Elliptische Kurven in der Charakteristik p $>$ 3 und die Implementierung der Arithmetik in der Programmiersprache Python

Studienarbeit T3_3101

Hochschule:	Duale Hochschule Baden-Württemberg Mannheim
Kurs:	TINF20IT2
Student 1:	Aaron Lacks, 1673436
Student 2:	Luc Forster, 5716926
Studiengangsleiter:	Prof. Dr. Nathan Sudermann-Merx
Betreuer:	Prof. Dr. Reinhold Hübl
Bearbeitungszeitraum:	18.10.2022 - 02.05.2023

Selbstständigkeitserklärung

Hiermit erkläre ich durch meine Unterschrift, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe.

Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken – dazu gehören auch Internetquellen – als solche kenntlich gemacht habe.

Ort, Datum	Unterschrift Student 1
Ort, Datum	Unterschrift Student 2

Zusammenfassung

Hier Text des Abstract in Deutsch.

Abstract

Hier Text des Abstract in Englisch.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abbildungsverzeichnis

1. Ziele und Vorgehensweise

In diesem Kapitel werden die Ziele dieser Studienarbeit sowie die geplante Vorgehensweise erläutert.

1.1. Ziel der Arbeit

Elliptische Kurven sind ein in der Kryptographie gängige Methode >zur Verschlüsselung von Daten. Da elliptische Kurven keine simple Thematik ist wird diese Studienarbeit geschrieben, um die recht komplexen elliptischen Kurven verständlicher zu machen. Neben der Definition der elliptischen Kurven mit $p > 3$ geht es insbesondere auch darum, die Arithmetik, also die Rechenregeln auf elliptischen Kurven, verständlich zu erläutern. Das erklärte soll durch ein praktisches Beispiel verdeutlicht werden. Dafür wird eine Verschlüsselung mittels elliptischen Kurven vorgenommen. Bewerkstelligt wird dies über ein simples Beispiel des *Elliptische Kurven Diffie-Hellmann*, kurz ECDH. Die Arithmetik und der ECDH wird am Ende der Studienarbeit noch in Python implementiert, damit man ein praktisch umgesetztes Beispiel sehen kann. Dadurch wird klar, wie elliptische Kurven mittels Programmcode implementiert werden können. Ziel der gesamten Arbeit ist es, elliptische Kurven mit $p > 3$ anschaulich und durch adäquate Beispiele einfach zu erklären. Um dieses Ziel zu erreichen, ist eine strukturierte Vorgehensweise essentiell.

1.2. Geplante Vorgehensweise

Wie zu Beginn jeder theoretischen Arbeit müssen dafür anfänglich einige grundlegende Themen besprochen werden wie algebraische Strukturen und wichtige Erklärungen und Definitionen wie die Primzahlen und Grundlagen der Verschlüsselung. Die einfeführten und erläuterten Grundlagen bilden die Wissensbasis, welche für das Verständnis aller Folgekapitel vorausgesetzt wird. Im Anschluss wird die Arithmetik sowie die Gruppeneigenschaften von eilliptischen Kruven erläutert. Weiterhin werden einige gängige Methoden erklärt, um Punkte auf elliptischen Kurven zu bestimmen. Im Anschluss wird das diskrete Logarithmusproblem auf elliptischen Kurven erläutert. Im Anschluss werden Kryptosysteme vorgestellt, denen das diskrete Logarithmusproblem zugrunde liegt. Die Kryptosysteme werden erläutert und es werden Beispiele zur Umsetzung mit elliptischen Kurven gegeben. Danach werden die Implementierungen verschiedenster Algorithmen, der Punktbestimmung sowie

der Kryptosysteme, welche in der Arbeit erläutert wurden, in Python implementiert. Dazu werden auch erzeugte Ausgaben der Programme und Code-Teile gezeigt. Am Ende der Studienarbeit wird ein Fazit sowie ein Zukunftsausblick gegeben. Es soll dadurch Optimierungspotentiale der Studienarbeit als auch zukünftige Anknüpfungspunkte bei folgenden Studienarbeiten herausgestellt und Vorschläge für mögliche Themen unterbreitet werden.

2. Grundlagen

Diese Studienarbeit befasst sich mit dem komplexen Thema der Elliptischen Kurven in der Kryptographie. Die Kryptographie ist ein mathematisches Thema, bei welchem es zu Anfang der Legung einer Grundlage für das Verständnis der Inhalte dieser Studienarbeit bedarf. In diesem Kapitel werden sowohl die mathematischen als auch die kryptographischen Grundlagen zum Verständnis der Inhalte dieser Studienarbeit gelegt.

2.1. Primzahlen

In der Zahlentheorie, einem Teilbereich der Mathematik, werden viele unterschiedliche Eigenschaften von Zahlen untersucht. Durch die Untersuchung erhofft man sich neue Erkenntnisse für Wissenschaft und Technik. Die Primzahlen als mathematisches Forschungsgebiet sind hierbei ein Teilbereich der Zahlentheorie. Im Folgenden werden Primzahlen definiert und deren Eigenschaften erläutert. Anschließend wird untersucht, wie Primzahlen berechnet werden können. Am Ende wird erläutert, welche Rolle Primzahlen in der Kryptologie und modernen Kryptosystemen innehaben.

2.1.1. Definition und Eigenschaften

Es gibt viele unterschiedliche Zahlenmengen. Beispielsweise gibt es die Menge der reellen Zahlen $\mathbb{R}$. Diese beinhalten als Teilmenge die rationalen und die irrationalen Zahlen. Die natürlichen Zahlen $\mathbb{N}$ bilden hierbei alle positiven ganzen Zahlen ab. Dabei gibt es $\mathbb{N}^+$ exklusive der Zahl 0 als Teilmenge mit

$$\mathbb{N}^+ = \{1, 2, 3, 4, 5, ...\}$$

und $\mathbb{N}_0$ inklusive der Zahl 0 als Teilmenge mit

$$\mathbb{N}_0 = \{0, 1, 2, 3, 4, 5, ...\}.$$

Die Primzahlen $\mathbb{P}$ sind hierbei etwas ganz besonderes. Sie unterscheiden sich von anderen Zahlen. Sie sind eine Teilmenge der natürlichen Zahlen und die Kardinalität ihrer Elemente ist unendlich respektive die Anzahl der Primzahlen ist unendlich. Die Unendlichkeit der Primzahlen konnte schon mit mehreren mathematischen Sätzen bewiesen werden, unter anderem dem Satz von Euklid. Auf die unendlichkeit der

Primzahlen sowie deren bestimmung wird später in 2.1.2 eingegangen.

Doch wie genau sind Primzahlen definiert? Dafür muss erst geklärt werden, was zusammengesetzte Zahlen sind. Dadurch können die Primzahlen klarer von anderen natürlichen Zahlen abgegrenzt werden. Eine natürliche Zahl mit $n \geq 2$ ist eine zusammengesetzte Zahl, falls es zwei natürliche Zahlen m und k mit den Eigenschaften:

$$m, k \geq 2 \text{ oder } m, k \neq n, \text{ für die gilt: } m \cdot k = n.$$

Zusammengesetzte natürliche Zahlen können also immer als Produkt zweier natürlicher Zahlen ≥ 2 beschrieben werden. Primzahlen bilden hierzu das Gegenstück. Eine Primzahl p ist eine natürliche Zahl mit $p \geq 1$, wobei p nur durch 1 und sich selbst teilbar sein darf. Durch diese Eigenschaft sind Primzahlen nicht zusammengesetzt. Sie können weder als Produkt von natürlichen Zahlen n mit $n \geq 2$ noch als Produkt von zwei Primzahlen beschrieben werden. Man nehme als Beispiel die Primzahl 7. Sie lässt sich nicht als Produkt von natürlichen Zahlen oder als Produkt von Primzahlen darstellen. Als Gegenbeispiel nimmt man die zusammengesetzte natürliche Zahl 28. Sie kann durch Multiplikation aus den Zahlen 2 und 14 gebildet werden:

$$2 \cdot 14 = 28.$$

Eine weitere Eigenschaft von Primzahlen ist, dass sie das Grundgerüst zur Bildung von Zahlen sind, da man aus ihnen alle natürlichen Zahlen bilden kann. Eine zusammengesetzte natürliche Zahl n mit $n \geq 2$ kann wie bereits beschrieben immer als Produkt von mindestens zwei weiteren natürlichen Zahlen dargestellt werden. Die einzelnen Faktoren können wiederum ebenfalls als Produkt von zwei weiteren natürlichen Zahlen dargestellt werden. Diese Aufteilung geht rekursiv so lange weiter, bis die Faktoren lediglich Primzahlen sind. Die übrig gebliebenen Faktoren dieses Produktes heißen Primfaktoren. Die Zerlegung einer zusammengesetzten natürlichen Zahl in ihre Primfaktoren nennt man Primfaktorzerlegung. Zweck dieser Primfaktorzerlegung ist es Zahl als Produkt von mehreren Primzahlen darzustellen. Nehmen wir als Beispiel die Zahl 28. Im vorigen Absatz stellten wir diese zusammengesetzte natürliche Zahl durch die Multiplikation von 2 und 14 dar. Die Zahl 2 ist eine Primzahl. Die Zahl 14 ist noch nicht in ihre Primzahlfaktoren zerlegt. Sie lässt sich als folgendes Produkt darstellen:

$$2 \cdot 7 = 14.$$

Da 7 auch eine Primzahl ist, wurden alle Primfaktoren gefunden. Die Zahl 28 lässt

sich in ihrer Primfaktorzerlegung also wie folgt darstellen:

$$2 \cdot 2 \cdot 7 = 28.$$

Die Mehrfachheit von Primzahlen lässt sich auch als Potenz schreiben. Somit wird daraus

$$2^2 \cdot 7 = 28.$$

Der Vorteil durch die Potenzen zeigt sich besonders bei großen Zahlen, da diese oft eine große Anzahl an Primfaktoren haben können. Nimmt man als Beispiel die Zahl 5281250000. Diese setzt sich mit ihren Primfaktoren wie folgt zusammen:

$$2 \cdot 2 \cdot 2 \cdot 2 \cdot 5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 \cdot 13 \cdot 13 = 5281250000.$$

Man erkennt rasch, dass sich die Primfaktoren mit der Potenzschreibweise zusammenfassen lassen und man so die Primfaktorzerlegung wie folgt darstellen kann:

$$2^4 \cdot 5^9 \cdot 13^2 = 5281250000.$$

Die Vorteile der Potenzschreibweise liegt hier auf der Hand, da man erheblich Zeit beim Aufschreiben und Platz auf dem Papier spart.

