/_{Z}$ olarak ifade ettiğimizde:

$$H_{a,d}: aX^3 + Y^3 + Z^3 = dXYZ \tag{4.41}$$



Şekil 4.49: Twisted Hessian eğrileri geometrik gösterimi.

- Twisted Hessian eğrilerde nokta toplama işlemi:

Projektif koordinatlarda $H_{a,d}$ eğrisi üzerinde iki nokta $P = (X_1, Y_1, Z_1)$ ve Q= (X_2, Y_2, Z_2) olsun; $P + Q = R = (X_3 : Y_3 : Z_3),\ Z_1 \neq 0,\ Z_2 \neq 0,\ P \neq Q$

$$X_3 = X_1 Z_1 Z_2^2 - Y_1^2 X_2 Y_2$$
$$Y_3 = Y_1 Z_1 Y_2^2 - a X_1^2 X_2 Z_2$$
$$Z_3 = a X_1 Y_1 X_2^2 - Z_1^2 Z_2 Y_2$$

Şekil 4.50: Twisted Hessian eğrilerde nokta toplama eşitlikleri.

- Aşağıdaki algoritma ile 12M + 1D + 3add maliyetle R noktası elde edilebilir:

$$A = X_1Y_1; D = Y_1Y_2; X_3 = AB - CD;$$
$$B = Z_1Z_2;\ E = Z_1Y_2; Y_3 = DE - FA;$$
$$C = X_2Y_1; F = aX_1X_2; Z_3 = FC - BE;$$

Şekil 4.51: Twisted Hessian eğrileri için nokta toplama algoritması.

- Twisted Hessian eğrilerde nokta çiftleme işlemi:

Projektif koordinatlarda $H_{a,d}$ eğrisi üzerinde bir nokta $P = (X_1, Y_1, Z_1)$ ve Q= (X_2, Y_2, Z_2) olsun, $2P = R = (X_3 : Y_3 : Z_3),\ Z_1 \neq 0, Z_3 \neq 0$

$$X_3 = X_1(Z_1^3 - Y_1^3)$$
$$Y_3 = Z_1(Y_1^3 - aX_1^3)$$
$$Z_3 = Y_1(aX_1^3 - Z_1^3)$$

Şekil 4.52: Twisted Hessian eğrilerinde nokta çiftleme eşitlikleri.

- Aşağıdaki algoritma ile 6M+3S+1C+3add maliyetle R noktası elde edilebilir.

$$A = X_1^2$$
$$B = Y_1^2$$
$$C = Z_1^2$$
$$D = X_1A$$
$$E = Y_1B$$
$$F = Z_1C$$
$$G = aD$$
$$X_3 = X_1(E - F)$$
$$Y_3 = Z_1(G - E),$$
$$Z_3 = Y_1(F - G)$$

Şekil 4.53: Twisted Hessian eğrileri için nokta çiftleme algoritması.

5. ELİPTİK EĞRİ KÜTÜPHANE YAZILIMI

Bu bölümde eliptik eğri formları için geliştirilen kütüphane uygulaması ele alınmıştır. 4. bölümde incelenmiş olan eliptik eğri formlarının nokta toplama ve nokta çiftleme işlemleri c# yazılım dili kullanarak .Net frameworkünde gerçeklenmiştir. Microsoft .Net kütüphanesindeki BigInteger yapısı kullanılarak 192 bitlik sayılarla işlemler yapılmıştır. Yazılan algoritmaların asal cisimlerde tanımlanmış olan NIST rutinleri ile testleri yapılmış, performans ve maliyet sonuçları değerlendirilmiştir.

5.1. Yazılıma Ait Aritmetik Fonksiyonların Tanımı

Geliştirilen kütüphanede "*prime*", modInt sınıfının nesnesini oluşturmaktadır. modInt sınıfı modüler aritmetik işlemlerini yapan sınıftır. Bu sınıf içerisinde tersini alma, çarpma, toplama, çıkarma ve indirgeme(mod alma) işlemleri .Net BigInteger kütüphanesi kullanarak tanımlanmıştır.

- modSub:

Eliptik eğriler için tanımlanan BigInteger çıkarma (subtraction) işlemini yapan metottur. Giriş değeri olarak byte tipinde verilen iki noktayı biribirinden çıkartır.

```
public void modSub(byte[] a, byte[] b, byte[] c)
{
Array.Copy(a, buf0, length);
Array.Copy(b, buf1, length);
Array.Clear(bufx, 0, length);
BigInteger tmp = BigInteger.Remainder(new BigInteger(buf0) - new
BigInteger(buf1), modulusBig);
if(tmp.Sign == -1)      tmp+= modulusBig;
if(tmp.Sign == -1)      tmp+= modulusBig;
tmp.ToByteArray().CopyTo(bufx, 0);
Array.Copy(bufx, c, length);
}
```

Şekil 5.1: Modüler çıkarma işlemi.

- modAdd:

Eliptik eğriler için tanımladığımız BigInteger toplama (addition) işlemini yapan metottur. Giriş değeri olarak byte tipinde verilen iki noktayı toplar.

```
public void modAdd(byte[] a, byte[] b, byte[] c)
{
Array.Copy(a, buf0, length);
Array.Copy(b, buf1, length);
Array.Clear(bufx, 0, length);
BigInteger temp = BigInteger.Remainder(new BigInteger(buf0) + new
BigInteger(buf1), modulusBig);
temp.ToByteArray().CopyTo(bufx, 0);
Array.Copy(bufx, c, length);
 }
```

Şekil 5.2: Modüler toplama işlemi.

- modMul:

Eliptik eğriler için tanımlanan BigInteger çarpma (multiplication) işlemini yapan metottur. Giriş değeri olarak byte tipinde verilen iki noktanın çarpım işlemini gerçekleştirir.

```
public void modMul(byte[] a, byte[] b, byte[] c)
{
Array.Copy(a, buf0, length);
Array.Copy(b, buf1, length);
Array.Clear(bufx, 0, length);
BigInteger.Remainder(new BigInteger(buf0) * new BigInteger(buf1),
modulusBig).ToByteArray().CopyTo(bufx, 0);
Array.Copy(bufx, c, length);
}
```

Şekil 5.3: Modüler çarpma işlemi.

- modInv:

Eliptik eğriler için tanımlanan BigInteger tersini alma (inversion) işlemini yapan metottur. Giriş değeri olarak byte tipinde verilen değerin çarpmaya göre tersinin bulunmasını sağlar. Inverse işlemi için Penk algoritması [10] kullanılmıştır.

```
public void modInv(byte[] a, byte[] c)
{
      Array.Copy(a, buf0, length);
      BigInteger u = new BigInteger(buf0);
      Array.Clear(c, 0, length);

      if(u.IsZero)return; c[0] = 1;
      if(u.IsOne)return;

      BigInteger r = 1; BigInteger v = modulus;
      BigInteger s = 0; Array.Clear(bufx, 0, length);
      Boolean reduce = true;

      do
      {int notBothOdd = 2;
           if(u.IsEven){
           u>>=1;
      if(r.IsEven)    r>>=1;
      else   r = (r+modulus)>>1;
      }
      elsenotBothOdd--;
      if(v.IsEven){
      v>>=1;
      if(s.IsEven)    s>>=1;
      elses = (s+modulus)>>1;
      }
      else notBothOdd--;
      if(notBothOdd == 0){
      if(u>v){
      u = u-v;
      if(r>s) r = r-s;
      else  r = r+modulus-s;
      }
      else{
      v = v-u;
      if(s>r)s = s-r;
      else  s = s+modulus-r;
      }
      }
      if(u.IsOne){
      reduce = false; r.ToByteArray().CopyTo(bufx,0);
      }
      if(v.IsOne){
      reduce = false; s.ToByteArray().CopyTo(bufx,0);
      }
      }
      while(reduce); Array.Copy(bufx, c, length);
      return;
}
```

Şekil 5.4: Penk inverse işlemi.

- modComplete:

Eliptik eğriler için tanımlanan BigInteger indirgeme (reduction), mod alma işlemini yapan metottur. Her işlemin sonunda yapılması gereken bir aritmetik işlemdir. Yapılan işlemlerin F_p cismi üzerinde olduğunu garanti eder.

```
public void modComplete(byte[] a, byte[] c)
{
Array.Copy(a, buf0, length);
Array.Clear(bufx, 0, length);
(new BigInteger(buf0) %modulus)
.ToByteArray().CopyTo(bufx,0);
Array.Copy(bufx, c, length);
}
```

Şekil 5.5: Mod alma işlemi.

5.2. NIST Rutinleri

NISTConstans sınıfında NIST tarafından önerilen 192 bitlik asal sayılarla ifade edilen S ve T noktalarına ait nokta toplama, nokta çıkarma, nokta çiftleme ve nokta skaler çarpımları const string olarak tanımlanmıştır. Eliptik eğri formlarında yapılacak olan nokta aritmetik işlemleri bu sınıf kullanılarak gerçekleştirilmiş ve yine işlem sonuçları bu sınıfta belirtilen işlem sonuçları ile karşılaştırılarak işlem doğruluğu kontrol edilmiştir.

```
static class NISTConstants
{

     public const string  P192_POINT_S_X =
     "d458e7d1 27ae671b 0c330266 d2467693 53a01207 3e97acf8";

     public const string  P192_POINT_S_Y =
     "32593050 0d851f33 6bddc050 cf7fb11b 5673a164 5086df3b";

     public const string  P192_POINT_S_Z =
     "00000000 00000000 00000000 00000000 00000000 00000001";

     public const string  P192_POINT_T_X =
     "f22c4395 213e9ebe 67ddecdd 87fdbd01 be16fb05 9b9753a4";

     public const string  P192_POINT_T_Y =
     "26442409 6af2b359 7796db48 f8dfb41f a9cecc97 691a9c79";

     public const string  P192_POINT_T_Z =
     "00000000 00000000 00000000 00000000 00000000 00000001";

     public const string  P192_S_PLUS_T_X =
     "48e1e409 6b9b8e5c a9d0f1f0 77b8abf5 8e843894 de4d0290";

     public const string  P192_S_PLUS_T_Y =
     "408fa77c 797cd7db fb16aa48 a3648d3d 63c94117 d7b6aa4b";

     public const string  P192_S_MINUS_T_X =
     "fc9683cc 5abfb4fe 0cc8cc3b c9f61eab c4688f11 e9f64a2e";

     public const string  P192_S_MINUS_T_Y =
     "093e31d0 0fb78269 732b1bd2 a73c23cd d31745d0 523d816b";

     public const string  P192_2S_X =
     "30c5bc6b 8c7da253 54b373dc 14dd8a0e ba42d25a 3f6e6962";

     public const string  P192_2S_Y =
     "0dde14bc 4249a721 c407aedb f011e2dd bbcb2968 c9d889cf";

     public const string  P192_SCALAR_D =
     "a78a236d 60baec0c 5dd41b33 a542463a 8255391a f64c74ee";

     public const string  P192_D_TIMES_S_X =
     "1faee420 5a4f669d 2d0a8f25 e3bcec9a 62a69529 65bf6d31";

     public const string  P192_D_TIMES_S_Y =
     "5ff2cdfa 508a2581 89236708 7c696f17 9e7a4d7e 8260fb06";

}
```

Şekil 5.6: NIST rutinleri giriş fonksiyonu.

5.3. Koordinat Dönüşümleri ve Yardımcı Fonksiyonlar

Bu bölümde projektif, Jacobian ve Chudnovsky koordinat sistemlerinden afin koordinat sistemlerine dönüşüm fonksiyonları tanımlanmıştır. Byte cinsinden verilen noktanın modüler aritmetik işlemlerinden geçirilerek afin koordinatlara dönüşümleri sağlanmıştır. Ayrıca string-byte dönüşümlerini yapan fonksiyonlar ifade edilmiştir.

- jacobianToAffineX:

Noktaya ait X parametresinin, Jacobian koordinatlardan afin koordinatlara dönüşümünü sağlar.

```
public static byte[] toAffineX(Point point, modInt prime)
{
// x = X / Z^2
     byte[] Z2 = new byte[point.Z.Length];
prime.modMul(point.Z, point.Z, Z2);
prime.modInv(Z2, Z2);
prime.modMul(point.X, Z2, Z2);
return Z2;
}
```

Şekil 5.7: X için Jacobian koordinatlardan afin koordinatlara dönüşüm fonksiyonu.

- jacobianToAffineY:

Noktaya ait Y parametresinin, Jacobian koordinatlardan afin koordinatlara dönüşümünü sağlar.

```
public static byte[] toAffineY(Point point, modInt prime)
{
// y = Y / Z^3
byte[] Z3 = new byte[point.Z.Length];
prime.modMul(point.Z, point.Z, Z3);
prime.modMul(point.Z, Z3, Z3);
prime.modInv(Z3, Z3);
prime.modMul(point.Y, Z3, Z3);
return Z3;
}
```

Şekil 5.8: Y için Jacobian koordinatlardan afin koordinatlara dönüşüm fonksiyonu.

- projectiveToAffineX:

Noktaya ait X parametresinin, projektif koordinatlardan afin koordinatlara dönüşümünü sağlar.

```
public static byte[] erdwardsToAffineX(Point point, modInt prime)
{
// x = X/Z
     byte[] Z3 = new byte[point.Z.Length];
     prime.modInv(point.Z, Z3); prime.modMul(point.X, Z3, Z3);
return Z3;
}
```

Şekil 5.9: X için projektif koordinatlardan afin koordinatlara dönüşüm fonksiyonu.

- projectiveToAffineY:

Noktaya ait Y parametresinin, projektif koordinatlardan afin koordinatlara dönüşümünü sağlar.

```
public static byte[] erdwardsToAffineY(Point point, modInt prime)
{// y = Y/Z
    byte[] Z3 = new byte[point.Z.Length];
     prime.modInv(point.Z, Z3);  prime.modMul(point.Y, Z3, Z3);
     return Z3;
}
```

Şekil 5.10: Y için projektif koordinatlardan afin koordinatlara dönüşüm fonksiyonu.

- extendedJQToAffineY:

Noktaya ait Y parametresinin, extended Jacobian projektif koordinatlardan afin koordinatlara dönüşümünü sağlar.

```
public static byte[] extendedJQToAffineY(Point point, modInt
prime)
{   // y = Y/Z^2
     byte[] Z3 = new byte[point.Z.Length];
     prime.modMul(point.Z, point.Z, Z3);
     prime.modInv(Z3, Z3);
     prime.modMul(point.Y, Z3, Z3);     return Z3; }
```

Şekil 5.11: Y için extended projektif koordinatlardan afin koordinatlara dönüşüm.

- invertToAffineX:

Noktaya ait X parametresinin, inverted projektif koordinatlardan afin koordinatlara dönüşümünü sağlar.

```
public static byte[] invertToAffineX(Point point, modInt prime)
{
// x = Z/X
    byte[] Z3 = new byte[point.Z.Length];
     prime.modInv(point.X, Z3);
     prime.modMul(point.Z, Z3, Z3);
     return Z3;
}
```

Şekil 5.12: X için inverted projektif koordinatlardan afin koordinatlara dönüşüm fonksiyonu.

- invertToAffineY:

Noktaya ait Y parametresinin, inverted projektif koordinatlardan afin koordinatlara dönüşümünü sağlar.

```
public static byte[] invertToAffineY(Point point, modInt prime)
{
// y = Z/Y
     byte[] Z3 = new byte[point.Z.Length];
     prime.modInv(point.Y, Z3);
     prime.modMul(point.Z, Z3, Z3);
     return Z3;
}
```

Şekil 5.13: Y için inverted projektif koordinatlardan afin koordinatlara dönüşüm fonksiyonu.

- StringToByteArray:

String olarak verilen NIST sabitlerini byte-array'e dönüştürmeye yarayan fonksiyondur.

```
public static byte[] StringToByteArray(string hex)
{
return Enumerable.Range(0, hex.Length).Where(x=>x % 2 == 0)
.Select(x => Convert.ToByte(hex.Substring(hex.Length-x-2, 2),
16)).ToArray();
}
```

Şekil 5.14: String'ten byte-array'e dönüşüm fonksiyonu.

- ByteArrayToHexString:

Byte array olarak elde edilen sonuçların NIST rutinleri ile karşılaştırmak için hex string'e çeviren fonksiyondur.

```
public static string ByteArrayToHexString(byte[] hex)
{
string str = ByteArrayToString(hex); string s = "";
for (int i = 0; i < str.Length; i++)
if(i%2==0) s += str.Substring(str.Length - i - 2, 2);
return s;
}
```

Şekil 5.15: Byte array'den string'e dönüşüm fonksiyonu.

- affineToExtendedJQX:

Noktaya ait X parametresinin, afin koordinatlardan extended Jacobian Projektif koordinatlara dönüşümünü sağlar.

```
public static byte[] affineToExtendedJQX(Point point,
  modInt prime)
{
     // x = X^2/Z
     byte[] Z3 = new byte[point.Z.Length];
     prime.modInv(point.Z, Z3);
     prime.modMul(point.X, Z3, Z3);
     prime.modMul(point.X, Z3, Z3);
     return Z3;
}
```

Şekil 5.16: X için afin koordinatlardan extended Jacobian koordinatlara dönüşüm fonksiyonu.

5.4. Eğri Üzerindeki Noktayı Tanımlayan Sınıf

- CurveP:

```
abstract class CurveP
{
public abstract ushort info {get;}
    public abstract ushort length {get;}
     public abstract byte[] prime {get;}
     public abstract byte[] aCoef {get;}
     public abstract byte[] bCoef {get;}
     public abstract byte[] xBase {get;}
     public abstract byte[] yBase {get;}
     public abstract byte[] times2 {get;}
     public abstract byte[] times3 {get;}
     public abstract byte[] inv2 {get;}

}
```

Şekil 5.17: Eğri üzerinde nokta parametrelerini tanımlayan fonksiyon.

- CurveP192:

```
class CurveP192 : CurveP
{
  private static readonly ushort _info = 24;
  private static readonly ushort _length = 24;
  private static readonly byte[] _prime = {
  0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xfe,0xff,0xff,0xff,
  0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
  };
  private static readonlybyte[] _aCoef = {
  0xfc,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xfe,0xff,0xff,0xff,
  0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
  };
  private static readonly byte[] _bCoef = {
  0xb1,0xb9,0x46,0xc1,0xec,0xde,0xb8,0xfe,0x49,0x30,0x24,0x72,
  0xab,0xe9,0xa7,0x0f,0xe7,0x80,0x9c,0xe5,0x19,0x05,0x21,0x64
  };
  private static readonly byte[] _xBase = {
  0x12,0x10,0xff,0x82,0xfd,0x0a,0xff,0xf4,0x00,0x88,0xa1,0x43,
  0xeb,0x20,0xbf,0x7c,0xf6,0x90,0x30,0xb0,0x0e,0xa8,0x8d,0x18
  };
```

Şekil 5.18: Eğri üzerinde 192 bitlik noktayı tanımlayan fonksiyon.

```
  private static readonly byte[]  _yBase = {
  0x11,0x48,0x79,0x1e, 0xa1,0x77,0xf9,0x73,0xd5,0xcd,0x24,0x6b,
  0xed,0x11,0x10,0x63, 0x78,0xda,0xc8,0xff, 0x95,0x2b,0x19,0x07
  };
  private static readonly byte[] _times2 = {
  0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
  };
  private static readonly byte[] _inv2 = {
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0xff,0xff,
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x7f
  };
  private static readonly byte[] _times3 = {
    0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
  };
  public override ushort info {get{return _info;  }}
  public override ushort length {get{return _length;}}
  public override byte[] prime {get{return _prime; }}
  public override byte[] aCoef {get{return _aCoef; }}
  public override byte[] bCoef {get{return _bCoef; }}
  public override byte[] xBase {get{return _xBase; }}
  public override byte[] yBase {get{return _yBase; }}
  public override byte[] times2 {get{return _times2; }}
  public override byte[] times3 {get{return _times3; }}
  public override byte[] times3 {get{return _inv2; }}
 }
```

Şekil 5.18: Devam.

- Nokta kopyalama (copyPoint):

P gibi AffinePoint türünden tanımlı bir noktayı yine AffinePoint türünden tanımlı R gibi bir noktaya kopyalar.

```
public void copyPoint(PointAffine P, PointAffine R)
{
R.info = P.info;
Array.Copy(P.X, R.X, prime.length);
Array.Copy(P.Y, R.Y, prime.length);
return;
}
```

Şekil 5.19: Eğri üzerinde 192 bitlik noktayı kopyalayan fonksiyon.

- İki boyutlu nokta tanımı:

```
class PointAffine
{
public uintinfo;
public byte[] X;public  byte[]  Y;
public PointAffine(CurveP curve)
{
info  = 0;
     X = new byte[curve.length];
Array.Clear(X, 0,curve.length);
curve.xBase.CopyTo(X, 0);
     Y = new byte[curve.length];
Array.Clear(Y, 0, curve.length);
curve.yBase.CopyTo(Y, 0);
}
public PointAffine(CurveP curve, byte[] inpX, byte[] inpY)
{
info  = 0;
X = new byte[curve.length];
Array.Clear(X, 0,curve.length);
inpX.CopyTo(X, 0);
     Y = new byte[curve.length];
Array.Clear(Y, 0, curve.length);
inpY.CopyTo(Y, 0);
}
}
```

Şekil 5.20: Eğri üzerinde iki boyutlu nokta tanımlayan fonksiyon.

- Üç boyutlu nokta tanımı:

```
class Point
{
public uint info;
public byte[] X;public byte[] Y;public byte[] Z;
public Point(CurveP curve, byte[] inpX,
byte[] inpY, byte[] inpZ)
{
info  = 0;
    X = new byte[curve.length];
Array.Clear(X, 0, curve.length);     inpX.CopyTo(X, 0);
     Y = new byte[curve.length];
Array.Clear(Y, 0, curve.length);     inpY.CopyTo(Y, 0);
Z = new byte[curve.length];
Array.Clear(Z, 0, curve.length);     inpZ.CopyTo(Z, 0);
}
}
```

Şekil 5.21: Eğri üzerinde üç boyutlu nokta tanımlayan fonksiyon.

5.5. WeierstrassCurve Sınıfı

Weierstrass eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public WeierstrassCurve(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
aCoef = curve.aCoef;
coef2 = curve.times2;
     bufU1 = new byte[len];
bufU2 = new byte[len];
     bufS1 = new byte[len];
bufS2 = new byte[len];
     bufW1 = new byte[len];
bufW2 = new byte[len];
     bufP1 = new byte[len];
bufP2 = new byte[len];
     bufR1 = new byte[len];
bufR2 = new byte[len];
     coef3   = curve.times3;
inv2 = curve.inv2;
}
```

Şekil 5.22: Weierstrass eğrisini tanımlayan fonksiyon.

- addPoint:

Projektif koordinatlarda Şekil 4.1 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(PointAffine P, PointAffine Q, PointAffine R)
{
prime.modMul(P.X, Q.Z, bufU1);
prime.modMul(Q.X, P.Z, bufU2);
prime.modMul(P.Y, Q.Z, bufS1);
prime.modMul(Q.Y, P.Z, bufS2);
prime.modMul(P.Z, Q.Z, bufW1);
prime.modAdd(bufU1, bufU2, bufW2);
prime.modSub(bufU2, bufU1, bufP1);
prime.modMul(bufP1, bufP1, bufP2);
prime.modSub(bufS2, bufS1, bufR1);
prime.modMul(bufR1, bufR1, bufR2);
prime.modMul(bufW1, bufR2, bufU1);
prime.modMul(bufW2, bufP2, bufU2);
prime.modSub(bufU1, bufU2, R.X);
prime.modMul(R.X, bufP1, R.X);
prime.modMul(coef2, bufU1, bufU1);
prime.modMul(coef3, bufU2, bufU2);
prime.modSub(bufU2, bufU1, R.Y);
prime.modMul(R.Y, bufR1, R.Y);
prime.modMul(bufP1, bufP2, bufU1);
prime.modAdd(bufS1, bufS2, bufU2);
prime.modMul(bufU1, bufU2, bufU2);
prime.modSub(R.Y, bufU2, R.Y);
prime.modMul(R.Y, inv2, R.Y);
prime.modMul(bufW1, bufU1, R.Z);
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.23: Weierstrass eğrisinde nokta toplama işlemini yapan fonksiyon.

- doublePoint:

Projektif koordinatlarda Şekil 4.2 ile ifade edilen nokta çiftleme işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint(PointAffine P, PointAffine R) // P + P =
R
{
prime.modMul(P.X, P.X, bufU1);
prime.modMul(P.Z, P.Z, bufW1);
prime.modMul(coef3, bufU1, bufS1);
prime.modMul(aCoef, bufW1, bufW1);
prime.modAdd(bufW1, bufS1, bufS1);
prime.modMul(P.Y, P.Z, bufS2);
prime.modMul(coef2, bufS2, bufS2);
prime.modMul(bufS2, bufS2, bufP1);
prime.modMul(bufS2, bufP1, bufP2);
prime.modMul(P.Y, bufS2, bufR1);
prime.modMul(bufR1, bufR1, bufR2);
prime.modAdd(P.X, bufR1, bufW2);
prime.modMul(bufW2, bufW2, bufW2);
prime.modSub(bufW2, bufU1, bufW2);
prime.modSub(bufW2, bufR2, bufW2);
prime.modMul(bufS1, bufS1, bufU2);
prime.modMul(coef2, bufW2, bufU1);
prime.modSub(bufU2, bufU1, bufU2);
prime.modMul(bufU2,bufS2, R.X);
prime.modSub(bufW2, bufU2, R.Y);
prime.modMul(R.Y, bufS1, R.Y);
prime.modMul(coef2, bufR2, bufR2);
prime.modSub(R.Y, bufR2, R.Y);
bufP2.CopyTo(R.Z, 0);
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.24: Weierstrass eğrisinde nokta çiftleme işlemini yapan fonksiyon.

5.6. JacobianCurve Sınıfı

Jacobian eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public JacobianCurve(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
T1 = new byte[len];
T2 = new byte[len];
T3 = new byte[len];
T4 = new byte[len];
inv2 = curve.inv2;
coef2 = curve.times2;
coef3 = curve.times3;
}
```

Şekil 5.25: Jacobian eğri parametrelerini tanımlayan fonksiyon.