2.1.2. Bestimmung von Primzahlen

Nachdem die grundlegenen Eigenschaften der Primzahlen angeführt wurden, muss auf die Bestimmung von Primzahlen eingegangen werden. Paulo Ribenboim geht in seinem Buch „*Die Welt der Primzahlen: Geheimnisse und Rekorde*“der Frage auf den Grund, ob primzahldefinierende Funktionen existieren. An einer Stelle des Buches geht er auf diese möglichen Funktionen und ihre Eigenschaften ein [1, S. 137]. Solch eine Funktion müsse laut ihm eine der folgenden drei Eigenschaften aufweisen, damit man sie zur Bestimmung von Primzahlen nutzen könne:

(a) $f(n) = p_n$ (die n-te Primzahl) für alle n ≥ 1;

(b) $f(n)$ ist immer prim und wenn n $\neq$ m, dann gilt: $f(n) \neq f(m)$;

(c) der positive Wertebereich der Funktion ist identisch mit der Menge der Primzahlen

Ribenboin erklärt, dass die Bedingung, um (a) zu erfüllen schärfer sei als (b) und als (c). Die bisher erzielten Resultate zur Findung einer Formel zur Bestimmung

von Primzahlen seien außerdem eher enttäuschend. Doch wenn die Funktionen zur Bestimmung von Primzahlen bisher enttäuschend waren, wie wurden diese bisher bestimmt?

Eine der simpelsten und sicher auch eine der ältesten Methoden ist das „Sieb des Eratosthenes“. Der Übersetzer Kai Brodersen beschreibt in seiner Übersetzung aus dem Jahre 2021 eines Buches aus dem Griechischen von Nikomachos von Gerasa, wie dieser sehr simpel die Funktionsweise des Siebes erläuterte [2, S. 7]. Die Richtigkeit dieses Verfahrens wurde von Nikomachos im frühen 2. Jh. n. Chr. belegt. Bei dem Verfahren schreibt man alle natürlichen Zahlen von 2 bis zu einer gewählten Zahl n in eine Liste. Um die Primzahlen zu erhalten, siebt man jetzt die zusammengesetzten natürlichen Zahlen aus, indem man Vielfache streicht. Man beginnt bei der kleinsten Zahl, der 2. Man schreitet in der Liste fort und streicht alle Vielfachen der 2 bis zur höchsten gewählten Zahl n durch. Anschließend beginnt man mit der nächstgrößeren Zahl, welche nicht durchgestrichen ist respektive ausgesiebt wurde und streicht von dieser ebenfalls alle Vielfachen bis zur höchsten Zahl n durch. Den simplen Algorithmus führt man nun solange fort, bis man keine Vielfachen mehr streichen kann. Die übriggebliebenen Zahlen sind die Reihe der Primzahlen bis n. Die Darstellung in einer Tabelle ist heutzutage geläufig, da dies übersichtlicher ist. In der folgenden Tabelle wurde der Algorithmus des Siebes des Eratosthenes von 2 bis 100 angewandt:

1	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	~~77~~	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
~~91~~	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

Das Sieb des Eratosthenes ist eine Möglichkeit, Primzahlen genau zu bestimmen. Problematisch wird es jedoch bei großen Zahlen. Für jede Zahl müssen je alle anderen Zahlen durchgegangen werden und es muss eine Teilbarkeitsprüfung durchgeführt werden. Umso größer die Zahlen werden, desto rechenaufwendiger wird die Anwendung

des Sieb des Eratosthenes, um Primzahlen zu finden. Dieses ist also kein optimaler Ansatz, große Primzahlen zu bestimmen. Neben dem Sieb des Eratosthenes gibt es viele weitere Methoden, Primzahlen zu bestimmen.

Über Probabilistische Verfahren können ebenfalls Primzahlen bestimmt werden. Der Unterschied bei probabilistischen Primzahltests ist, dass die bestimmten Zahlen im Gegensatz zu den deterministischen Primzahltests wie das Sieb des Eratosthenes nur zu einem bestimmten Prozentsatz Primzahlen sind. Ein Beispiel ist der Miller-Rabin-Primzahltest. Weitz beschreibt diesen in seinem Buch über die Mathematik für Informatiker [**?**, S. 97]. Der Miller-Rabin-Test basiert auf dem kleinen Satz von Fermat und auf einer Eigenschaft der modularen Arithmetik: Wenn p eine Primzahl ist, dann sind die einzigen Zahlen in $\mathbb{Z}_p$, deren Quadrat 1 ist, die Zahl 1 und die Zahl - 1. Im Gegensatz zu anderen probabilistischen Primzahltests wie dem Fermat-Test kann bei dem Miller-Rabin-Test eine maximale Fehlerwahrscheinlichkeit für eine zu testende Zahl angegeben werden. Die Besprechung aller dieser Verfahren und Algorithmen soll jedoch nicht Thema dieser Arbeit sein.

2.1.3. Rolle der Primzahlen in der Kryptografie

Primzahlen haben einen effektiven Nutzen in der Kryptografie. Bevor man sich fragt, wie Primzahlen in der Kryptografie genutzt werden, sollte man die Kryptografie vorher definieren. Dietmar Wätjen beschreibt diesne Begriff seinem Buch „*Kryptographie: Grundlagen, Algorithmen, Protokolle*“aus dem Jahr 2018 [3, S. 1]. Kryptografie ist die Wissenschaft vom geheimen Schreiben. Kernziel ist es dabei, einen unverschlüsselten Text, genannt Klartext in einen Chiffretext überführt. Dieser Vorgang heißt *chiffrieren*. Der Vorgang, bei welchem der Chiffretext wieder in den Klartext überführt wird, heißt *dechiffrieren*. Für beide Vorgänge werden Schlüssel notwendig. Die Abbildung 2.1 stellt dies übersichtlich dar:
Wätjen beshreibt Kryptografische System, auch Kryptosystem genannt, als ein System aus fünf Komponenten:

1. Klartextraum M
2. Chiffretextraum C
3. Schlüsselraum K
4. Familie von Chiffriertransformationen $E_k : M \to C$ mit $k \in K$
5. Familie von Dehiffriertransformationen $D_k : C \to M$ mit $k \in K$

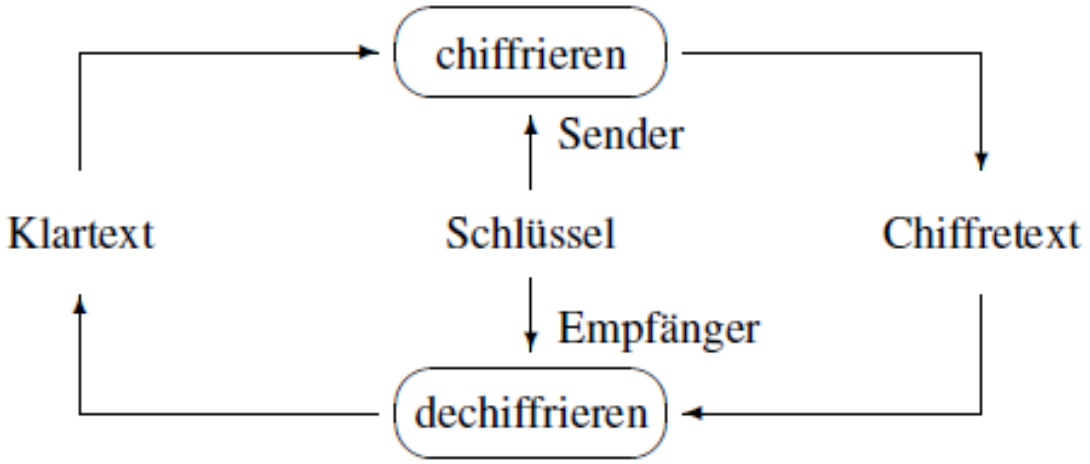

Abbildung 2.1.: Kryptografische Verschlüsselung
Quelle: [3, S. 1]

Laut Watjen sind M, C und K höchstens, abzählbare Mengen. Eine Chiffriertransformation E_K wird durch einen Schlüssel K und einen Chiffrieralgorithmus E definiert, welcher für jede Familie gleich ist. Eine Dechiffriertransformation D_K wird ebenfalls durch einen Schhlüssel K bestimmt. Desweiteren sollen die Kryptografischen Systeme nach Watjen die folgenden drei Eigenschaften aufweisen [3, S. 3]:

(1) Chiffrier- und Dechiffriertransformationen müssen für alle Schlüssel effizient berechnet werden können.

(2) Kryptographische Systeme müssen leicht zu benutzen sein. Das bedeutet, dass es ohne Schwierigkeiten möglich sein muss, einen Schlüssel K sowie die zugehörigen Abbildungen E_K und D_K zu finden.

(3) Die Sicherheit des Systems sollte auf der Geheimhaltung der Schlüssel und nicht auf der Geheimhaltung der Algorithmen beruhen. Die Kenntnis der Methode des Chiffrierens und Dechiffrierens soll also noch nicht den Klartext liefern.

Die heutzutage eingesetzten kryptographischen Systeme ermöglichen also eine schnelle und sichere Übertragung von sensiblen Informationen. Die Übertragung der Daten läuft also privat ab, da ein abgefangenes Chiffrat nicht direkt lesbar ist. Doch wie helfen dabei Primzahlen?

Primzahlen sind aufgrund ihrer hervorragenden Eigenschaften für die Kryptographie nützlich. Schon der Mathematiker Riemann suchte nach einer Ordnung hinter den

Primzahlen. Die Verteilung der Primzahlen scheint pseudozufällig. Dadurch können nur sehr schwierig Vorhersagen zu kryptographischen Systemen getroffen werden, die Primzahlen verwenden. Weiterhin besteht das Problem, dass es sehr schwer ist, ein Produkt aus zwei Primzahlen zu faktorisiern, wenn die Primzahlen sehr groß sind. Hübl erklärt dies in seinem Skript zur Ktyptologie [4, S. 93]. Auf dem Faktorisierungsproblem baut beispielsweise das RSA-Verfahren auf, dass es sich zunutze macht, dass die Faktorisierung von $N = p \cdot q$ nicht einfach gefunden werden kann, wenn p groß ist. Aus diesem Grund sind RSA-Verfahren sicherer, umso größer die Schlüssellänge ist, da dafür dafür große Primzahlen benötigt werden. Generell gesprochen benötigen viele Verfahren, welche den Klartext in einen Chiffretext überführen, für ihren Algorithmus Primzahlen. Ein gutes Beispiel ist der Diffie-Hellmann-Schlüsselaustausch (DHKE). In 2.3.2 wird anschaulich gezeigt, wie Primzahlen beim DHKE verwendet werden.