- addPoint:

Jacobian koordinatlarda Şekil 4.4 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(Point P, Point Q, Point R)
{
int idx = prime.length;
if (P.info == 1)
{
    R.info = Q.info;
    Q.X.CopyTo(R.X, 0);  Q.Y.CopyTo(R.Y, 0);
    int i = prime.length;
    while (--i >= 0)     R.Z[i] = 0x00; R.Z[0] = 0x01;
    long t = BitConverter.ToInt16(R.Z,0);
    return; // R
}

if (Q.info == 1)
{
    R.info = P.info;
    P.X.CopyTo(R.X,0);
P.Y.CopyTo(R.Y,0);
P.Z.CopyTo(R.Z,0);
    return;
}
```

Şekil 5.26: Jacobian koordinatlarda nokta toplama işlemini yapan fonksiyon.

```
prime.modMul(P.Z, P.Z, T1);
prime.modMul(T1, P.Z, T2);
prime.modMul(T1, Q.X, T1);
prime.modMul(T2, Q.Y, T2);
prime.modSub(T1, P.X, T1);
prime.modSub(T2, P.Y, T2);

// 7. if T1=0 then if T2 = 0 then return 2(X2:Y2:1)
int indexA = prime.length;
int indexB = prime.length;
int idz = prime.length;

while (--indexA >= 0)
    if (T1[indexA] != 0)
        break;
if (indexA < 0)
{
    while (--indexB >= 0)
        if (T2[indexB] != 0)
            break;
    if (indexB < 0)
    {
        while (--idz >= 0)
            R.Z[idz] = 0;
        Q.X.CopyTo(R.X, 0);
Q.Y.CopyTo(R.Y, 0);
        R.Z[prime.length - 1] = 0x01;
        doublePoint(R, R);
        prime.modComplete(R.X, R.X);
        prime.modComplete(R.Y, R.Y);
        prime.modComplete(R.Z, R.Z);
        return;
    }
else{
        R.info = 1;
        return;
    }
}
prime.modMul(P.Z, T1, R.Z);
prime.modMul(T1, T1, T3);
prime.modMul(T3, T1, T4);
prime.modMul(T3, P.X, T3);
prime.modMul(coef2, T3, T1);
prime.modMul(T2, T2, R.X);
prime.modSub(R.X, T1, R.X);
prime.modSub(R.X, T4, R.X);
prime.modSub(T3, R.X, T3);
prime.modMul(T3, T2, T3);
prime.modMul(T4, P.Y, T4);
prime.modSub(T3, T4, R.Y);
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
 }
```

Şekil 5.26: Devam.

- doublePoint:

Jacobian koordinatlarda Şekil 4.6 ile ifade edilen nokta çiftleme işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint(Point P, Point R)
{
if (P.info == 1)
return;
prime.modMul(P.Z, P.Z, T1);
prime.modSub(P.X, T1, T2);
prime.modAdd(P.X, T1, T1);
prime.modMul(T1, T2, T2);
prime.modMul(coef3, T2, T2);
prime.modMul(coef2, P.Y, R.Y);
prime.modMul(R.Y, P.Z, R.Z);
prime.modMul(R.Y, R.Y, R.Y);
prime.modMul(R.Y, P.X, T3);
prime.modMul(R.Y, R.Y, R.Y);
prime.modMul(R.Y, inv2, R.Y);
prime.modMul(T2, T2, R.X);
prime.modMul(coef2, T3, T1);
prime.modSub(R.X, T1, R.X);
prime.modSub(T3, R.X, T1);
prime.modMul(T1, T2, T1);
prime.modSub(T1, R.Y, R.Y);
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.27: Jacobian koordinatlarda nokta çiftleme işlemini yapan fonksiyon.

5.7. Chudnovsky Jacobian Sınıfı

Chudnovsky Jacobian eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public Chudnovsky(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
aCoef = curve.aCoef;
bCoef = curve.bCoef;
bufA = new byte[len];
bufB = new byte[len];
bufC = new byte[len];
bufD = new byte[len];
T1 = new byte[len];
T2 = new byte[len];
coef2 = curve.times2;
coef3 = curve.times3;
}
```

Şekil 5.28: Chudnovsky Jacobian parametrelerini tanımlayan fonksiyon.

- addPoint:

Chudnovsky Jacobian koordinatlarda Şekil 4.7 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(Point P, Point Q, Point R)
{
int indexA = prime.length;
while (--indexA >= 0)
    if (P.Z[indexA] != 0) break;
if (indexA < 0)
{
    R.info = Q.info;
    Q.X.CopyTo(R.X, 0);
    Q.Y.CopyTo(R.Y, 0);
    Q.Z.CopyTo(R.Z, 0);
    return;
}
```

Şekil 5.29: Chudnovsky Jacobian nokta toplama işlemini yapan fonksiyon.

```
// if(Z2=0) return P
int indexB = prime.length;
while (--indexB >= 0)
    if (Q.Z[indexB] != 0)
        break;

if (indexB < 0)
{
    R.info = P.info;
    P.X.CopyTo(R.X, 0);
    P.Y.CopyTo(R.Y, 0);
    P.Z.CopyTo(R.Z, 0);
    return;
}

prime.modMul(Q.Z, Q.Z, bufA);
prime.modMul(P.X, bufA, bufC);
prime.modMul(P.Z, P.Z, T1);
prime.modMul(T1, Q.X, R.X);
prime.modSub(R.X, bufC, bufB);
prime.modMul(P.Z, Q.Z, R.Z);
prime.modMul(bufB, R.Z, R.Z);
prime.modMul(bufA, Q.Z, bufA);
prime.modMul(P.Y, bufA, R.Y);
prime.modMul(T1, P.Z, bufD);
prime.modMul(bufD, Q.Y, bufD);
prime.modSub(bufD, R.Y, bufD);

int indexC = prime.length;
int indexD = prime.length;
while (--indexC >= 0)
    if (bufB[indexC] != 0)
        break;

if (indexC < 0) // bufB = 0
{
    while (--indexD >= 0)
        if (bufD[indexD] != 0)
            break;

    if (indexD < 0) // bufC = 0
    {
        P.X.CopyTo(R.X, 0);
        P.Y.CopyTo(R.Y, 0);
        P.Z.CopyTo(R.Z, 0);

           doublePoint(R, R);
        prime.modComplete(R.X, R.X);
        prime.modComplete(R.Y, R.Y);
        prime.modComplete(R.Z, R.Z);
        return;
    }
```

Şekil 5.29: Devam.

```
    else
    {
        // Z3 = 0, Z3^2 = 0, Z3^3=0
        // return Q
        int idz = prime.length;
        while (--idz >= 0)
            R.Z[idz] = 0;

        Q.X.CopyTo(R.X, 0);
        Q.Y.CopyTo(R.Y, 0);

        prime.modComplete(R.X, R.X);
        prime.modComplete(R.Y, R.Y);
        prime.modComplete(R.Z, R.Z);

        return;
    }
}

prime.modAdd(R.X, bufC, R.X);
prime.modMul(bufB, bufB, bufA);
prime.modMul(R.X, bufA, R.X);
prime.modMul(bufD, bufD, T1);
prime.modSub(T1, R.X, R.X);

prime.modMul(bufA, bufB, T2);
prime.modMul(R.Y, T2, T2);
prime.modMul(bufC, bufA, R.Y);
prime.modSub(R.Y, R.X, R.Y);
prime.modMul(R.Y, bufD, R.Y);
prime.modSub(R.Y, T2, R.Y);

//prime.modMul(R.Z, R.Z, bufA);
//prime.modMul(bufA, R.Z, bufB);

prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
prime.modComplete(bufA, bufA);
prime.modComplete(bufB, bufB);
}
```

Şekil 5.29: Devam.

- doublePoint:

Chudnovsky Jacobian koordinatlarda Şekil 4.8 ile ifade edilen nokta çiftleme işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint(Point P, Point R)
{
int indexA = prime.length;
while (--indexA >= 0)
if (P.Z[indexA] != 0)
        break;

if (indexA < 0)
{
    R.info = P.info;
    P.X.CopyTo(R.X, 0);
    P.Y.CopyTo(R.Y, 0);
    P.Z.CopyTo(R.Z, 0);
    return;
}
prime.modMul(P.Y, P.Z, R.Z);
prime.modMul(coef2, R.Z, R.Z);
prime.modMul(P.Y, P.Y, R.Y);
prime.modMul(P.X, R.Y, bufA);
prime.modMul(coef2, bufA, bufA);
prime.modMul(coef2, bufA, bufA);
prime.modMul(P.X, P.X, R.X);
prime.modMul(P.Z, P.Z, bufB);
prime.modMul(bufB, bufB, bufB);
prime.modMul(bufB, aCoef, bufB);
prime.modMul(coef3, R.X, R.X);
prime.modAdd(bufB, R.X, bufB);
prime.modMul(bufB, bufB, R.X);
prime.modMul(coef2, bufA, T1);
prime.modSub(R.X, T1, R.X);
prime.modSub(bufA, R.X, bufA);
prime.modMul(bufA, bufB, bufA);
prime.modMul(R.Y, R.Y, R.Y);
prime.modMul(coef2, R.Y, R.Y);
prime.modMul(coef2, R.Y, R.Y);
prime.modMul(coef2, R.Y, R.Y);
prime.modSub(bufA, R.Y, R.Y);
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.30: Chudnovsky Jacobian nokta çiftleme işlemini yapan fonksiyon.

5.8. Jacobian Quartics Sınıfı

Jacobian Quartic eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public JacobiQuartics(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
aCoef = curve.aCoef;
bCoef = curve.bCoef;
T1 = new byte[len];
T2 = new byte[len];
T3 = new byte[len];
T4 = new byte[len];
T5 = new byte[len];
T6 = new byte[len];
T7 = new byte[len];
T8 = new byte[len];
coef2 = curve.times2;
}
```

Şekil 5.31: Jacobian quartics eğri parametrelerini tanımlayan fonksiyon.

- addPoint:

Jacobian Quartic eğrilerde Şekil 4.12 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(Point P, Point Q, Point R)
{
P.X.CopyTo(T1, 0);
P.Y.CopyTo(T2, 0);
P.Z.CopyTo(T3, 0);
Q.X.CopyTo(T4, 0);
Q.Y.CopyTo(T5, 0);
Q.Z.CopyTo(T6, 0);
prime.modMul(T1, T3, T7);
prime.modAdd(T2, T7, T7);
prime.modMul(T4, T6, T8);
prime.modAdd(T5, T8, T8);
prime.modMul(T2, T5, T2);
```

Şekil 5.32: Jacobian quartics eğrisinde nokta toplama işlemini yapan fonksiyon.

```
prime.modMul(T7, T8, T7);
prime.modSub(T7, T2, T7);
prime.modMul(T1, T4, T5);
prime.modAdd(T1, T3, T1);
prime.modMul(T3, T6, T8);
prime.modAdd(T4, T6, T4);
prime.modMul(T5, T8, T6);
prime.modSub(T7, T6, T7);
prime.modMul(T1, T4, T1);
prime.modSub(T1, T5, T1);
prime.modSub(T1, T8, T1);
prime.modMul(T1, T1, T3);
prime.modMul(coef2, T6, T6);
prime.modSub(T3, T6, T3);
prime.modMul(T3, T6, T3);
prime.modMul(aCoef, T6, T4);
prime.modAdd(T2, T4, T2);
prime.modMul(T8, T8, T4);
prime.modMul(T5, T5, T8);
prime.modAdd(T4, T8, T5);
prime.modMul(T2, T5, T2);
prime.modAdd(T2, T3, T2);
prime.modSub(T4, T8, T5);

T7.CopyTo(R.X, 0);
T2.CopyTo(R.Y, 0);
T5.CopyTo(R.Z, 0);

prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.32: Devam.

- doublePoint:

Jacobian Quartic eğrilerde Şekil 4.13 ile ifade edilen nokta çiftleme işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint(Point P, Point R)
{
prime.modMul(P.X, P.X, T1);
prime.modMul(T1, T1, T2);
prime.modMul(P.Y, P.Y, T3);
prime.modMul(P.Z, P.Z, T4);
prime.modMul(T4, T4, T5);
prime.modAdd(T1, T4, T6);
prime.modAdd(P.X, P.Z, T7);
prime.modMul(T7, T7, T7);
prime.modSub(T7, T6, T7);
prime.modMul(T7, T7, T8);
prime.modAdd(P.Y, T7, R.X);
prime.modMul(R.X, R.X, R.X);
prime.modSub(R.X, T3, R.X);
prime.modSub(R.X, T8, R.X); // X3
prime.modMul(aCoef, T8, T1);
prime.modMul(coef2, T1, T1);
prime.modMul(coef2, T3, T3);
prime.modMul(coef2, T3, T3);
prime.modAdd(T1, T3, R.Y);
prime.modAdd(T5, T2, T1);
prime.modMul(T1, R.Y, R.Y);
prime.modMul(T8, T8, T3);
prime.modAdd(R.Y, T3, R.Y); // Y3
prime.modSub(T5, T2, R.Z);
prime.modMul(coef2, R.Z, R.Z); // Z3

prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.33: Jacobian quartics eğrisinde nokta çiftleme işlemini yapan fonksiyon.

5.9. Extended Jacobi Quartic Sınıfı

Extended Jacobian Quartic eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public ExtendedJacobiQuartic(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
inv2 = curve.inv2;
dCoef = curve.bCoef;
t1 = new byte[len];
t2 = new byte[len];
t3 = new byte[len];
coef2 = curve.times2;
}
```

Şekil 5.34: Extended Jacobian quartics eğri parametrelerini tanımlayan fonksiyon.

- addPoint:

Extended Jacobian Quartic eğrilerde Şekil 4.15 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(Point P, Point Q, Point T, Point R)
{
byte[] T3 = new byte[P.Z.Length];
prime.modAdd(T.X, P.Z, t1);
prime.modMul(dCoef, T.Y, t2);
prime.modAdd(t2, Q.Z, t2);
     prime.modMul(t1, t2, t1);
     prime.modMul(P.Z, T.Y, t2);
     prime.modMul(T.X, Q.Z, T3);
     prime.modSub(t1, t3, t1);
     prime.modMul(dCoef, t2, t3);
     prime.modSub(t1, t3, t1);
     prime.modAdd(T3, t2, t3);
     prime.modSub(T3, t2, T3);
     prime.modSub(P.X, P.Y, R.Z);
```

Şekil 5.35: Extended Jacobian quartics eğrisinde nokta çiftleme işlemini yapan fonksiyon.

```
      prime.modAdd(Q.Z, Q.Y, t2);
      prime.modMul(R.Z, t2, R.Z);
prime.modMul(P.X, Q.Z, t2);
      prime.modMul(P.Y, Q.Y, R.Y);
      prime.modSub(R.Z, t2, R.Z);
      prime.modAdd(R.Z, R.Y, R.Z);
      prime.modAdd(R.Y, t1, R.Y);
      prime.modMul(coef2, t2, t1);
      prime.modSub(t3, t1, t1);
      prime.modAdd(R.Y, t2, R.Y);
      prime.modMul(t1, R.Y, R.Y);
      prime.modAdd(R.Z, T3, R.X);
      prime.modMul(R.X, R.X, R.X);
      prime.modMul(R.Z, R.Z, R.Z);
      prime.modSub(R.Y, R.Z, R.Y);
      prime.modSub(R.X, R.Z, R.X);
      prime.modMul(T3, T3, T3);
      prime.modSub(R.X, T3, R.X);
      prime.modMul(inv2, R.X, R.X);
      prime.modComplete(R.X, R.X);
      prime.modComplete(R.Y, R.Y);
      prime.modComplete(T3, T3);
      prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.35: Devam

- doublePoint:

Extended Jacobian Quartic eğrilerde Şekil 4.17 ile ifade edilen nokta çiftleme işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint(Point P, Point R)
{
byte[] T3 = new byte[P.Z.Length];

prime.modAdd(P.X, P.Y, T3);
prime.modMul(P.X, P.X, R.X);
prime.modMul(P.Y, P.Y, R.Y);
prime.modMul(P.Z, P.Z, R.Z);
prime.modMul(T3, T3, T3);
prime.modAdd(R.X, R.Y, R.X);
```

Şekil 5.36: Extended Jacobian quartics eğrisinde nokta çiftleme işlemini yapan fonksiyon.

```
prime.modSub(T3, R.X, T3);
prime.modMul(coef2, R.Z, R.Z);
prime.modSub(R.Z, R.X, R.Z);
prime.modAdd(T3, R.Z, R.X);
prime.modMul(T3, T3, T3);
prime.modMul(R.Z, R.Z, R.Z);
prime.modMul(R.X, R.X, R.X);
prime.modSub(R.X, T3, R.X);
prime.modSub(R.X, R.Z, R.X);
prime.modMul(coef2, R.Z, R.Z);
prime.modMul(coef2, R.Y, R.Y);
prime.modMul(R.Y, R.Y, R.Y);
prime.modAdd(R.Y, T3, R.Y);
prime.modSub(R.Y, R.Z, R.Y);
prime.modMul(coef2, T3, T3);

prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(T3, T3);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.36: Devam.

5.10. Jacobi Intersection Sınıfı

Jacobian Intersection eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public JacobiIntersection(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
aCoef = curve.aCoef;
bCoef = curve.bCoef;
bufA = new byte[len];
bufB = new byte[len];
bufC = new byte[len];
bufD = new byte[len];
bufE = new byte[len];
bufF = new byte[len];
bufG = new byte[len];
bufH = new byte[len];
bufJ = new byte[len];
bufK = new byte[len];
coef2 = curve.times2;
}
```

Şekil 5.37: Jacobi intersection eğri parametrelerini tanımlayan fonksiyon.

- addPoint:

Jacobian Intersection eğrilerde Şekil 4.18 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(Point P, Point Q, Point R)
{
byte[] T1 = new byte[P.Z.Length]; T1 = P.Z;
byte[] T2 = new byte[Q.Z.Length]; T2 = Q.Z;
byte[] T3 = new byte[R.Z.Length];

prime.modMul(P.X, P.Y, bufA);
prime.modMul(T1, P.Z, bufB);
prime.modMul(Q.X, Q.Y, bufC);
prime.modMul(T2, Q.Z, bufD);
prime.modMul(P.X, T2, bufE);
prime.modMul(P.Y, Q.Z, bufF);
prime.modMul(T1, Q.X, bufG);
prime.modMul(P.Z, Q.Y, bufH);
prime.modMul(bufA, bufD, bufJ);
prime.modMul(bufB, bufC, bufK);
prime.modAdd(bufH, bufF, R.X);
prime.modAdd(bufE, bufG, R.Y);
prime.modMul(R.X, R.Y, R.X);
prime.modSub(R.X, bufJ, R.X);
prime.modSub(R.X, bufK, R.X); // X3
prime.modAdd(bufH, bufE, R.Y);
prime.modSub(bufF, bufG, R.Z);
prime.modMul(R.Y, R.Z, R.Y);
prime.modSub(R.Y, bufJ, R.Y);
prime.modAdd(R.Y, bufK, R.Y); // Y3
prime.modMul(bCoef, bufA, bufA);
prime.modSub(bufB, bufA, T3);
prime.modAdd(bufC, bufD, bufC);
prime.modMul(T3, bufC, T3);
prime.modMul(bCoef, bufJ, bufJ);
prime.modAdd(T3, bufJ, bufJ);
prime.modSub(bufJ, bufK, T3); // T3
prime.modMul(bufG, bufG, bufG);
prime.modMul(bufH, bufH, bufH);
prime.modAdd(bufH, bufG, R.Z); // Z3
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(T3, T3);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.38: Jacobi intersection eğrisinde nokta toplama işlemini yapan fonksiyon.

- doublePoint:

Jacobian Intersection eğrilerde Şekil 4.20 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint(Point P, Point R)
{
byte[] T1 = new byte[P.Z.Length]; T1 = P.Z;
byte[] T3 = new byte[R.Z.Length];

prime.modMul(P.X, P.Y, bufA);
prime.modMul(T1, P.Z, bufB);
prime.modMul(bufA, bufA, bufC);
prime.modMul(bufB, bufB, bufD);
prime.modMul(bufA, bufB, R.X);
prime.modMul(coef2, R.X, R.X); // X3
prime.modMul(P.Y, P.Y, bufE);
prime.modMul(bufE, bufE, bufE); // Y^4
prime.modAdd(bufC, bufE, R.Y);
prime.modAdd(R.Y, R.Y, R.Y);
prime.modMul(bCoef, bufC, bufC);
prime.modSub(R.Y, bufC, R.Y);
prime.modSub(R.Y, bufB, R.Y);
prime.modSub(bufD, bufC, T3); // T3
prime.modAdd(bufD, bufC, R.Z); // Z3

prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(T3, T3);
prime.modComplete(R.Z, R.Z);
 }
```

Şekil 5.39: Jacobi intersection eğrisinde nokta çiftleme işlemini yapan fonksiyon.

5.11. Twisted Jacobi Intersection Sınıfı

Twisted Jacobian Intersection eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public TwistedJacobiIntersections(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
aCoef = curve.aCoef;
bCoef = curve.bCoef;
bufA = new byte[len];
bufB = new byte[len];
bufC = new byte[len];
bufD = new byte[len];
bufE = new byte[len];
bufF = new byte[len];
bufG = new byte[len];
bufH = new byte[len];
bufJ = new byte[len];
bufK = new byte[len];
coef2 = curve.times2;
}
```

Şekil 5.40: Twisted Jacobi intersection eğri parametrelerini tanımlayan fonksiyon.

- addPoint:

Twisted Jacobian intersection eğrilerde Şekil 4.22 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(Point P, Point Q, Point R)
{
byte[] T1 = new byte[P.Z.Length]; T1 = P.Z;
byte[] T2 = new byte[Q.Z.Length]; T2 = Q.Z;
byte[] T3 = new byte[R.Z.Length];
prime.modMul(P.X, P.Y, bufA);
prime.modMul(T1, P.Z, bufB);
prime.modMul(Q.X, Q.Y, bufC);
prime.modMul(T2, Q.Z, bufD);
prime.modMul(P.X, T2, bufE);
prime.modMul(P.Y, Q.Z, bufF);
prime.modMul(T1, Q.X, bufG);
prime.modMul(P.Z, Q.Y, bufH);
prime.modMul(bufA, bufD, bufJ);
prime.modMul(bufB, bufC, bufK);
prime.modAdd(bufH, bufF, R.X);
prime.modAdd(bufE, bufG, R.Y);
prime.modMul(R.X, R.Y, R.X);
prime.modSub(R.X, bufJ, R.X);
prime.modSub(R.X, bufK, R.X); // X3
prime.modAdd(bufH, bufE, R.Y);
prime.modMul(aCoef, bufG, R.Z);
prime.modSub(bufF, R.Z, R.Z);
prime.modMul(R.Y, R.Z, R.Y);
prime.modSub(R.Y, bufJ, R.Y);
prime.modMul(aCoef, bufK, R.Z);
prime.modAdd(R.Y, R.Z, R.Y); // Y3
prime.modMul(bCoef, bufA, bufA);
prime.modSub(bufB, bufA, T3);
prime.modAdd(bufC, bufD, bufC);
prime.modMul(T3, bufC, T3);
prime.modMul(bCoef, bufJ, bufJ);
prime.modAdd(T3, bufJ, bufJ);
prime.modSub(bufJ, bufK, T3); // T3
prime.modMul(bufG, bufG, bufG);
prime.modMul(aCoef, bufG, bufG);
prime.modMul(bufH, bufH, bufH);
prime.modAdd(bufH, bufG, R.Z); // Z3
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(T3, T3);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.41: Twisted Jacobi intersection eğrisinde nokta toplama işlemini yapan fonksiyon.