2.2. Algebraische Strukturen

Definiert durch die Zahlentheorie und als zentraler Untersuchungsgegenstand des mathematischen Teilgebietes der universellen Algebra, liefern algebraische Strukturen die Basis zur Realisierung komplexer symmetrischer und asymmetrischer Kryptosysteme, weshalb wir im folgenden Kapitel die Eigenschaften relevanter algebraischer Strukturen näher betrachten wollen. Darüber hinaus möchten wir Ihnen auch einige Werkzeuge zum Rechnen in der jeweiligen algebraischen Struktur an die Hand geben, welche zur späteren Realisierung von Kryptosystemen benötigt werden.

Unter einer sehr allgemeinen Betrachtung ist eine mathematische Struktur eine Liste nichtleerer Mengen, genannt Trägermengen, mit Elementen aus den Trägermengen, genannt Konstanten, und mengentheoretischer Konstruktionen über den Trägermengen. Diese sind konkret Funktionen über den Trägermengen. Im Weiteren beschränken wir uns auf den Fall einer einzigen Trägermenge, wodurch die Strukturen als homogen bezeichnet werden können.[5, S. 355]

Definition: Homogene algebraische Struktur Eine homogene algebraische Struktur ist ein Tupel $(M, c_1, \ldots, c_m, f_1, \ldots, f_n)$ mit $m, n \in \mathbb{N}$ und $n \geq 1$. Dabei ist M eine nichtleere Menge, genannt **Trägermenge**, alle c_i sind Elemente aus M, genannt die **Konstanten**, und alle f_i sind s_i-stellige Funktionen $f_i : M \to M$ im Fall $s = 1$ und $f_i : M^{s_i} \to M$ im Fall $s_i > 1$, genannt die (inneren) **Operationen**. Die lineare Liste$(0, \ldots, 0, s_1, \ldots, s_n)$ mit m Nullen heißt **Typ** oder die **Signatur**.

Laut dieser Definition muss eine homogen algebraische Struktur nicht unbedingt Konstanten enthalten, jedoch mindestens eine Operation. Das Paar $(\mathbb{N}, +)$ bildet beispielsweise eine homogene algebraische Struktur des Typs (2). Das 5-Tupel $(\mathbb{N}, 0, 1, +, \cdot)$ bildet ebenfalls eine homogen algebraische Struktur des Typs (0,0,2,2).
Algebraische Strukturen unterscheiden sich grundsätzlich durch ihren Typ. Wirklich charakterisiert werden sie aber erst durch die jeweils geltenden Axiome, d.h. bestimmte Eigenschaften, welche für die Konstanten und Operationen gefordert werden. Durch die Hinzunahme immer weiterer Axiome, entsteht eine Hierarchie immer feinerer Strukturen. Wir beginnen mit dem Monoid.[5, S. 355, 356]

2.2.1. Monoid

Definition: Monoid Eine algebraische Struktur $(M, e, \cdot)$ des Typs (0,2) heißt ein Monoid, falls für alle $x, y, z \in M$ die folgenden Monoid Axiome gelten:

- (Ass) $x \circ (y \circ z) = (x \circ y) \circ z$
- (Neu) $e \circ x = x = x \circ e$

Gilt zusätzlich noch für alle $x, y \in M$ die Gleichung $x \circ y = y \circ x$, so heiß $(M, e, \circ)$ ein **kommutatives Monoid**.

Die erste und die letzte Gleichung bilden das Assoziativ- und Kommutativgesetz ab. Durch die mittlere Gleichung wird ein neutrales Element e bezüglich der Operation gefordert, wobei sowohl die **Linksneutralität** als auch die **Rechtsneutralität** spezifiziert wird.

Einfache Beispiele für Monoide sind $(\mathbb{N}, 0, +)$, $(\mathbb{N}, 1, \cdot)$ und $(\mathbb{Z}, 0, +)$. Die Potenzierung in solchen Monoiden ist folgendermaßen definiert.[5, S. 356]

Definition: Potenzierung In einem Monoid $(M, e, \cdot)$ definiert man die n-te **Potenz** x^n von $x \in M$ durch $x^0 := e$ und $x^{x+1} = x \cdot x^n$ für alle $n \in \mathbb{N}$.

Daraus ergibt sich für das Monoid $(\mathbb{N}, 1, \cdot)$ die aus $\mathbb{R}$ gewohnte Potenzierung. Nach welcher für ein $x \in \mathbb{N}$ die Potenzierung $x^n = x_1 \cdot x_2 \cdot \cdots \cdot x_n$ ergibt. Betrachtet man jedoch den Monoid $(\mathbb{N}, 0, +)$, so ergibt analog dazu für ein $x \in \mathbb{N}$ die Potenzierung $x^n = x_1 + x_2 + \cdots + x_n = x \cdot n$, was also einer Multiplikation von x mit n entspricht [5, S. 356, 357].

2.2.2. Gruppe

Definition: Gruppe Eine algebraische Struktur $(G, e, \circ, inv)$ des Typs $(0, 2, 1)$ heißt **Gruppe**, falls für alle $x, y, z \in G$ die folgenden Axiome gelten:

- (Ass) $x \circ (y \cdot z) = (x \circ y) \circ z$
- (Neu) $e \circ x = x$
- (Inv) $inv(x) \circ x = e$

Gilt zusätzlich die Gleichung $x \circ y = y \circ x$ für alle $x, y \in G$, so heißt $(G, e, \circ, inv)$ eine **kommutative Gruppe** oder Abelsche Gruppe. Die Gruppenoperation $\circ$ ist *abgeschlossen*. D.h. für alle $a, b \in G$ gilt $a \circ b = c \in G$.
In jeder Gruppe $(G, e, \circ, inv)$ gelten für alle $x \in G$ folgende Formeln [5, S. 357-359]:

$$x \circ x = x \Rightarrow x = e \qquad x \circ e = x \qquad x \circ inv(x) = e$$

Diese lassen sich durch Anwendung der obigen Axiome und ein wenig Logik beweisen. Die Beweise können im Buch von Berghammer unter [5, S. 358, 359] nachvollzogen werden. Die folgenden Formeln beziehen sich auf die Eindeutigkeit neutraler und inverser Elemente in Gruppen.

$$(\forall z \in G : x \circ z = z) \Rightarrow x = e \qquad x \circ y = e \Rightarrow x = inv(y)$$

Auch diese lassen sich mit den schon bekannten Gruppengesetzen beweisen, was unter [5, S. 358, 359] nachgeschlagen werden kann. Weiterhin sind die folgenden Rechenregeln für Gruppen zu nennen.

$$inv(x \circ y) = inv(y) \circ inv(x) \qquad inv(inv(x)) = x \qquad inv(e) = e$$

Die erste Gleichung lässt sich leicht durch folgende Rechnung zeigen:

$$(inv(y) \circ inv(x)) \circ (x \circ y) = inv(y) \circ (inv(x) \circ x) \circ y = inv(y) \circ e \circ y = inv(y) \circ y = e$$

Damit ist gezeigt, dass $inv(y) \circ inv(x)$ linksinvers zu $x \circ y$ ist. Unter dieser Prämisse und der Eindeutigkeit der inversen Elemente muss die erste Gleichung gelten. Die beiden anderen Gleichungen lassen sich auf sehr ähnliche Weise zeigen.

Um Gruppen ein wenig anschaulicher zu machen, betrachten wir im Folgenden ein paar Beispiele:

- $(\mathbb{Z}, +)$ ist eine Gruppe. $\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$ ist die Menge der ganzen Zahlen, welche zusammen mit der Addition als Gruppenoperation eine abelsche Gruppe bildet, wobei $e = 0$ das neutrale Element und $-a$ das Inverse eines beliebigen Elements $a \in \mathbb{Z}$ ist.

- Ein Gegenbeispiel ist $(\mathbb{Z} \setminus \{0\}, \cdot)$. Die Menge der ganzen Zahlen (ohne die 0) mit der Multiplikation als Gruppenoperation bildet keine Gruppe, da es kein Inverses Element a^{-1} für jedes Element $a \in \mathbb{Z}$ gibt [6, S. 239, 240].

2.2.3. Zyklische Gruppe

Die eben eingeführten Gruppen besitzen unendlich viele Elemente. Für die Kryptographie interessant sind jedoch endliche algebraische Strukturen, weshalb im Folgenden die endlichen Gruppen eingeführte werden.

Definition: Endliche Gruppe Eine Gruppe $(G, \circ)$ ist *endlich*, wenn sie eine endliche Anzahl an Elementen hat. Die Anzahl der Elemente der Gruppe wird als *Kardinalität* oder *Ordnung* der Gruppe G mit $|G|$ bezeichnet[6, S. 241].
Beispiele für endliche Gruppen sind:

- $(\mathbb{Z}_n, +)$: Die Kardinalität von $\mathbb{Z}_n$ ist $|\mathbb{Z}_n| = n$, da $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$.

- $(\mathbb{Z}_n^*, \cdot)$: Die Menge $\mathbb{Z}_n^*$ besteht aus den positiven Zahlen kleiner n, die teilerfremd zu n sind, d.h. es gilt $ggt(a, n) = 1$ für jedes $a \in \mathbb{Z}_n^*$. Die Kardinalität ist daher durch die eulersche Phi-Funktion gegeben, d.h. $|\mathbb{Z}_n^*| = \Phi(n)$. So hat beispielsweise die Gruppe $\mathbb{Z}_9^*$ eine Kardinalität von $\Phi(9) = 3^2 - 3^1 = 6$. Die sechs Gruppenelemente sind $1, 2, 4, 5, 7, 8$.

Für die Konstruktion eines DLPs wird eine weitere Spezialisierung der endlichen Gruppen benötigt, die sogenannten zyklischen Gruppen. Einleitend dazu wollen wir zunächst den Begriff der Ordnung eines Elements definieren.