- doublePoint:

Twisted Jacobian Intersection eğrilerde Şekil 4.23 ile ifade edilen nokta çiftleme işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint(Point P, Point R)
{
byte[] T1 = new byte[P.Z.Length]; T1 = P.Z;
byte[] T3 = new byte[R.Z.Length];

prime.modMul(P.Y, P.Z, bufA);
prime.modMul(bufA, bufA, bufB);
prime.modMul(P.X, T1, bufC);
prime.modMul(bufC, bufC, bufD);
prime.modMul(P.Z, T1, bufE);
prime.modMul(bufE, bufE, bufE);
prime.modMul(coef2, bufE, bufE);
prime.modAdd(bufA, bufC, bufC);
prime.modMul(bufC, bufC, bufC);
prime.modSub(bufC, bufB, R.X);
prime.modSub(R.X, bufD, R.X); // X3
prime.modMul(aCoef, bufD, bufC); //bufC = a*D
prime.modSub(bufB, bufC, R.Y); // Y3
prime.modSub(bufE, bufC, T3);
prime.modSub(T3, bufB, T3); // T3
prime.modAdd(bufB, bufC, R.Z); // Z3

prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(T3, T3);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.42: Twisted Jacobi intersection eğrisinde nokta çiftleme işlemini yapan fonksiyon.

5.12. Edwards Sınıfı

Edwards eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public EdwardsCurve(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
dCoef = curve.aCoef;
cCoef = curve.bCoef;
bufA = new byte[len];
bufB = new byte[len];
bufC = new byte[len];
bufD = new byte[len];
bufE = new byte[len];
bufF = new byte[len];
bufG = new byte[len];
bufH = new byte[len];
coef2 = curve.times2;
T1 = new byte[len];
T2 = new byte[len];
}
```

Şekil 5.43: Edwards eğri parametrelerini tanımlayan fonksiyon.

- addPoint:

Edwards eğrilerde Şekil 4.27 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(Point P, Point Q, Point R)
{
prime.modMul(P.Z, Q.Z, bufA);
prime.modMul(bufA, bufA, bufB);
prime.modMul(P.X, Q.X, bufC);
prime.modMul(P.Y, Q.Y, bufD);
prime.modMul(bufC, bufD, bufE);
prime.modMul(bufE, dCoef, bufE);
prime.modSub(bufB, bufE, bufF);
```

Şekil 5.44: Edwards eğrisinde nokta toplama işlemini yapan fonksiyon.

```
prime.modAdd(bufB, bufE, bufG);
prime.modAdd(P.X, P.Y, T1);
prime.modAdd(Q.X, Q.Y, T2);
prime.modMul(T1, T2, R.X);
prime.modSub(R.X, bufC, R.X);
prime.modSub(R.X, bufD, R.X);
prime.modMul(R.X, bufA, R.X);
prime.modMul(R.X, bufF, R.X);
prime.modSub(bufD, bufC, R.Y);
prime.modMul(R.Y, bufA, R.Y);
prime.modMul(R.Y, bufG, R.Y);
prime.modMul(bufF, bufG, R.Z);
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.44: Devam.

- doublePoint:

Edwards eğrilerde Şekil 4.29 ile ifade edilen nokta çiftleme işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint2(Point P, Point R)
{
prime.modAdd(P.X, P.Y, bufB);
prime.modMul(bufB, bufB, bufB);
prime.modMul(P.X, P.X, bufC);
prime.modMul(P.Y, P.Y, bufD);
prime.modAdd(bufC, bufD, bufE);
prime.modMul(P.Z, P.Z, bufH);
prime.modMul(coef2, bufH, bufH);
prime.modSub(bufE, bufH, bufH);
prime.modSub(bufB, bufE, bufA);
prime.modMul(bufA, bufH, R.X);
prime.modSub(bufC, bufD, bufA);
prime.modMul(bufA, bufE, R.Y);
prime.modMul(bufE, bufH, R.Z);
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.45: Edwards eğrisinde nokta çiftleme işlemini yapan fonksiyon.

5.13. Inverted Twisted Edwards Sınıfı

Inverted Twisted Edwards eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public InvertedTwistedErdwardsCurve(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
aCoef = curve.bCoef;
dCoef = curve.bCoef;
cCoef = curve.bCoef;
bufA = new byte[len];
bufB = new byte[len];
bufC = new byte[len];
bufD = new byte[len];
bufE = new byte[len];
bufU = new byte[len];
bufH = new byte[len];
bufI = new byte[len];
bufT1 = new byte[len];
bufT2 = new byte[len];
coef2 = curve.times2;
}
```

Şekil 5.46: Inverted twisted Edwards eğri parametrelerini tanımlayan fonksiyon.

- addPoint:

Inverted Twisted Edwards eğrilerde Şekil 4.34 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(Point P, Point Q, Point R)
{
prime.modMul(P.Z, Q.Z, bufA);
prime.modMul(bufA, bufA, bufB);
prime.modMul(dCoef, bufB, bufB);
prime.modMul(P.X, Q.X, bufC);
prime.modMul(P.Y, Q.Y, bufD);
prime.modMul(bufC, bufD, bufE);
prime.modMul(aCoef, bufD, bufH);
prime.modSub(bufC, bufH, bufH);
prime.modAdd(P.X, P.Y, bufT1);
prime.modAdd(Q.X, Q.Y, bufT2);
prime.modMul(bufT1, bufT2, bufI);
prime.modSub(bufI, bufC, bufI);
prime.modSub(bufI, bufD, bufI);
prime.modAdd(bufE, bufB, R.X);
prime.modMul(cCoef, R.X, R.X);
prime.modMul(R.X, bufH, R.X); // X3
prime.modSub(bufE, bufB, R.Y);
prime.modMul(cCoef, R.Y, R.Y);
prime.modMul(R.Y, bufI, R.Y); // Y3
prime.modMul(bufA, bufH, R.Z);
prime.modMul(R.Z, bufI, R.Z);

prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.47: Inverted twisted Edwards eğrisinde nokta toplama işlemini yapan fonksiyon.

- doublePoint:

Inverted Twisted Edwards eğrilerde Şekil 4.36 ile ifade edilen nokta çiftleme işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint(Point P, Point R)
{
prime.modMul(P.X, P.X, bufA);
 prime.modMul(P.Y, P.Y, bufB);
     prime.modMul(aCoef, bufB, bufU);
     prime.modAdd(bufA, bufU, bufC);
     prime.modSub(bufA, bufU, bufD);
     prime.modAdd(P.X, P.Y, bufE);
     prime.modMul(bufE, bufE, bufE);
    prime.modSub(bufE, bufA, bufE);
     prime.modSub(bufE, bufB, bufE);
     prime.modMul(bufC, bufD, R.X);
     prime.modMul(P.Z, P.Z, R.Y);
     prime.modMul(dCoef, bufA, R.Y);
     prime.modMul(coef2, R.Y, R.Y);
     prime.modSub(bufC, R.Y, R.Y);
     prime.modMul(bufE, R.Y, R.Y);
     prime.modMul(bufD, bufE, R.Z);

     prime.modComplete(R.X, R.X);
     prime.modComplete(R.Y, R.Y);
     prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.48: Inverted twisted Edwards eğrisinde nokta çiftleme işlemini yapan fonksiyon.

5.14. Huff Sınıfı

Huff eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public HuffCurve(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
aCoef = curve.aCoef;
bCoef = curve.bCoef;
bInv = curve.invb;
bufA = new byte[len];
bufB = new byte[len];
bufC = new byte[len];
bufD = new byte[len];
bufE = new byte[len];
bufF = new byte[len];
bufG = new byte[len];
bufH = new byte[len];
bufT1 = new byte[len];
bufT2 = new byte[len];
bufT3 = new byte[len];
bufT4 = new byte[len];
}
```

Şekil 5.49: Huff eğri parametrelerini tanımlayan fonksiyon.

- addPoint:

Huff eğrilerde Şekil 4.39 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(Point P, Point Q, Point R)
{
prime.modMul(P.X, Q.X, bufA);
prime.modMul(P.Y, Q.Y, bufB);
prime.modMul(P.Z, Q.Z, bufD);
prime.modMul(bCoef, bufA, bufE);
prime.modMul(aCoef, bufB, bufF);
prime.modAdd(P.X, P.Z, bufT1);
prime.modAdd(Q.X, Q.Z, bufT2);
prime.modMul(bufT1, bufT2, bufG);
prime.modSub(bufG, bufA, bufG);
prime.modSub(bufG, bufD, bufG);
prime.modAdd(P.Y, P.Z, bufT3);
prime.modAdd(Q.Y, Q.Z, bufT4);
prime.modMul(bufT3, bufT4, bufH);
prime.modSub(bufH, bufB, bufH);
prime.modSub(bufH, bufD, bufH);
prime.modAdd(bufE, bufD, bufT2);
prime.modSub(bufE, bufD, bufT3);
prime.modSub(bufD, bufF, bufT4);
prime.modMul(bInv, bufT2, R.X);
prime.modMul(R.X, bufT3, R.X);
prime.modMul(R.X, bufT4, R.X);
prime.modMul(bufG, bufH, R.Y);
prime.modMul(R.Y, bufT2, R.Y);
prime.modAdd(bufF, bufD, bufT2);
prime.modMul(bufG, bufT2, R.Z);
prime.modMul(R.Z, bufT3, R.Z);
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.50: Huff eğrisinde nokta toplama işlemini yapan fonksiyon.

- doublePoint:

Huff eğrilerde Şekil 4.40 ile ifade edilen nokta çiftleme işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint(Point P, Point R)
{
prime.modMul(P.X, P.X, bufA);
prime.modMul(P.Y, P.Y, bufB);
prime.modMul(P.Z, P.Z, bufC);
prime.modMul(bCoef, bufA, bufD);
prime.modMul(aCoef, bufB, bufE);
prime.modAdd(P.X, P.Z, bufT1);
prime.modMul(bufT1, bufT1, bufF);
prime.modSub(bufF, bufA, bufF);
prime.modSub(bufF, bufC, bufF);
prime.modAdd(P.Y, P.Z, bufT2);
prime.modMul(bufT2, bufT2, bufG);
prime.modSub(bufG, bufB, bufG);
prime.modSub(bufG, bufC, bufG);
prime.modSub(bufD, bufC, bufT1);
prime.modAdd(bufD, bufC, bufT2);
prime.modSub(bufC, bufE, bufA);
prime.modMul(bufT1, bufT2, R.X);
prime.modMul(R.X, bufA, R.X);
prime.modMul(bufF, bufG, R.Y);
prime.modMul(R.Y, bufT2, R.Y);
prime.modAdd(bufC, bufE, R.Z);
prime.modMul(R.Z, bufF, R.Z);
prime.modMul(R.Z, bufT1, R.Z);
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.51: Huff eğrisinde nokta çiftleme işlemini yapan fonksiyon.

5.15. Lopez Dahap Sınıfı

Lopez Dahap eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public LopezDahap(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
aCoef = curve.aCoef;
bCoef = curve.bCoef;
bufA = new byte[len];
bufB = new byte[len];
bufC = new byte[len];
bufD = new byte[len];
bufE = new byte[len];
bufF = new byte[len];
bufG = new byte[len];
bufH = new byte[len];
bufI = new byte[len];
bufJ = new byte[len];
bufK = new byte[len];
}
```

Şekil 5.52: Lopez Dahab eğri parametrelerini tanımlayan fonksiyon.