Definition: Ordnung eines Elements Die *Ordnung* $ord(a)$ eines Elements a einer Gruppe $(G, \circ)$ ist die kleinste positive ganze Zahl k mit

$$a^k = \underbrace{a \circ a \circ \cdots \circ a}_{k \text{ mal}} = e$$

wobei e neutrales Element von G ist [6, S. 241].
Nachfolgend wollen wir ein Beispiel betrachten:

Wir suchen die Ordnung von $a = 3$ in der Gruppe $\mathbb{Z}_{11}^*$. Dazu berechnen wir die Potenzen von a bis wir das neutrale Element 1 erhalten.

$$\begin{aligned}
a^1 &= 3 \\
a^2 &= a \cdot a = 9 \\
a^3 &= a^2 \cdot a = 27 \equiv 5 \bmod 11 \\
a^4 &= a^3 \cdot a = 15 \equiv 4 \bmod 11 \\
a^5 &= a^4 \cdot a = 12 \equiv 1 \bmod 11
\end{aligned}$$

Aus der letzten Zeile folgt $\text{ord}(3) = 5$ in $\mathbb{Z}_{11}^*$.
Es ist interessant zu sehen, was passiert, wenn man weiter mit a multipliziert:

$$\begin{aligned}
a^6 &= a^5 \cdot a = 1 \cdot a \equiv 3 \bmod 11 \\
a^7 &= a^5 \cdot a^2 = 1 \cdot a^2 \equiv 9 \bmod 11 \\
a^8 &= a^5 \cdot a^3 = 1 \cdot a^3 \equiv 5 \bmod 11 \\
a^9 &= a^5 \cdot a^4 = 1 \cdot a^4 \equiv 4 \bmod 11 \\
a^{10} &= a^5 \cdot a^5 = 1 \cdot a^5 \equiv 1 \bmod 11 \\
a^{11} &= a^{10} \cdot a = 1 \cdot a \equiv 3 \bmod 11 \\
&\vdots
\end{aligned}$$

Wie zu sehen ist, durchlaufen die Potenzen von a nach Erreichen des neutralen Elements e immer wieder die Sequenz $\{3, 9, 5, 4, 1\}$. Mit diesem Wissen können wir jetzt eine zyklische Gruppe definieren [6, S. 241, 242].

Definition: Zyklische Gruppe Eine Gruppe G, die ein Element α mit der maximalen Ordnung $\text{ord}(\alpha) = |G|$ enthält nennt man *zyklisch*. Elemente mit maximaler Ordnung nennt man *primitive Elemente* oder *Generatoren* [6, S. 242].

Der Name *Generator* kommt daher, dass durch die Potenzierung $\alpha^i = a$ des primitiven Elements jedes andere Gruppenelement a dargestellt werden kann, also die gesamte Gruppe *generiert* wird. Das folgende Beispiel zeigt die Erzeugung der zyklischen

Gruppe $\mathbb{Z}_{11}^*$ durch das primitive Element $\alpha = 2$.

$$\begin{aligned} \alpha &= 2 & \alpha^6 &\equiv 9 \bmod 11 \\ \alpha^2 &= 4 & \alpha^7 &\equiv 7 \bmod 11 \\ \alpha^3 &= 8 & \alpha^8 &\equiv 3 \bmod 11 \\ \alpha^4 &\equiv 5 \bmod 11 & \alpha^9 &\equiv 6 \bmod 11 \\ \alpha^5 &\equiv 10 \bmod 11 & \alpha^{10} &\equiv 1 \bmod 11 \end{aligned}$$

Aus der letzten Gleichung folgt, dass $\text{ord}(a = 2) = 10 = |\mathbb{Z}_{11}^*|$. Damit ist gezeigt, dass das Element $a = 2$ wirklich ein Generator der zyklischen Gruppe $\mathbb{Z}_{11}^*$ ist.
Wie man aus den eben gezeigten Beispielen erkennen kann, sind einige Gruppenelemente Generatoren der Gruppe und andere nicht. Welche Ordnungen der Elemente in einer zyklischen Gruppe vorkommen hängt davon ab, welche natürlichen Zahlen die Gruppenkardinalität teilen, da die Ordnung eines Elements immer die Gruppenkardinalität teilt. Betrachten wir nun wieder die Gruppe $\mathbb{Z}_{11}^*$, deren Kardinalität $|\mathbb{Z}_{11}^*| = 10$ ist, so ergeben sich mögliche Ordnungen der Elemente von $1, 2, 5$ und 10, da diese die einzigen natürlichen Zahlen sind die 10 teilen. Folgende Veranschaulichung zeigt dies [6, S. 243, 244].

$$\begin{aligned} \text{ord}\ (1) &= 1 & \text{ord}\ (6) &= 10 \\ \text{ord}\ (2) &= 10 & \text{ord}\ (7) &= 10 \\ \text{ord}\ (3) &= 5 & \text{ord}\ (8) &= 10 \\ \text{ord}\ (4) &= 5 & \text{ord}\ (9) &= 5 \\ \text{ord}\ (5) &= 5 & \text{ord}\ (10) &= 2 \end{aligned}$$

Satz: Primitive Elemente in einer zyklischen Gruppe G Ist G eine zyklische Gruppe, dann gilt:

1. Die Anzahl der primitiven Elemente in G ist $\Phi(|G|)$.
2. Ist $|G|$ prim, dann sind alle Elemente $a \neq 1 \in G$ primitiv [6, S. 244].

Die erste Eigenschaft kann leicht gezeigt werden durch, $\Phi(10) = (5-1)(2-1) = 4$. Es muss also vier primitive Elemente in $\mathbb{Z}_{11}^*$ geben, welche konkret die Elemente $2, 6, 7$ und 8 sind. Die zweite Eigenschaft folgt implizit aus der ersten, da bei einer

primen Kardinalität der Gruppe keine natürliche Zahl außer der 1 und der Kardinalität selber die Gruppenkardinalität teilt, und diese somit die einzigen möglichen Elementordnungen sind. Da nur das Element 1 die Ordnung 1 haben kann, haben alle anderen Elemente die Ordnung $|G|$, sind also Generatoren der Gruppe [6, S. 244].

2.2.4. Untergruppen

Untermengen innerhalb zyklischer Gruppe können wiederum selbst Gruppen sein. Solche werden als Untergruppen bezeichnet. Im Fall von zyklischen Gruppen ist die Anzahl der jeweiligen Untergruppen leicht zu ermitteln. Sie entspricht nämlich genau der Anzahl an natürlichen Teilern der Gruppenkardinalität, wie aus dem folgenden Satz von Lagrange hervorgeht.

Satz: Satz von Lagrange Sei H eine Untegruppe von G. Dann teilt $|H|$ die Gruppenkardinalität $|G|$ [6, S. 245].
Das *Finden* zyklischer Untergruppen innerhalb einer zyklischen Gruppe wird durch folgenden Satz gezeigt.

Satz: zyklische Untergruppen Sei $(G, \circ)$ eine zyklische Gruppe. Dann ist jedes Element $a \in G$ mit $\text{ord}(a) = s$ ein primitives Element einer zyklischen Untergruppe mit s Elementen [6, S. 244].

Die Kernaussage dieses Satzes ist, dass jedes Element einer zyklischen Gruppe Generator einer zyklischen Untergruppe ist. Wir betrachten erneut die zyklische Gruppe $\mathbb{Z}_{11}^*$. Wie oben bereits gezeigt hat das Gruppenelement $a = 3$ die Ordnung 5, ist also Generator einer Untergruppe mit mit 5 Elementen. Die Elemente dieser Untergruppe sind eben jene, welche durch die Potenzierung von 3 mod 11 erzeugt werden können, also $\{3, 9, 5, 4, 1\}$. So kann für jedes Element verfahren werden, alle Untergruppen von $\mathbb{Z}_{11}^*$ gezeigt werden können. Folgende Aufstellung zeigt beispielhaft alle Untergruppen von $\mathbb{Z}_{11}^*$.

Ordnung: 1 Generatoren: $\{1\}$ Elemente: $\{1\}$

Ordnung: 2 Generatoren: $\{10\}$ Elemente: $\{1, 10\}$

Ordnung: 5 Generatoren: $\{3, 4, 5, 9\}$ Elemente: $\{1, 3, 4, 5, 9\}$

Ordnung: 10 Generatoren: $\{2, 5, 7, 8\}$ Elemente: $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

Wie zu erkennen ist, kann es mehrere Generatoren der selben Untergruppe geben. Außerdem interessant ist die Tatsache, dass Untergruppen primer Ordnung p abgesehen von dem neutralen 1 keine Überschneidungen mit anderen Untergruppen primer Ordnung aufweisen, da jedes Element die Ordnung p hat und somit Generator der Untergruppe ist. Dies verdeutlicht, dass der Satz *Primitive Elemente in einer zyklischen Gruppe* auch für zyklische Untergruppen gilt [6, S. 244-246].
Der folgende Satz fasst die Erkenntnisse des letzten Abschnitts zusammen und gibt eine Methode zur Konstruktion einer Untergruppe für eine gegebene zyklische Gruppe.

Satz: Konstruktion einer zyklischen Untergruppe Sei G eine endliche zyklische Gruppe der Ordnung n und sei α ein Generator von G. Dann existiert für jede ganze Zahl k, die n teilt, genau eine zyklische Untergruppe H von G mit der Ordnung k. Diese Untergruppe wird erzeugt von $\alpha^{n/k}$. H besteht genau aus den Elementen $a \in G$, die die Bedingung $a^k = 1$ erfüllen. Es gibt keine weiteren Untergruppen [6, S. 246].

Mit anderen Worten sagt dieser Satz, dass lediglich ein primitives Element α und die Ordnung einer zyklischen Gruppe $|G|$ benötigt wird, um alle Untergruppen von G zu konstruieren. Im folgenden Beispiel wollen wir dies zu Konstruktion einer Untergruppe anwenden.

Betrachten wir dazu erneut die zyklische Gruppe $\mathbb{Z}_{11}^*$ und das primitive Element $\alpha = 6$. Wenn wir nun eine Generator β der Untergruppe mit der Ordnung $k = 5$ finden möchten, berechnen wir:

$$\beta = \alpha^{n/k} = 6^{10/5} = 6^2 = 36 \equiv 3 \bmod 11$$

Wie wir oben schon gezeigt haben ist 3 tatsächlich ein Generator der Untergruppe $\{1, 3, 4, 5, 9\}$ mit $k = 5$ Elementen[6, S. 246].

2.2.5. Ring

Ein **Ring** ist eine algebraische Struktur $(R, 0, 1, +, \cdot, -)$ des Typs $(0, 0, 2, 2, 1)$ mit den folgenden Eigenschaften:

1. Es ist $(R, 0, +, -)$ eine kommutative Gruppe
2. Es ist $(R, 1, \cdot)$ ein Monoid.

3. Für alle $x, y, z \in R$ gelten die Distributivgesetze $x(y + z) = xy + xz$ und $(y + z)x = yx + zx$

Ist $(R, 1, \cdot)$ ein kommutatives Monoid, so nennt man $(R, 0, 1, +, \cdot, -)$ einen kommutativen Ring.

2.2.6. Körper

Ein Körper ist eine spezielle Form des Rings. Die Definition des Körpers wurde aus [5, S. 364-365] entnommen. Ein Ring $(K, 0, 1, +, \cdot, -)$ heißt **Körper**, wenn er kommutativ ist sowie gilt, dass $0 \neq 1$. Weiterhin muss folgende Formel

$$x \neq 0 \Rightarrow \exists y \in K : yx = 1$$

für alle $x \in K$ gelten.

Berghammer beschreibt in seinem Buch ein Stilbruch bei der Definition der Körper. Im Vergleich zu den Monoiden, Gruppen und Ringen erfordert der Körper eine Ungleichung als Axiom. Weiterhin existiert ein noch komplizierteres Axiom, nämlich eine Implikation mit einer Ungleichung als linker und einer Existenzquantifizierung als rechter Seite. Weiterhin ist im Gegensatz zu den Gruppen das linksinverse Element y zu x im zweiten Axiom bezüglich der Multiplikation nicht durch eine Spezifikation, sondern durch eine Existenzquantifizierung spezifiziert. Dadurch spezifiziert Berghammer Körper als algebraische Strukturen, welche nicht gleichungsdefiniert sind [5, S. 364-365] Der Satz $x \neq 0$ impliziert, dass es nur ein linksinverses Element geben kann. Aufgrund der Kommutativität von „$\cdot$“ ist dieses linksinverse Element auch rechtsinvers.

Desweiteren haben Körper weitere Eigenschaften. Diese sind dem Buch über elementare Algebra und Zahlentheorie von Stroth und Waldecker entnommen [7, S. 57-59].Sei K ein Körper. Dann gilt

1. Die kleinste natürliche Zahl n mit der Eigenschaft $\underbrace{1_K + ... + 1_K = 0_K}_{n-mal}$ wird die **Charakteristik** von K genannt. Mann schreibt das als $\text{char}(K) = n$. Gibt es keine solche Zahl n, so schreibt man $\text{char}(K) = 0$.

2. Ein Teilkörper von K ist ein Teilring, der mit Einschränkung der Verknüpfungen selbst ein Körper ist.

3. Der Durchschnitt aller Teilkörper von K heißt **Primkörper** von K.

4. Zwei Körper heißen **isomorph** genau dann, wenn es einen bijektiven Ringhomomorphismus von dem einen in den anderen gibt.

Wie man sieht, ist ein Körper ein sehr komplexes Gebilde. Einige bekannte Körper sind die Mengen $\mathbb{Q}$, $\mathbb{R}$ und $\mathbb{C}$ mit der Multiplikation und der Addition innerhalb der Körper. Unter den Körpern befinden sich auch die endlichen Körper. Endliche Körper hat eine besondere Eigenschaft. Diese bezieht sich auf die Endlichkeit und der Anzahl ihrer Elemente. Die folgenden Definitionen und Aussagen sind dem Script zur Kryptografie von Reinhold Hübl entnommen [4, S. 266].

Definition: Endliche Körper und Primkörper Sei K ein Körper $(K, +, \cdot)$. K ist ein endlicher Körper, wenn $|K| < \infty$. Ist p eine Primzahl, so schreibt man $\mathbb{F}_p$ für den Körper $\mathbb{Z}/p\mathbb{Z}$. Für eine Primzahl p heißt der Körper $\mathbb{F}_p$ **Primkörper** der Charakteristik p. Dabei wird $\mathbb{F}_p$ durch folgende Elemente beschrieben:

$$\mathbb{F}_p = \{1, ..., p-1\}$$

Alle Elemente in $\mathbb{F}_p$ sind teilerfremd zu p. Daher sind alle Äquivalenzklassen

$$[1], [2], ..., [p-1]$$

Einheiten in $\mathbb{F}_p$. Durch $x \in \mathbb{F}_p \backslash \{0\}$ hat damit jedes Element ein Inverses. Damit wurde eine weitere wichtige Eigenschaft von Körpern gefunden, nämlich dass jedes Element in diesem Körper außer der Zahl Null ein Inverses in Bezug auf die Multiplikation hat. Der folgende Satz wurde aus dem Script von Hübl entnommen. Es handelt sich um einen Satz von Fermat [4, S. 269].

Satz: Die Einheitengruppe $\mathbb{F}_p^*$ eines endlichen Körpers $\mathbb{F}_p$ ist zyklisch.

Im Kapitel 2.2.2 wurde das Inverse definiet. Für die Rechnungen in einem endlichem Körper $\mathbb{F}_p$ können keine rationale Zahlen $\mathbb{Q}$ dargestellt werden. In manchen Rechnungen kommt man jedoch nicht darum, mit Brüchen oder etwaigen rationalen Zahlen zu hantieren. Damit man jedoch eindeutige Ergebnisse in $\mathbb{F}_p$ bekommt, muss man den Kehrwert über das Inverse bilden. Nimmt man als Beispiel die Zahl $\frac{3}{11}$ in $\mathbb{F}_{17}$. Da es keine rationalen Zahlen in der zyklischen Gruppe gibt, muss man anders an die Werte kommen. Als erstes schreibt man um: $\frac{3}{11} = 3 \cdot \frac{1}{11}$. Jetzt berechnet man das inverse Element von 11 in $\mathbb{F}_{17}$. Das inverse Element ist die Zahl 14, da

$11 \cdot 14 \equiv 1 \bmod 17$. Man rechnet $\frac{3}{11} = 3 \cdot \frac{1}{11} = 3 \cdot 14 = 42 \equiv 8 \bmod 17$.

Das Inverse berechnet man üblicherweise mit dem erweiterten euklidischen Algorithmus oder Potenzregeln mit Zuhilfenahme des kleinen Satzes von Fermat. Für diese Studienarbeit wird jedoch der erweiterte euklidische Algorithmus betrachtet. Mit diesem kann man das multiplikative Inverse der Zahl a^{-1} der Zahl $a \neq 0$ bestimmen, also den Kehrwert von a. Um den erweiterten euklidischen Algorithmus durchzuführen, muss im Vorfeld der reguläre euklidische Algorithmus durchgeführt worden sein. Der euklidische Algorithmus berechnet den größten gemeinsamen Teiler von zwei Zahlen a und b. In einer Internetveröffentlichung der Hochschule für Technik in Stuttgart wird die Grundfunktionsweise des euklidischen Algorithmus in einem Satz auf den Punkt gebracht: *Teilen so lange mit Rest, bis der Rest 0 erreicht* [**?**, S. 2]. Bei dem Iterationsschritt vor dem Schritt, bei welchem die 0 erreicht wurde, ist der Rest der $ggt(a, b)$.

Als erstes legt man zwei Zahlen a und b fest, für welche der euklidische Algorithmus durchgeführt werden soll. Im ersten Schritt teilt man die kleinere Zahl durch die größere und schreibt den Rest auf. Im nächsten Schritt teilt man die kleinere Zahl durch den Rest und schreibt dann wieder den Rest auf. Dies wird solange wiederholt, bis die 0 erreicht ist. Die Vorgehensweise soll anhand von zwei Beispielen erfolgen. Als erstes Beispiel wird $a = 174$ und $b = 102$ gesetzt.

$$
\begin{aligned}
174 &= 1 \cdot 102 + 72 \\
102 &= 1 \cdot 72 + 30 \\
72 &= 2 \cdot 30 + 12 \\
30 &= 2 \cdot 12 + 6 \\
12 &= 2 \cdot 6 + 0
\end{aligned}
$$

Als erstes haben wir eine Gleichung aufgestellt. Die Zahl 174 setzt sich aus $1 \cdot 102$ und dem Rest 72 zusammen. Im nächsten Schritt nimmt man die größte Zahl und stellt sie links von der Gleichung, in diesem Fall die Zahl 102. Man prüft, wie oft der vorherige Rest 72 in die 102 reinpasst. Der Rest ist die Zahl 30. Die Zahl 102 wird nun durch 72 substituiert. Der vorherige Rest 30 passt zwei mal in die 72. Der Rest ist 12. Die 30 wird im nächsten Iterationsschritt links der Gleichung gesetzt. Der vorherige Rest 12 passt zwei mal in die Zahl 30 rein und es ergibt sich ein Rest von 6. Im nächsten Iterationsschritt setzen wir die 12 links in die Gleichung ein. Der vorherige Rest 6 passt genau zwie mal in die 12 rein. Da sich keiN Rest ergibt, ist der Algorithmus vollständig durchgelaufen. Im vorletzten Iterationsschritt kann man

den größten gemeinsamen Teiler ablesen. Der durchgeführte euklidische Algorithmus ergibt, dass der $ggt(174, 102)$ die Zahl 6 ist.

Hübl beschreibt den euklidischen Algorithmus in seinem Script über eine allgemeine Formel [4, S. 256-257]. Gegeben sei a und b mit $b \geq a$. Sei $i = 0$, $r_1 = a$ und $r_0 = b$. Man dividiert r_i durch r_{i+1} mit Rest durch folgende Formel:

$$r_i = q \cdot r_{i+1} + c$$

Dabei ist q eine natürliche Zahl und der Rest $c \in \{0, 1, ..., r_{i+1} - 1\}$. Falls $b = 0$, dann ist der Algorithmus fertig. Falls $b \neq 0$, setzt man $r_{i+2} = c$ und $i = i + 1$. Dies wird so oft wiederholt, bis der Rest 0 ist und der Algorithmus durchgelaufen ist. Der größte gemeinsame Teiler ist in beiden Fällen der Rest im vorletzten Iterationsschritt vor dem Erreichen der 0.

Nun wird ein etwas längeres Beispiel betrachtet. Man setzt $a = 575$ und $b = 839$. Dadurch ist $i = 0$, $r_0 = a = 839$ und $r_1 = b = 575$.