- addPoint:

Lopez Dahap eğrilerinde Şekil 4.41 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(Point P, Point Q, Point R)
{
prime.modMul(P.X, Q.Z, bufA);
prime.modMul(Q.X, P.Z, bufB);
prime.modMul(bufA, bufA, bufC);
prime.modMul(bufB, bufB, bufD);
prime.modAdd(bufA, bufB, bufE);
prime.modAdd(bufC, bufD, bufF);
prime.modMul(Q.Z, Q.Z, bufG);
prime.modMul(P.Y, bufG, bufG);
prime.modMul(P.Z, P.Z, bufH);
prime.modMul(Q.Y, bufH, bufH);
prime.modAdd(bufG, bufH, bufI);
prime.modMul(bufI, bufE, bufJ);
prime.modMul(P.Z, Q.Z, R.Z);
prime.modMul(bufF, R.Z, R.Z); // Z3
prime.modAdd(bufC, bufG, R.X);
prime.modMul(bufB, R.X, bufB);
prime.modAdd(bufH, bufD, R.X);
prime.modMul(bufA, R.X, R.X);
prime.modAdd(R.X, bufB, R.X); // X3
prime.modAdd(bufJ, R.Z, R.Y);
prime.modMul(R.Y, R.X, R.Y);
prime.modMul(bufA, bufJ, bufB);
prime.modMul(bufF, bufG, bufC);
prime.modAdd(bufB, bufC, bufB);
prime.modMul(bufB, bufF, bufB);
prime.modAdd(bufB, R.Y, R.Y); // Y3
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.53: Lopez Dahab eğrisinde nokta toplama işlemini yapan fonksiyon.

- doublePoint:

Lopez Dahap eğrilerinde Şekil 4.42 ile ifade edilen nokta çiftleme işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint(Point P, Point R)
{
prime.modMul(P.X, P.Z, bufA);
prime.modMul(P.X, P.X, bufB);
prime.modAdd(bufB, P.Y, bufC);
prime.modMul(bufA, bufC, bufD);
prime.modMul(bufA, bufA, R.Z);
prime.modMul(bufC, bufC, R.X); //C^2
prime.modMul(aCoef, aCoef, bufA); //a^2
prime.modMul(bufA, R.Z, bufA);  //a^2*Z3
prime.modAdd(R.X, bufA, R.Y);
prime.modAdd(R.Y, bufD, R.X);
prime.modAdd(R.Z, bufD, R.Y);
prime.modMul(R.Y, R.X, R.Y);
prime.modMul(bufB, bufB, bufB);
prime.modMul(bufB, R.Z, bufB);
prime.modAdd(R.Y, bufB, R.Y);
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.54: Lopez Dahab eğrisinde nokta çiftleme işlemini yapan fonksiyon.

5.16. Hessian Sınıfı

Hessian eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public HessianCurve(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
aCoef = curve.aCoef;
bCoef = curve.bCoef;
T1 = new byte[len];
T2 = new byte[len];
T3 = new byte[len];
T4 = new byte[len];
T5 = new byte[len];
T6 = new byte[len];
T7 = new byte[len];
}
```

Şekil 5.55: Hessian eğri parametrelerini tanımlayan fonksiyon.

- addPoint:

Hessian eğrilerde Şekil 4.46 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(Point P, Point Q, Point R)
{
P.X.CopyTo(T1,0);
P.Y.CopyTo(T2,0);
P.Z.CopyTo(T3,0);
Q.X.CopyTo(T4,0);
Q.Y.CopyTo(T5,0);
Q.Z.CopyTo(T6,0);
     prime.modMul(T1, T6, T7);
prime.modMul(T1, T5, T1);
prime.modMul(T3, T5, T5);
prime.modMul(T3, T4, T3);
prime.modMul(T2, T4, T4);
prime.modMul(T2, T6, T2);
prime.modMul(T2, T7, T6);
prime.modMul(T2, T4, T2);
prime.modMul(T3, T4, T4);
prime.modMul(T3, T5, T3);
prime.modMul(T1, T5, T5);
prime.modMul(T1, T7, T1);
prime.modSub(T1, T4, T1);
prime.modSub(T2, T5, T2);
prime.modSub(T3, T6, T3);
     T2.CopyTo(R.X,0);
T1.CopyTo(R.Y,0);
T3.CopyTo(R.Z,0);

prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.56: Hessian eğrisinde nokta toplama işlemini yapan fonksiyon.

- doublePoint:

Hessian eğrilerde Şekil 4.48 ile ifade edilen nokta çiftleme işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint(Point P, Point R)
{
prime.modMul(P.X, P.X, T1);
prime.modAdd(P.X, P.Y, T2);
prime.modMul(P.Y, T2, T2);
prime.modAdd(P.Z, P.X, T3);
prime.modMul(P.Z, T3, T3);
prime.modAdd(T1, T3, T3);
prime.modAdd(T1, T2, T2);
prime.modSub(P.X, P.Y, T1);
prime.modMul(T2, T1, T1);
prime.modMul(T1, P.Z, R.Z);
prime.modSub(P.Z, P.X, T2);
prime.modMul(T3, T2, T2);
prime.modMul(T2, P.Y, R.X);
prime.modAdd(T1, T2, T3);
     int i = prime.length;
while (--i >= 0)
R.Y[i] = 0x00;
     prime.modSub(R.Y, T3, T3);
prime.modMul(T3, P.X, R.Y);
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.57: Hessian eğrisinde nokta çiftleme işlemini yapan fonksiyon.

5.17. Twisted Hessian Sınıfı

Twisted Hessian eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public TwistedHessianCurve(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
aCoef = curve.aCoef;
bCoef = curve.bCoef;
bufA = new byte[len];
bufB = new byte[len];
bufC = new byte[len];
bufD = new byte[len];
bufE = new byte[len];
bufF = new byte[len];
bufG = new byte[len];
bufT = new byte[len];
}
```

Şekil 5.58: Twisted Hessian eğri parametrelerini tanımlayan fonksiyon.

- addPoint:

Twisted Hessian eğrilerde Şekil 4.51 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(Point P, Point Q, Point R)
{
prime.modMul(P.X, Q.Z, bufA);
prime.modMul(P.Z, Q.Z, bufB);
prime.modMul(P.Y, Q.X, bufC);
prime.modMul(P.Y, Q.Y, bufD);
prime.modMul(P.Z, Q.Y, bufE);
prime.modMul(P.X, Q.X, bufF);
prime.modMul(aCoef, bufF, bufF);
prime.modMul(bufA, bufB, bufG);
prime.modMul(bufC, bufD, bufT);
prime.modSub(bufG, bufT, R.X); // X3
prime.modMul(bufD, bufE, bufG);
prime.modMul(bufF, bufA, bufT);
prime.modSub(bufG, bufT, R.Y); // Y3
prime.modMul(bufF, bufC, bufG);
prime.modMul(bufB, bufE, bufT);
prime.modSub(bufG, bufT, R.Z); // Z3
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.59: Twisted Hessian eğrisinde nokta toplama işlemini yapan fonksiyon.

- doublePoint:

Twisted Hessian eğrilerde Şekil 4.53 ile ifade edilen nokta çiftleme işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint(Point P, Point R)
{
prime.modMul(P.X, P.X, bufA);
prime.modMul(P.Y, P.Y, bufB);
prime.modMul(P.Z, P.Z, bufC);
prime.modMul(bufA, P.X, bufD);
prime.modMul(bufB, P.Y, bufE);
prime.modMul(bufC, P.Z, bufF);
prime.modMul(aCoef, bufD, bufG);
prime.modSub(bufE, bufF, R.X);
prime.modMul(P.X, R.X, R.X);    // X3
prime.modSub(bufG, bufE, R.Y);
prime.modMul(P.Z, R.Y, R.Y);    // Y3
prime.modSub(bufF, bufG, R.Z);
prime.modMul(P.Y, R.Z, R.Z);    // Z3
     prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.60: Twisted Hessian eğrisinde nokta çiftleme işlemini yapan fonksiyon.

5.18. Twisted Edwards Sınıfı

Twisted Edwards eğrisini tanımlayan sınıftır. Parametre olarak CurveP nesnesini alır.

```
public TwistedEdwardsCurve(CurveP curve)
{
int len = curve.length;
prime = new modInt(curve.prime);
aCoef = curve.aCoef;
dCoef = curve.aCoef;
bufA = new byte[len];
bufB = new byte[len];
bufC = new byte[len];
bufD = new byte[len];
bufE = new byte[len];
bufF = new byte[len];
bufG = new byte[len];
bufH = new byte[len];
bufJ = new byte[len];
coef2 = curve.times2;
}
```

Şekil 5.61: Twisted Edwards eğri parametrelerini tanımlayan fonksiyon.

- addPoint:

Twisted Edwards eğrilerde Şekil 4.31 ile ifade edilen nokta toplama işlemini gerçekleştiren fonksiyondur.

```
public void addPoint(Point P, Point Q, Point R)
{
prime.modMul(P.Z, Q.Z, bufA);
prime.modMul(bufA, bufA, bufB);
prime.modMul(P.X, Q.X, bufC);
prime.modMul(P.Y, Q.Y, bufD);
prime.modMul(bufC, bufD, bufE);
prime.modMul(bufE, dCoef, bufE);
prime.modSub(bufB, bufE, bufF);
prime.modAdd(bufB, bufE, bufG);
prime.modAdd(P.X, P.Y, bufB);
prime.modAdd(Q.X, Q.Y, bufE);
prime.modMul(bufB, bufE, R.X);
prime.modSub(R.X, bufC, R.X);
prime.modSub(R.X, bufD, R.X);
prime.modMul(R.X, bufA, R.X);
prime.modMul(R.X, bufF, R.X); // X3
prime.modMul(aCoef, bufC, R.Y);
prime.modSub(bufD, R.Y, R.Y);
prime.modMul(R.Y, bufG, R.Y);
prime.modMul(R.Y, bufA, R.Y); // Y3
prime.modMul(bufF, bufG, R.Z);// Z3
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.62: Twisted Edwards eğrisinde nokta toplama işlemini yapan fonksiyon.

- doublePoint:

Twisted Edwards eğrilerde Şekil 4.32 ile ifade edilen nokta çiftleme işlemini gerçekleştiren fonksiyondur.

```
public void doublePoint(Point P, Point R)
{
prime.modAdd(P.X, P.Y, bufA);
prime.modMul(bufA, bufA, bufB);
prime.modMul(P.X, P.X, bufC);
prime.modMul(P.Y, P.Y, bufD);
prime.modMul(aCoef, bufC, bufE);
prime.modAdd(bufE, bufD, bufF);
prime.modMul(P.Z, P.Z, bufH);
prime.modMul(coef2, bufH, bufJ);
prime.modSub(bufF, bufJ, bufJ);
prime.modSub(bufB, bufC, R.X);
prime.modSub(R.X, bufD, R.X);
prime.modMul(R.X, bufJ, R.X); // X3
prime.modSub(bufE, bufD, R.Y);
prime.modMul(R.Y, bufF, R.Y); // Y3
prime.modMul(bufF, bufJ, R.Z);// Z3
prime.modComplete(R.X, R.X);
prime.modComplete(R.Y, R.Y);
prime.modComplete(R.Z, R.Z);
}
```

Şekil 5.63: Twisted Edwards eğrisinde nokta çiftleme işlemini yapan fonksiyon.

5.18. Algoritma Doğruluk Testi

Bu bölümde önceki bölümlerde c# sınıfları ile tanımlanmış olan eliptik eğri algoritmalarının, NIST test vektörleri ile testi yapılmıştır. Aşağıda örnek olarak afin koordinat sisteminde nokta toplama ve nokta çiftleme işlemleri için test yazılımı gösterilmiştir.

```
WeierstrassCurve curve = new WeierstrassCurve(curve192);
Point w_pointP = new Point(curve192, Px, Py, Pz);
Point w_pointQ = new Point(curve192, Qx, Qy, Qz);
Point w_pointR = new Point(curve192, Rx, Ry, Rz);
```

Şekil 5.64: Weierstrass eğri tanımı.

Yukarıdaki gibi eğri tanımı yapıldıktan sonra nokta toplama fonksiyonu çağrılarak, projektif koordinatlardan afin koordinat sistemine dönüşüm yapılıp NIST test vektörleri ile doğruluğu "if" bloğu içerisinde kontrol edilmiştir.

```
curve.addPoint(w_pointP, w_pointQ, w_pointR);   // add R = P
string w_add_pointRx = utility.ByteArrayToHexString(
  utility.projectiveToAffineX(w_pointR, prime));
string w_add_pointRy = utility.ByteArrayToHexString(
  utility.projectiveToAffineY(w_pointR, prime));
if (w_add_pointRx.ToUpper() ==
    NISTConstants.P192_S_PLUS_T_X.ToUpper().Replace(" ", "")
    && w_add_pointRy.ToUpper() ==
   NISTConstants.P192_S_PLUS_T_Y.ToUpper().Replace(" ", ""))
     {
   Console.WriteLine("Weierstrass ADD POINT: P + Q = R is
correct.");
}
```

Şekil 5.65: NIST vektörleri ile nokta toplama testi.

Aynı şekilde nokta çiftleme fonksiyonu çağrılarak, projektif koordinatlardan afin koordinat sistemine dönüşüp yapılıp NIST test vektörleri ile doğruluğu "if" bloğu içerisinde kontrol edilmektedir.