$i = 0$: $839 = 1 \cdot 575 + 264$. Man setzt $r_2 = 264$

$i = 1$: $575 = 2 \cdot 264 + 47$. Man setzt $r_2 = 47$

$i = 2$: $264 = 5 \cdot 47 + 29$. Man setzt $r_2 = 29$

$i = 3$: $47 = 1 \cdot 29 + 18$. Man setzt $r_2 = 18$

$i = 4$: $29 = 1 \cdot 18 + 11$. Man setzt $r_2 = 11$

$i = 5$: $18 = 1 \cdot 11 + 7$. Man setzt $r_2 = 7$

$i = 6$: $11 = 1 \cdot 7 + 4$. Man setzt $r_2 = 4$

$i = 7$: $7 = 1 \cdot 4 + 3$. Man setzt $r_2 = 3$

$i = 8$: $4 = 1 \cdot 3 + 1$. Man setzt $r_2 = 1$

$i = 9$: $3 = 3 \cdot 1 + 0$. $\longrightarrow$ **STOPP**

Die durchgeführte Rechnung ergibt, dass der $ggt(839, 575)$ die Zahl 1 ist.

Der euklidische Algorithmus liefert nur den größten gemeinsamen Teiler und nicht das Inverse. Um das Inverse zu erlangen, muss der euklidische Algorithmus zurückgerechnet werden. Diese Rückrechnung wird auch als der erweiterte euklidische Algorithmus bezeichnet. Dabei steht der Rest auf der linken Seite der Gleichung und man rechnet den euklidischen Algorithmus zurück, damit eine Gleichung der folgenden Form

entsteht: $g = x \cdot a + y \cdot b$, wobei $a, b \in \mathbb{N}\ \{0\}$ und $ggt(a, b) = g$ [4, S. 257]. Folgend nun die Rückwärtsrechnung für das erste Beispiel.

$$\begin{aligned}
6 &= 30 - 2 \cdot 12 \\
&= 30 - 2 \cdot (72 - 2 \cdot 30) = 5 \cdot 30 - 2 \cdot 72 \\
&= 5 \cdot (102 - 72) - 2 \cdot 72 = 5 \cdot 102 - 7 \cdot 72 \\
&= 5 \cdot 102 - 7 \cdot (174 - 102) = 12 \cdot 102 - 7 \cdot 174
\end{aligned}$$

Damit wurde die Gleichung, die den Rest beschreibt gefunden. Diese ist

$$6 = 12 \cdot 102 - 7 \cdot 174$$

Diese kleine Beispielrechnung für den erweiterten euklidischen Algorithmus dient lediglich der Veranschaulichung der Funktionsweise. Da der Rest 6 ist hat 174 kein multiplikatives Inverses Modulo 102. Nun wird der erweiterte euklidische Algorithmus für das zweite Beispie angewendet, in welchem der Rest 1 ist und somit ein multiplikatives Inverses von 575 Modulo 839 existiert.

$$\begin{aligned}
1 &= 4 - 3 \\
&= 4 - (7 - 4) \\
&= 2 \cdot (11 - 7) - 7 = 2 \cdot 11 - 3 \cdot 7 \\
&= 2 \cdot 11 - 3 \cdot (18 - 11) = 5 \cdot 11 - 3 \cdot 18 \\
&= 5 \cdot (29 - 18) - 3 \cdot 18 = 5 \cdot 29 - 8 \cdot 18 \\
&= 5 \cdot 23 - 8 \cdot (47 - 29) = 13 \cdot 29 - 8 \cdot 47 \\
&= 13 \cdot (264 - 5 \cdot 47) - 8 \cdot 47 = 13 \cdot 264 - 73 \cdot 47 \\
&= 13 \cdot 264 - 73 \cdot (575 - 2 \cdot 264) = 159 \cdot 264 - 73 \cdot 575 \\
&= 159 \cdot (839 - 575) - 73 \cdot 575 = 159 \cdot 839 - 232 \cdot 575
\end{aligned}$$

Damit wurde die Gleichung, die den Rest beschreibt gefunden. Diese ist

$$1 = 159 \cdot 839 - 232 \cdot 575$$

Das Inverse wird nun wie folgt gefunden:

$$\begin{aligned} 1 &\equiv x \cdot 575 \bmod 839 \\ x &\equiv \frac{1}{575} \bmod 839 \\ x &= -232 \equiv 607 \bmod 839 \end{aligned}$$

Das multiplikative Inverse von 575 Modulo 839 ist 607.

Bei der Berechnung der Punkte einer elliptischen Kurve wird oft das Inverse benötigt, da man durch die Formeln zur Berechnung der x-Werte und der y-Werte oft rationale Zahlen in Form von Brüchen bekommt. Durch das multiplikative Inverse kann eine rationale Zahl $\mathbb{Q}$ durch eine ganze Zahl $\mathbb{Z}$ dargestellt werden. Dies ist essentiell für die Rechnung in $\mathbb{F}_p$, da man nur mit ganzen Zahlen rechnet. Da die Rechnung von Hand sehr umständlich und unübersichtlich ist empfiehlt es sich, den euklidischen Algorithmus und den erweiterten euklidischen Algorithmus als Programm abzubilden. In Kapitel **??** wurde die beiden Algorithmen in Python umgesetzt.

2.3. Das diskreter Logarithmusproblem

Das DLP bildet die mathematische Grundlage für viele asymmetrische Kryptosysteme, wie den DHKE oder die Elgamal-Verschlüsselung. Etwas legere ausgedrückt könnte man das DLP als Baukasten für asymmetrische Kryptosysteme betrachten, da es für zyklische Gruppen verallgemeinert werden kann. Im Folgenden wollen wir zunächst das DLP in Primzahlkörpern betrachten. Hierzu greifen wir auf die in Kapitel 2.2.3 erläuterten Eigenschaften zyklischer Gruppen zurück.

Definition: Diskretes Logarithmusproblem (DLP) in $\mathbb{Z}_p^*$ Gegeben sind die endliche zyklische Gruppe $\mathbb{Z}_p^*$ der Ordnung $p-1$, ein primitives Element $\alpha \in \mathbb{Z}_p^*$ und ein weiteres Element $\beta \in \mathbb{Z}_p^*$. Das DLP ist das Problem, eine ganze Zahl x im Bereich $1 \leq x \leq p-1$ zu finden, sodass:

$$\alpha^x \equiv \beta \bmod p$$

Wie in Kapitel XY gezeigt, muss ein solches x existieren, da α ein primitives Element ist, was bedeutet dass jedes andere Gruppenelement durch die Potenzierung α^x

dargestellt werden kann. Das gesuchte x wird als *diskreter Logarithmus* von β zur Basis α bezeichnet. Formal beschrieben als:

$$x = \log_\alpha \beta \bmod p.$$

Anders als der gewöhnliche Logarithmus in $\mathbb{R}$ ist der diskrete Logarithmus in einer endlichen zyklischen Gruppe nicht effizient zu berechnen. Grund dafür sind die Eigenschaften endlicher zyklischer Gruppen. Denn wie in Kapitel XY gezeigt, generiert die Potenzierung eines primitiven Elements α die jeweilige zyklische Gruppe bzw. Untergruppe in einer nicht absehbaren Reihenfolge.

Für kleinere Gruppen lässt sich x natürlich durch stumpfes Ausprobieren sog. *brute Force* in einer kurzen Zeit ermitteln. Diskrete Logarithmen in ausreichend großen Gruppen sind jedoch sehr schwer zu ermitteln.

Um den diskreten Logarithmus x von $\alpha^x \equiv \beta \bmod p$ zu ermitteln muss ein potenzieller Angreifer jedes x zwischen 1 und $p-1$ als möglichen Exponenten ausprobieren. Der Erzeuger kann zum effizienten berechnen von β, jedoch den Square-and-Multiply-Algorithmus verwenden, was das DLP zu einer Einwegfunktion macht.

Neben dem trivialen brute Force Angriff auf das DLP gibt es weitere Verfahren, welche die Menge an möglichen x Werten einschränken. Der Babystep-Giantstep-Algorithmus reduziert die Komplexität des DLPs von $O(p-1)$ auf $O(\sqrt{p-1})$. Noch weiter kann die Komplexität durch den Pohling-Hellman-Algorithums eingeschränkt werden. Durch diesen Angriff kann bei bekannter Primfaktorzerlegung der Gruppenordnung n, die Komplexität des DLPs auf $O(\sqrt{q})$ reduzieren, wobei q der größte Faktor von n ist. Aus diesem Grund wählt man in der Praxis eine Gruppe mit primer Ordnung für das DLP. Da die Ordnung von Gruppen $\mathbb{Z}_p^*$ dem Wert $p-1$ entspricht, der offensichtlich nicht prim ist, greift man auf Untergruppen von $\mathbb{Z}_p^*$ mit primer Ordnung zurück.[6]

2.3.1. Das verallgemeinerte diskrete Logarithmusproblem

Wie oben bereits erwähnt kann das DLP in jeglichen zyklischen Gruppen definiert werden. Bis jetzt haben wir immer die zyklische Gruppe $\mathbb{Z}_p^*$ betrachtet. Das DLP kann folgendermaßen verallgemeinert werden:

Definition: Verallgemeinertes diskretes Logarithmusproblem Gegeben sei eine endliche zyklische Gruppe G mit der Gruppenoperation $\circ$ und der Kardinalität n. Wir betrachten ein primitives Element $\alpha \in G$ und ein weiteres Element $\beta \in G$. Das diskrete Logarihtmusproblem liegt darin, eine ganze Zahl x im Bereich $1 \leq x \leq n$ zu finden, sodass:

$$\beta = \underbrace{\alpha \circ \alpha \circ \cdots \circ \alpha}_{x \text{ mal}} = \alpha^x$$

Auch hier ist die Existenz von x sicher, da α als primitives Element die gesamte Gruppe generiert. Das DLP lässt sich zwar in jeglichen zyklischen Gruppen definieren, jedoch ist es in einigen nicht *schwierig* zu lösen, wodurch es keine Einwegfunktion mehr darstellt. Ein Beispiel dafür sind additive Gruppen $G = (\mathbb{Z}_p, +)$. Die x-fache Anwendung der Gruppenoperation auf einen Generator α zur Erzeugung eines Elements β kann als *Multiplikation* mod p dargestellt werden. Der gesuchte Faktor x lässt sich dann leicht durch die Multiplikation von β und $inv(\alpha)$ berechnen. Letzteres lässt sich effizient mittels des EEAs ermitteln.
Gruppen auf denen ein DLP für kryptografische Zwecke definiert werden kann sind:

1. Multiplikative Gruppen des Primkörpers $\mathbb{Z}_p$ oder Untergruppen von hoher Primzahlordnung. Der DHKE nutz klassischerweise diese Gruppen. Auch die Elgamal-Verschlüsselung und der digitale Signaturalgorithmus (DSA) nutzen diese Gruppen.

2. Zyklische Gruppen welche über elliptischen Kurven gebildet werden. Das DLP ist in diesen Gruppen besonders schwer zu lösen, da kein Algorithmus bekannt ist, der die Komplexität des DLPs stark reduziert, wie dies beispielsweise beim DLP in $\mathbb{Z}_p^*$ durch Siebmethoden möglich ist. Deshalb werden diese Gruppen in modernen Kryptosystemen zunehmend eingesetzt, mit dem Effekt, dass gleiche Sicherheit mit deutlich geringerer Schlüssellänge erzielt werden kann.

3. Multiplikative Gruppen von endlichen Körpern $GF(2^m)$ oder Untergruppen davon. Sie sind in der Praxis weniger verbreitet, und es nicht vollständig geklärt ob Angriffe auf das DLP in $GF(2^m)$ nicht effizienter sind als jene gegen das DLP in Primkörpern.

4. Hyperelliptische Kurven oder algebraische Varietäten, die auch Verallgemeinerungen von elliptischen Kurven sind. Trotz einiger Vorteile gegenüber den klassischen elliptischen Kurven, sind die hyperelliptischen Kurven in der Praxis nicht sehr verbreitet.

Es gab im Laufe der Zeit einige weitere Vorschläge für mögliche Gruppen, welche jedoch nicht weiter verfolgt wurden, da sich meist herausstellte, dass das DLP einfacher zu lösen ist.[6]

2.3.2. Diffie-Hellmann Key Exchange (DHKE)

Der DHKE wurde im Jahr 1976 von Whitfield Diffie und Martin Hellmann veröffentlicht. Damit ist er das älteste asymmetrsiche Verfahren überhaupt. Der DHKE bringt eine Lösung für das große Problem der symmetrischen Kryptografie: den Schlüsselaustausch. Er ermöglicht zwei Parteien den Austausch bzw. die Vereinbarung eines gemeinsamen Geheimnisses über einen unsicheren Kanal. Dabei nutzt der DHKE das im vorigen Kapitel 2.3 erläuterte DLP. Prinzipiell beruht der DHKE auf der Idee, dass das Potenzieren in $\mathbb{Z}_p^*$ mit primen p eine Einwegfunktion ist und die Exponentation kommutativ ist, d.h.

$$k = (\alpha^x)^y \equiv (\alpha^y)^x \bmod p.$$

Im Folgenden wird der DHKE in seiner Grundform erläutert. In der Praxis wird der DHKE oft in erweiterter Form verwendet um Angriffen entgegenzuwirken.
Das Ziel der Kommunikationspartner Alice und Bob ist es einen gemeinsamen geheimen Schlüssel über einen unsicheren Kanal zu vereinbaren. Unter Umständen gibt es auch eine dritte Partei, welche die öffentlichen Parameter für den Schlüsselaustausch festlegt, dies kann aber auch durch Alice und Bob selbst ausgehandelt werden. Der DHKE besteht genau genommen aus zwei Protokollen: dem Set-up-Protokoll und dem Hauptprotokoll, das den eigentlichen Schlüsselaustausch durchführt. Das Setup sieht folgendermaßen aus:

Diffie-Hellman-Set-up:

1. Wähle eine große Primzahl p
2. Wähle eine ganze Zahl $\alpha \in \{2, 3, ..., p-2\}$ mit einer hohen Ordnung, welche einen sehr großen Primteiler hat
3. Veröffentliche p und α

Dies sind die öffentlichen Parameter manchmal auch als *Domain-Parameter* bezeichnet, welche wie erwähnt auch vorgegeben sein können. Wenn Alice und Bob die öffentlichen Parameter p und α erhalten haben, können sie einen gemeinsamen geheimen Schlüssel k mit dem folgenden Protokoll berechnen:

Diffie-Hellman-Schlüsselaustausch:

Alice		**Bob**
Wähle $a = k_{pr,A} \in \{2, ..., p-2\}$		Wähle $b = k_{pr,B} \in \{2, ..., p-2\}$
Berechne $A = k_{pub,A} \equiv \alpha^a \bmod p$		Berechne $B = k_{pub,B} \equiv \alpha^b \bmod p$
	$\xleftarrow{k_{pub,A}=A}$	
	$\xrightarrow{k_{pub,B}=B}$	
$k_{AB} = k_{pub,B}^{k_{pr,A}} \equiv B^a \bmod p$		$k_{AB} = k_{pub,A}^{k_{pr,B}} \equiv A^b \bmod p$

Einem Angreifer ist es nicht möglich aus einem der öffentlichen Schlüssel den jeweiligen privaten Schlüssel zu berechnen, was sich auf das DLP zurückführen lässt. Der öffentliche Schlüssel

$$A = k_{pub,A} \equiv \alpha^a$$

wird aus α potenziert mit dem privaten Schlüssel a mod p berechnet. Das DLP ist es, mit bekanntem A und α den Exponenten a zu ermitteln. Dies ist wie in vorigen Kapitel 2.3 erläutert nicht effizient möglich.

Das beide Parteien am Ende den gleichen Schlüssel k_{AB} berechnet haben lässt sich wie folgt zeigen.

Alice berechnet:

$$k_{AB} \equiv B^a \equiv (\alpha^b)^a \equiv \alpha^{ab} \bmod p$$

Bob berechnet:

$$k_{AB} \equiv A^b \equiv (\alpha^a)^b \equiv \alpha^{ab} \bmod p$$

Es besitzen also beide den gleichen Schlüssel $k_{AB} \equiv \alpha^{ab} \bmod p$. Dieser kann im weiteren als Schlüssel für eine symmetrische Verschlüsselung wie z.B. AES dienen. Das folgende Beispiel zeigt den DHKE mit kleinen Zahlen. Als Domain-Parameter werden $p = 29$ und $\alpha = 2$ gewählt. Der Schlüsselaustausch läuft wie folgt ab:

Alice		**Bob**
Wähle $a = k_{pr,A} = 5$		Wähle $b = k_{pr,B} = 12$
$A = k_{pub,A} = 2^5 \equiv 3 \bmod 29$		Berechne $B = k_{pub,B} = 2^{12} \equiv 7 \bmod 29$
	$\xleftarrow{A=3}$	
	$\xrightarrow{B=7}$	
$k_{AB} \equiv B^a = 7^5 \equiv 16 \bmod 29$		$k_{AB} \equiv A^b = 3^{12} \equiv 16 \bmod 29$

Wie in der letzten Zeile zu sehen, haben beide Parteien erfolgreich $k_{AB} = 16$ berechnet.[6]

3. Elliptische Kurven

Als Basis für asymmetrische Kryptosysteme können elliptische Kurven dazu genutzt werden, die verschlüsselungstechnische Effektivität mathematischer Probleme, wie das des diskreten Logarithmus, zu erhöhen. Bei der Kryptographie unter Verwendung elliptischer Kurven, kann mit deutlich kürzerer Schlüssellänge ein gleichwertiges Ergebnis im Vergleich zum klassichen Diffie-Hellman-Key-Exchange oder der ElGamal-Verschlüsselung erzielt werden. Dieser Effekt wird durch die spezielle Arithmetik auf elliptischen Kurven erzielt, deren mathematische Grundlage, konkrete Eingenschaften und Funktionsweise im folgenden Kapitel erörtert werden soll.

Elliptische Kurven können über beliebigen Körpern definiert werden. Für die Kryptographie interessant sind elliptische Kurven über Primkörpern.

Um das weitere Verständnis zu verbessern, wollen wir erst eine uns schon bekannte Kurve ansehen. In Abbildung 3.1 ist die Lösungsmenge der Polynomgleichung $x^2 + y^2 = r^2$ über $\mathbb{R}$ dargestellt. Wie zu sehen ist, handelt es sich hierbei um die Kreisgleichung. Der zu sehende Kreis ist nichts anderes als die Menge aller Punkte, welche die Kreisgleichung erfüllen. Ein Beispiel für eine solchen ist der Punkt $(r, 0)$. Wenn x den Wert r hat, muss y folglich den Wert 0 haben. Ein Gegenbeispiel ist der Punkt $(r, r/2)$. Dieser erfüllt die Kreisgleichung nicht. Die Kreisgleichung kann

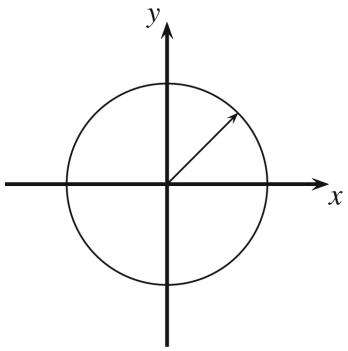

Abbildung 3.1.: Einheitskreis

verallgemeinert werden, indem den Termen x^2 und y^2 Koeffizienten voran gesetzt werden. Eine solche Gleichung, $ax^2 + by^2 = c$ erzeugt über $\mathbb{R}$ eine Ellipse, wie in Abbildung 3.2 zu sehen.

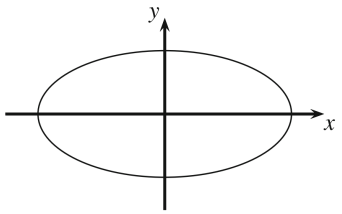

Abbildung 3.2.: Ellipse

Eine elliptische Kurve ist nun eine spezielle Polynomgleichung, der Form $y^2 = x^3 + ax + b$, unter der Bedingung $4a^3 + 27b^3 \neq 0$. Eine solche Gleichung über $\mathbb{R}$ ist in Abbildung 3.3 dargestellt. Damit elliptische Kurven sinnvoll in der Kryptologie

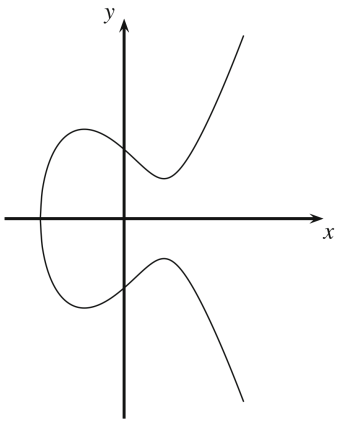

Abbildung 3.3.: Elliptische Kurve

eingesetzt werden können, muss die Polynomgleichung über einem endlichen Körper betrachtet werden. Dies können entweder Primkörper oder Erweiterungskörper dieser sein, wobei in dieser Arbeit nur Primkörper behandelt werden. Einfach gesprochen heißt das: alle Berechnungen werden modulo p durchgeführt.