```
curve.doublePoint(w_pointP, w_pointR);    // double R = 2P
     string w_double_pointRx = utility.ByteArrayToHexString(
       utility.projectiveToAffineX(w_pointR, prime));
     string w_double_pointRy = utility.ByteArrayToHexString(
       utility.projectiveToAffineY(w_pointR, prime));
     if (w_double_pointRx.ToUpper() ==
     NISTConstants.P192_2S_X.ToUpper().Replace(" ", "")
     && w_double_pointRy.ToUpper() ==
     NISTConstants.P192_2S_Y.ToUpper().Replace(" ", ""))
     {
Console.WriteLine("Weierstrass DOUBLE POINT: 2P = R is
correct.");
}
```

Şekil 5.66: NIST vektörleri ile nokta çiftleme testi.

6. PERFORMANS ÖLÇÜMLERİ

Eliptik eğri tabanlı kriptografinin kullanılabilirliğini belirleyen en önemli faktörlerden biri skaler çarpma işlemidir. Bir noktanın bir k skaler sayısı ile çarpımı EEK için temel bir işlemdir. Eğri üzerinde Q ve P tanımlı noktalar, k ise tam sayı olmak üzere Q=kP şeklinde ifade edilen skaler çarpma işlemi, P=(x,y) noktasından başlayarak, k defa nokta çiftleme ile başlayıp, nokta toplama ile devam eden işlemler sonucunda Q=kP noktasını elde etme sürecidir. Bu sürecin en hızlı bir şekilde olması, en az işlem ve en az zaman maliyetini sağlaması Bölüm 4'de anlatılan algoritma tasarımlarını önemli kılmaktadır.

Nokta toplama ve nokta çiftleme işlemleri pratikte kolayca uygulanabilir olmasına rağmen işlem maliyetleri çok yüksek olabilmektedir. Bu nedenle özellikle kısıtlı kaynaklara sahip donanımlar için EEK'nın kullanımında bu yüksek maliyetler göz önüne alınarak en uygun, en hızlı algoritma tasarımları kullanılmalıdır.

Bu bölümde 192 bitlik byte türünden ifade edilen P ve Q noktaları ile NIST test vektörleri kullanılarak, farklı iki pc donanımında nokta toplama ve nokta çiftleme işlemleri için ayrı ayrı performans ölçümleri yapılmıştır.

Performans ölçümleri için Windows XP işletim sistemine sahip Intel(R) Core2 Quad 2.4 GHz işlemcisi, 1 Gb Ram'i olan bilgisayar donanımı kullanılmıştır.

Önceki bölümde anlatılan yazılımı gerçekleştirilmiş olan algoritmaların nokta toplama ve nokta çiftleme işlemleri, .Net kütüphanesindeki Stopwatch nesnesi kullanılarak incelenmiş, ilgili işlemin timing değerlerinin ortalaması alınarak aşağıdaki tabloda belirtilen sonuçlar elde edilmiştir. İşlem ortalamaları hesaplanırken ilgili fonksiyon 10 bin defa tekrarlanarak geçen zaman milisaniye cinsinden hesap edilmiştir.

Tabloda M (Çarpma), S (Kare alma), D (Sabit sayı ile çarpma), Add (Toplama ve çıkarma işlemleri), times2 (2 ile çarpma işlemi), times3 (3 ile çarpma işlemi), Inv (Tersini alma işlemi), Inv2 (1/2 ile çarpma) olarak ifade edilmiştir.

- Örnek olarak maliyet hesabı şu şekilde ifade edilebilir:
  - $T_1 = X_1^2 X_2^2$ → 2 kare alma işlemi ve 1 çarpma işlemi (2S + 1M)
  - $T_2 = X_1 + 2Y_1$ → 1 toplama ve 1 adet 2 ile çarpma işlemi (1add + 1times2)

Algritma tasarımlarını incelediğimiz farklı koordinat sistemlerinde eliptik eğri modellerinin nokta toplama işlem maliyetleri aşağıdaki tabloda gösterilmiştir.

Tablo 6.1: Eliptik eğri modelleri nokta toplama maliyet tablosu.

	Eliptik Eğri Modeli	**Koordinat Sistemi**	**Nokta Toplama**
1	Weierstrass	Projektif	12M+2S+7add + 1times2 + 1times3 + 1Inv2
2	Jacobian	Jacobian	8M+3S+6add+1times2
3	Chudnowsky Jacobian	Jacobian	13M+4S+6add
4	Jacobian Quartics	Projektif	10M+3S+1D+13add+1times2
5	Extended Jacobi Quartics	Extended Projektif	7M+3S+2D+17add+1times2+1Inv2
6	Jacobi Intersection	Projektif	13M+2S+2D+13add
7	Twisted Jacobi Intersection	Projektif	13M+2S+5D+13add
8	Edwards	Projektif	10M+1S+1D+7add
9	Twisted Edwards	Projektif	10M+1S+2D+7add
10	Twisted Edwards	Inverted	9M+1S+2D+7add
11	Huff	Projektif	11M+3D+12add
12	Lopez Dahap	Projektif	13M+4S+9add
13	Hessian	Projektif	12M+3add
14	Twisted Hessian	Projektif	12M+1D+3add

Bölüm 4'de ifade edilen algoritmaların nokta toplama işlemleri teorik olarak yukarıda ifade edildiği gibidir. Tablo incelendiğinde Weierstrass ve Extended Jacobi Quartic eliptik eğri modellerinin algoritma tasarımlarında en çok farklı işlem içeren eğri modeli olduğu gözlenmektedir. Aynı şekilde Hessian eğri modelinin ise algoritma tasarımında sadece çarpma ve toplama işlemlerinin olması dikkat çekmektedir.

Algritma tasarımlarını incelediğimiz farklı koordinat sistemlerinde eliptik eğri modellerinin nokta çiftleme işlem maliyetleri aşağıdaki tabloda gösterilmiştir.

Tablo 6.2: Eliptik eğri modelleri nokta çiftleme maliyet tablosu.

	Eliptik Eğri Modeli	**Koordinat Sistemi**	**Nokta Çiftleme**
1	Weierstrass	Projektif	5M + 6S + 1D + 7add + 3times2 + 1times3
2	Jacobian	Jacobian	5M+4S+5add+2times2+1times3
3	Chudnowsky Jacobian	Jacobian	3M+6S+1D+4add +7times2+1times3
4	Jacobian Quartics	Projektif	1M+9S+1D+10add+4times2
5	Extended Jacobi Quartics	Extended Projektif	8S+9add+4times2
6	Jacobi Intersection	Projektif	3M+4S+1D+7add +1times2
7	Twisted Jacobi Intersection	Projektif	3M+4S+1D+7add+1times2
8	Edwards	Projektif	3M+4S+5add+1times2
9	Twisted Edwards	Projektif	3M+4S+1D+6add+1times2
10	Twisted Edwards	Inverted	3M+4S+2D+6add+1times2
11	Huff	Projektif	6M+5S+2D+10add
12	Lopez Dahap	Projektif	4M+5S+1D+5add
13	Hessian	Projektif	7M+1S+8add
14	Twisted Hessian	Projektif	6M+3S+1D+3add

Bölüm 4'de ifade edilen algoritmaların nokta çiftleme işlemleri teorik olarak yukarıda ifade edildiği gibidir. Tablo incelendiğinde Weierstrass ve Chudnowsky Jacobian eliptik eğri modellerinin algoritma tasarımlarında en çok farklı işlem içeren eğri modelleri olduğu gözlenmektedir. Extended Jacobi Quartic eğri modelinde kullanılan algoritma ise en çok kare alma işleminin yapıldığı algoritma olduğu görülmektedir.

6.1. Stopwatch Kullanımı

Aritmetik işlemler, nokta toplama ve nokta çiftleme işlemleri için geçen süre (performans) hesaplaması .Net kütüphanesindeki Stopwatch nesnesi kullanılarak gerçekleştirilmiştir. Stopwatch nesnesinin kullanımı aşağıdaki gösterildiği şekilde yapılmıştır.

Öncelikli olarak Stopwatch nesnesi *Stopwatch stopwatch = new Stopwatch();* ifadesi ile oluşturulmuştur. Bu nesne for döngüsü içerisinde "Start" edilmiş fonksiyon bitiminde ise stop edilerek geçen toplam süre kaydedilmiştir.

Nokta toplama ve çarpma işlemlerinin *for* döngüsü içerisinde kaç defa yapılacağını belirten integer bir değer ataması aşağıdaki şekilde yapılır.

- int maxNumber = 10000;

For döngüsü içerisinde elde edilen ilk değer hesaba katılmadığı için *maxNumber* değerine (10bin) ulaşıncaya kadar döngü devam eder.

```
double t0 = 0;
for (int i = 0; i =< maxNumber; i++)
      {
          if (i == 1) // (i=0) ilk değer hesaba katılmıyor
             stopwatch.Start();
          curve.addPoint(w_pointP, w_pointQ, w_pointR);   // add R = P + Q
      }
      stopwatch.Stop();
      t0 = stopwatch.Elapsed.TotalMilliseconds;
```

Şekil 6.1: Stopwatch kullanımı.

Ortlama geçen zamanı bulmak için Stopwatch nesnesinin tutmuş olduğu toplam geçen süre, *maxNumber* sayısına bölünür.

- Ortalama geçen süre = (t0 / maxNumber) ifadesi ile elde edilmiştir.

6.2. Aritmetik İşlem Maliyetleri

Microsoft .Net platformunda c# ile gerçekleştirilen yazılımda byte cinsinden 192 bitlik sayılarla yapılan çarpma, kare alma, ters alma, toplama, çıkarma, mod alma gibi aritmetik işlemlerin intel Core 2 Quad işlemcili pc'de Bölüm 6.1'de anlatılan Stopwatch nesnesi kullanılarak elde edilen performans ölçümleri aşağıdaki tabloda verilmiştir.

Tablo 6.3: 192 Bitlik sayı için aritmetik işlem süreleri (ms).

İşlem	F_{P192}
Çarpma	0,002889
Kare alma	0,002777
Ters alma	0,222348
Toplama	0,001576
Çıkarma	0,001562
2 ile çarpma	0,001521
3 ile çarpma	0,001558
½ ile çarpma	0,002780
Mod Alma	0,000756

Yazılımda *modMul* fonksiyonu ile gerçekleştirdiğimiz çarpma işlemi (M-Multiplication), 0,002889 ms iken *modInv* fonksiyonu ile gerçeklediğimiz tersini alma işlemi (I-Inversion) 0,222348 ms çıkmıştır. Bu sonuca göre ters alma işlemi çarpma işleminin yaklaşık olarak 77 katı bir maliyete denk düşmektedir. Çıkan bu sonuç ters alma işleminin nokta toplama ve nokta çiftleme işlemlerindeki maliyetinin ne kadar yüksek olduğunu göz önüne sermektedir. Bu nedenle eliptik eğri işlemlerinde ters alma işleminden kaçınılmalıdır. Tersini alma işlemi .Net kütüphanesi Framework 4.0 BigInteger sınıfında bulunmadığı için, Penk algoritması [26] kullanılarak gerçekleştirilmiştir. Daha uygun (efficient) ve işlem maliyeti az bir ters alma algoritması kullanılarak bu maliyet daha da düşürülebilir.

Kare alma işlemi (S-Squaring) ve ½ ile çarpma işlemi, çarpma işlemi ile yaklaşık aynı maliyete denk gelmektedir. Kare alma işleminde farklı bir algoritma kullanılmamış sayının kendisi ile çarpımı şeklinde ifade edilmiştir.

modAdd fonksiyonu ile gerçekleştirdiğimiz toplama (addition), *modSub* fonksiyonu ile yaptığımız çıkarma (subtraction), 2 ile çarpma ve 3 ile çarpma işlemleri, çarpma işlemi ile karşılaştırıldığı zaman yaklaşık iki kat fark olduğu gözükmektedir. Toplama ve çıkarmanın çarpmaya göre çok daha düşük maliyeti vardır. Bazı platformlarda yapılan işlemlerde bu fark çok daha büyük olduğu için maliyet analizlerinde toplama ve çıkarma işlemi ihmal edilmektedir. Burada ele alınan perofrmans değerlendirmesinde toplama ve çıkarma işlemleri değerlendirmeye katılmıştır. 2 ile çarpma işlemi, sayının kendisi ile toplamı şeklinde de ifade edilebileceğinden çıkan performans sonucu toplama işlemine denk düşmektedir.

Mod alma işlemi (indirgeme-reduction), her çarpma işleminden sonra gerçekleştirilmesi gereken bir aritmetik işlemdir. Yazılımda *modComplete* fonksiyonu ile gerçekleştirdiğimiz bu işlem her nokta toplama veya nokta çiftleme işlemi sonrasında çağrılarak indirgeme yapılmış ve sonuçların F_{192} alanı içerisinde kalması sağlanmıştır. Bu işlem artı maliyet getirse de çarpma işlemine göre çok daha düşük maliyet içerdiğinden ve fonksiyonlara etkisi aynı oranda olduğundan performans değerlendirmesinde çok önemli etkisi bulunmamaktadır.

6.3. Teorik ve Pratik Nokta İşlem Maliyetleri

Toplam maliyet hesabı yapılırken Bölüm 6.2'de incelenen aritmetik işlemlerin performansları göz önüne alındığında, Bölüm 4'de ele alınan algoritmalarda ifade edilen toplama, çıkarma, 2 ile ve 3 ile çarpma işlemleri için geçen zaman değerleri yaklaşık eşit alınıp toplanarak "add" olarak ifade edilmiştir. Aynı şekilde çarpma, kare alma, sabit sayı ile çarpma ve 2'ye bölme işlemleri de toplanarak "M" ile ifade edilmiştir.

Pratik olarak nokta toplama ve nokta çiftleme işlemleri performans ölçümleri için geliştirmiş olduğumuz yazılımla, üzerinde 2.4 GHz hızında ve 1 GB Ram bulunan Intel Core2 Quad işlemcili Windows XP işletim sistemi çalışan PC ile elde edilen sonuçlar aşağıdaki gibidir.

Tablo 6.4: Eliptik eğri modelleri nokta toplama ve çiftleme zamanlama tablosu.

	Eliptik Eğri Modeli	**Nokta Toplama Maliyeti**	**Nokta Toplama Süresi**	**Nokta Çiftleme Maliyeti**	**Nokta Çiftleme Süresi**
1	Weierstrass	15M+9add	0,044938	12M+11add	0,047336
2	Jacobian	11M+7add	0,033717	9M+8add	0,034547
3	Chunowsky Jacobian	17M+6add	0,040264	10M+12add	0,041168
4	Jacobian Quartics	14M+14add	0,053631	11M+14add	0,048003
5	Extended Jacobi Quartics	13M+18add	0,060701	8M+13add	0,041756
6	Jacobi Intersection	17M+13add	0,055715	8M+8add	0,030251
7	Twisted Jacobi Intersection	20M+13add	0,064364	8M+8add	0,029298
8	Edwards	12M+7add	0,038906	7M+6add	0,027281
9	Twisted Edwards	13M+7add	0,041618	8M+7add	0,031268
10	Inverted Twisted Edwards	12M+7add	0,038854	9M+7add	0,036252
11	Huff	14M+12add	0,053891	13M+10add	0,049180
12	Lopez Dahap	17M+9add	0,048831	10M+5add	0,034077
13	Hessian	12M+3add	0,032395	8M+8add	0,032980
14	Twisted Hessian	13M+3add	0,033524	10M+3add	0,029220

Yapılan çalışmada elde edilen sonuçlara bakıldığı zaman, farklı koordinat sistemleri üzerinde gerçekleştirilen farklı eliptik eğri formları için ortaya çıkan sonuçların, standart sapmalar göz önüne alınarak, Çizelge 6.4'de gösterildiği gibi teorik olarak belirtilen maliyet analizleri ile uyumlu zaman gereksinimleri gösterdigi görülmüştür. Yukarıda elde edilen sonuçları incelediğimizde nokta toplama işleminde Hessian eğri modelinin 0,032395 ms, Twisted Hessian eğri modelinin 0,033524 ms ve Jacobian eğri modelinin 0,033717 ms değerleri ile daha iyi performans sergilediği gözlenmiştir.

Nokta çiftleme işlemleri incelendiğinde ise Edwards eğri modelinin 0,027281 ms, Twisted Hessian eğri modelinin 0,029220 ms ve Twisted Jacobi Intersection modelinin 0,029298 ms gibi bir değerle diğer algoritmalara göre daha iyi performans elde ettiği görülmüştür.

7. SONUÇ ve ÖNERİLER

Bu tezde, eliptik eğri kripto sistemlerinin kullandığı matematik temeller, aritmetik işlemler, eğri üzerinde nokta toplama ve nokta çiftleme gibi geometrik ve aritmetik işlemler konu edinilmiş, eliptik eğrilerle çalışmanın getirdiği işlem maliyetlerini azaltmak için geliştirilen eliptik eğri formları ve eliptik eğri algoritmaları farklı koordinat sistemlerinde kullanılarak bir yazılım gerçekleştirilmiştir.

Açık anahtar şifreleme algoritmaları içerisinde, eliptik eğri tabanlı şifreleme anahtar boyutlarının kısalığı nedeni ile daha avantajlıdır. Bu avantajı daha da artırmak için çeşitli eliptik eğri formları ve eliptik eğri algoritmaları geliştirilmeye devam etmektedir. Eliptik eğri algoritmalarında ise verimlilik ve performans, nokta işlemlerinin hızına bağlıdır. Nokta işlemlerinin en önemli özelliği olan skaler çarpma operasyonu için gerekli nokta toplama ve nokta çarpma işlemlerinin daha verimli (effcient), daha hızlı ve daha az maliyetle gerçeklenebilmesi eliptik eğri formlarının tasarımını önemli kılmaktadır. Donanım olarak sınırlı imkanların olduğu ve özellikle yüksek hızlda ve güvenli şifreleme işlemlerinin yapılabilmesi için literatüre geçmiş önemli eliptik eğri modelleri bulunmaktadır. Bu formlardan başlıcaları bu tezde deneysel olarak incelenmiştir.

Eliptik eğri formlarının iyileştirilmeye açık olması ve bu eğriler üzerinde daha hızlı nokta toplama ve nokta çiftleme gibi aritmetik işlemlerin yapılabiliyor olması, EEK'nin daha çok kullanılma tercihini artıran unsurlar olduğu gözlenmiştir.

Koordinat sistemlerine bağlı olarak Eliptik formların nokta toplama veya nokta çiftlemeye göre birbirlerine üstünlük sağladıkları görülmektedir. Özellikle tersini alma gibi matematiksel işlemlerde, daha az çarpma işlemi gibi maliyet azaltacak yeni yaklaşımlar aranmaya devam edilmelidir. Çizelge 6.4'de de görüldüğü gibi eliptik eğri formlarında "twisted" formlar nokta toplama işlemine göre nokta çiftleme işleminde en iyi hız değerlerini elde etmektedir.

Günümüzde hem sosyal alanda hem de ticari alanda yoğun olarak kullandığımız akıllı cep telefonları gibi kısıtlı donanım kaynaklarına sahip mobil cihazlarda EEK kullanımı, daha iyi performansa sahip eliptik eğri algoritmalarının tasarımını önemli kılmaktadır.

KAYNAKLAR

[1] Koblitz N., (1987), "Elliptic Curve Cryptosystems", Mathematics of Computation, 48, 203-209.

[2] Miller V., (1985), "Uses of Elliptic Curves in Cryptography", Advances in Cryptology, 218, 417-426.

[3] Akben S. B., Subaşı A., (2005), "RSA ve Eliptik Eğri Algoritmasının Performans Karşılaştırması", Kahramanmaraş Sütçü İmam Üniversitesi Fen ve Mühendislik Dergisi, 8(1), 35-41.

[4] Murphy T., (2006), "Course 373: Finite Fields", Trinity College School of Mathematics, University of Dublin, Ireland.

[5] Mentens N., Ors S. B., Preneel B., Vandewalle J., (2004), "An FPGA implementation of an elliptic curve processor over GF (2m)", Proceedings of the 14th ACM Great Lakes Symposium on VLSI, 454-457, Boston, MA, USA, 26-28 April.

[6] Ors S. B., Batina L., Preneel B., Vandewalle J., (2003), "Hardware implementation of an elliptic curve processor over GF(p)", Proceeding of the 14th IEEE International Conference on Application-Specific Systems, Architectures, and Processors, 433-443, Leuven, Heverlee, Belgium, 24-26 June.

[7] NIST, (2001), Specification for the Advanced Encryption System AES (FIPS 197), National Institute for Standarts and Technology.

[8] Hankerson D., Menezes A., Vanstone S., (2003), "Guide to Elliptic Curve Cryptography", 1th Edition, Springer-Verlag.

[9] Menezes A. J., (1993), "Elliptic Curve Public Key Cryptosystems", 3th Edition, Kluwer Academic Publishers.

[10] Knuth D. E., (1998), "The Art of Computer Programming 2 / Seminumerical Algorithms", 3th Edition, Addison-Wesley.

[11] Ashraf M., Kırlar B. B., (2012), "On The Alternate Models of Elliptic Curves", International Journal of Information Security Science, 1, 2-3.

[12] NIST, (2010), Mathematical routines for the NIST prime elliptic curves, National Institute for Standarts and Technology.

[13] Cohen H., Miyaji A., Ono T., (1998), "Efficient Elliptic Curve Exponentiation Using Mixed Coordinates", Advances in Cryptography, 15, 4-5.

[14] Web 1, (2014), http://www.hyperelliptic.org/EFD, (Erişim Tarihi: 30/04/2014).

[15] Billet O., Joye M., (2003), “The Jacobi Model of an Elliptic Curve and Side Channel Analysis”, Applied Algebra, Algebraic Algorithms and Error Correcting Codes: 15th International Symposium, 34-43, Toulouse, France, 12-16 May.

[16] Hisil H., Wong K.K., Carter G., Dawson E., (2009), “Jacobi Quartic Curves Revisited”, Australasian Conference on Information Security and Privacy, 452-468, Brisbane, Australia, 1-3 July.

[17] Liardet P., Smart N., (2001), “Preventing SPA/DPA in ECC systems using the Jacobi form”, Cryptographic Hardware and Embedded Systems, 391-401, Paris, France, 13-16 May.

[18] Feng R., Nie M., Wu H., (2013), “Twisted Jacobi Intersections Curves”, Theory and Applications of Models of Computation - 10th International Conference, 24-35, Hong Kong, China, 20-22 May.

[19] Edwards H., (2007), “A normal form for elliptic curves”, Bulletin of the American Mathematical Society, 44(3), 393-422.

[20] Bernstein D., Birkner P., Joye M., Lange T., Peters C., (2008), “Twisted Edwards curves”, First International Conference on Cryptology in Africa, 389-405, Casablanca, Morocco, 11-14 June.

[21] Joye M., Tibouchi M., Vergnaud D., (2010), “Huff's model for elliptic curves”, 9th Algorithmic Number Theory Symposium, 234-250, Inria, Nancy, France, 19-23 July.

[22] Feng R., Wu H., (2012), “Elliptic curves in Huff's mode”, Wuhan University Journal of Natural Sciences, 17(6), 473-480.

[23] Joye M., Quisquater J., (2001), “Hessian elliptic curves and sidechannel attacks”, Cryptographic Hardware and Embedded Systems, 402-410, Paris, France, 13-16 May.

[24] Farashahi R. R., Joye M., Efficient, (2010), “Arithmetic on Hessian Curves”, 13th International Conference on Practice and Theory in Public Key Cryptography, 243-260, Paris, France, 26-28 May.

[25] L´opez J., Dahab R., (1999), “Fast multiplication on elliptic curves over GF(2m) without precomputation”, International Workshop on Cryptographic Hardware and Embeded Systems, 316-327, Worcester, Massachusetts, USA, 12-13 Agust.

ÖZGEÇMİŞ

Ergin Öztürk 1980 yılında Kastamonu'da doğdu. 2002 yılında İstanbul Üniversitesi Elektrik-Elektronik Mühendisliği Bölümü'nden mezun oldu. 2001-2003 yılları arasında Elkosis Elektronik Ltd. Şti.'nde, 2005-2007 yılları arasında Akgül Ambalaj Ltd. Şti.'nde çalıştı. 2007 yılından beri TUBİTAK BİLGEM İleri Teknolojiler Araştırma Merkezi, Yazılım ve Simülasyon Birimi'nde araştırmacı olarak çalışmaktadır. Evli ve iki çocuk babasıdır.