Definition: Elliptische Kurven über Primkörpern Die *elliptische Kurve* über $\mathbb{F}_p$, ist die Menge aller Punkte (x, y) mit $x, y \in \mathbb{F}_|$, welche die folgende Gleichung erfüllen:

$$y^2 \equiv x^3 + ax + b \bmod p, \text{ wobei } a, b \in \mathbb{F}_p$$

und die Bedingung

$$4a^3 + 27b^2 \neq 0 \tag{3.1}$$

gelten müssen. Zu der elliptischen Kurve gehört des Weiteren auch der imaginäre *Punkt im Unendlichen* $\mathcal{O}$.

Durch die Bedingung 3.1 werden sog. Singularitäten ausgeschlossen. Andernfalls gäbe es Punkte, deren Tangente nicht eindeutig definiert ist, was für das Rechnen auf elliptischen Kurven jedoch erforderlich ist.[6, 273-276]

3.1. Arithmetik

Nachdem elliptische Kurven nun definiert wurden, stellt sich die Frage, wie diese in der Kryptographie eingesetzt werden können. Wenn wir uns an das in Kapitel 2.3 zurückerinnern, wird für die Konstruktion eines **DLP**s eine zyklische Gruppe benötigt. Eine eben solche findet sich in der Punktmenge der elliptischen Kurve wieder. Offen bleibt wie die Gruppenoperation definiert ist. Diese muss die in Kapitel 2.2.2 geforderten Gruppengesetze erfüllen.

Als Symbol für die Gruppenoperation wird das Additionszeichen + verwendet. Durch die Gruppenoperation muss aus zwei Punkten $P = (x_1, y_2)$ und $Q = (x_2, y_2)$ der Kurve ein dritter Punkt R auf der Kurve berechnet werden.

$$P + Q = R$$

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$

Am verständlichsten lässt sich diese Operation grafisch zeigen. Elliptische Kurven über endlichen Körpern können grafisch nicht sinnvoll dargestellt werden. Ihre Form und Arithmetik lassen sich jedoch gut veranschaulichen wenn man sie auf $\mathbb{R}$ abbildet. Im Folgenden betrachten wir eine Elliptische Kurve, dargestellt in einem kartesischen Koordinatensystem, um die Gruppeneigenschaften bezüglich der Punktaddition zu zeigen. Hierbei sind nun zwei Fälle zu unterscheiden.

Punktaddition $P + Q$: Falls $P \neq Q$ erfolgt die geometrische Konstruktion, indem zunächst eine Grade durch die beiden Punkte gelegt wird. Aufgrund der Kurveneigenschaften hat diese im Normalfall einen dritten Schnittpunkt mit der Kurve. Dieser wird an der x-Achse gespiegelt um den gesuchten Punkt R zu erhalten. Abbildung 3.4 zeigt die beschriebene Konstruktion.

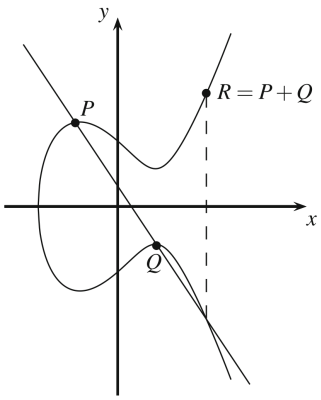

Abbildung 3.4.: Punktaddition

Punktverdopplung $P + P$: Falls P und Q identisch sind erfolgt die geometrische Konstruktion, indem eine Tangente an den Punkt P angelegt wird. Diese liefert wieder einen weiteren Schnittpunkt mit der Kurve, welcher an der x-Achse gespiegelt wird um den Punkt R zu erhalten. Anstatt $R = P + Q$ schreibt man in diesem Fall $R = P + P = 2P$ Abbildung 3.5 zeigt die beschriebene Konstruktion.

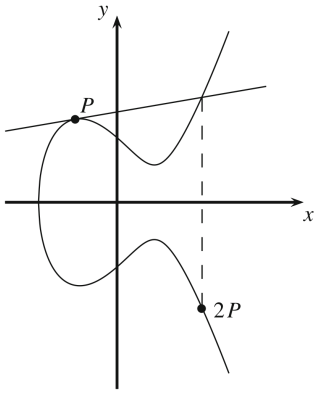

Abbildung 3.5.: Punktverdopplung

Sonderfall $P + (-P)$: In beiden genannten Fällen, Punktaddition und Punktverdopplung, kann es passieren, dass es keinen weiteren Schnittpunkt mit der Kurve gibt. Dieser Fall tritt für die Punktaddition ein, wenn die x-Koordinaten der Punkte identisch sind. Für die Punktverdopplung tritt der Fall ein, wenn die y-Koordinate Null ist. In beiden Situationen ergibt sich für die Schnittgerade bzw. Tangente, welche den genannten Schnittpunkt liefern soll, eine Parallele zur y-Achse. Diese schneidet die elliptische Kurve nicht mehr. Abbildung 3.6 zeigt den Sonderfall für die Punktaddition. Als Ergebnis in einem solchem Fall wird der *unendlich ferne Punkt* $\mathcal{O}$ definiert.

Dieser hat noch weitere Eigenschaften im Bezug auf die Gruppeneigenschaften, welche im Kapitel 3.2 genauer erläutert werden.

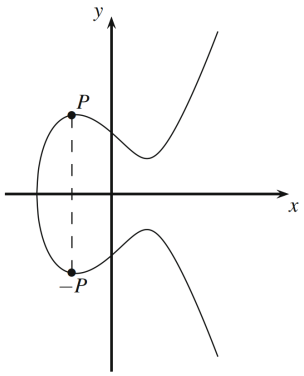

Abbildung 3.6.: Unendlich ferner Punkt

Nach dieser grafischen Veranschaulichung sollte es leichter fallen die folgenden Formeln für die Punktaddition bzw. Punktverdopplung nachvollziehen zu können. Die Gruppenoperation existiert in jedem Körper, weshalb die Berechnung von R, wie grade gezeigt über den reellen Zahlen $\mathbb{R}$, als auch über einem Primkörper $\mathbb{F}_p$ durchgeführt werden kann.[6, 276-280]

Die Formeln für die Punktaddition und - verdopplung auf elliptischen Kurven können anhand der grade gezeigten Veranschaulichung hergeleitet werden.

Herleitung: Formeln für Punktaddition bzw. Punktverdopplung Gegeben ist die Gleichung der elliptischen Kurve $y^2 = x^3 + ax + b$ und die Punkte $P = (x_1, y_1)$ und $Q = (x_2, y_2)$. Zunächst ist die Geradengleichung der Sekante durch P und Q zu ermitteln. Eine Grade im Allgemeinen hat die Form

$$g : y = sx + m.$$

Der Parameter s ist dabei die Steigung der Geraden und m ist der Schnittpunkt mit der y-Achse. Die Steigung s lässt sich unter der Bedingung $x_1 \neq x_2$ und $y_1 \neq y_2$ wie gewohnt durch Anlegen des Steigungsdreiecks berechnen, also mit der Formel

$$s = \frac{y_2 - y_1}{x_2 - x_1}.$$

Sollte $x_1 = x_2$ und $y_1 \neq y_2$ sein, ergibt sich durch Punktaddition der unendlich ferne Punkt $\mathcal{O}$, aufgrund der zur y-Achse parallelen Schnittgerade. Im Falle einer Punkt-

verdopplung muss s über die Tangente der Elliptischen Kurve im entsprechenden Punkt ermittelt werden. Dazu leiten leiten wir die Kurvengleichung der elliptischen Kurve nach x ab. Es ergibt sich

$$\frac{\delta y}{\delta x} = \frac{3x^2 + a}{2\sqrt{x^3 + ax + b}}.$$

Für den Fall das $y = 0$ ist, wird die Steigung unendlich groß, es ergibt sich als Ergebnis der Punktverdopplung also der unendlich ferne Punkt. Bei genauerer Betrachtung fällt auf, dass der Nenner genau $2y$ entspricht, weshalb auch

$$\frac{\delta y}{\delta x} = \frac{3x^2 + a}{2y}$$

geschrieben werden kann. Da die erste Ableitung die Steigung der elliptischen Kurve in jedem Punkt beschreibt, haben wir somit eine Formel mit welcher wird durch einsetzen eines Punktes $P = (x_1, y_1)$, die jeweilige Tangentensteigung ermitteln können. Es gilt also

$$s = \frac{3x_1^2 + a}{2y_1}.$$

Zur Bestimmung des Schnittpunkts mit der y-Achse kann nun einer der beiden Punkte P oder Q in die Geradengleichung $y = s \cdot x + m$ eingesetzt werden. Durch Einsetzen des Punkts $P = (x_1, y_1)$ erhalten wir die folgende Gleichung

$$y_1 = s \cdot x_1 + m,$$

welche nach m aufgelöst folgendermaßen aussieht:

$$m = y_1 - s \cdot x_1$$

Durch Einsetzten aller Parameter in die obige Geradengleichung ergibt sich

$$y = \frac{y_2 - y_1}{x_2 - x_1} \cdot x + y_1 - \frac{y_2 - y_1}{x_2 - x_1} \cdot x_1$$

für die gesuchte Gerade g durch die Punkte P und Q. Um den dritten Schnittpunkt dieser Geraden g mit der elliptischen Kurve E zu ermitteln, sind beide Kurven gleichzusetzen. Da es für das weitere Vorgehen keine Rolle spielt und es der Übersichtlichkeit dient, werden im Folgenden wieder die Parameter s und m statt eben gezeigten Konkretisierungen verwendet. Es ergibt sich die Gleichung

$$(sx + m)^2 = x^3 + ax + b.$$

Nach ausmultiplizieren der Gleichung erhalten wir

$$0 = x^3 - s^2x^2 + (a - 2sm) \cdot x - m^2 + b.$$

Im Normalfall ist das allgemeine Lösen eines solchen kubischen Polynoms nicht trivial. Wir haben hier jedoch den Vorteil, dass zwei der drei Schnittpunkte von g mit E schon bekannt sind. Fassen wir das Polynom als Funktion

$$f(x) = x^3 - s^2x^2 + (a - 2sm) \cdot x - m^2 + b$$

auf, so sind wir Im Grunde auf der Suche nach den Nullstellen dieser kubischen Funktion. Da die Punkte P und Q auf der Geraden g sowie auf der elliptischen Kurve E liegen, lösen sie die obige Gleichung, sind also Nullstellen der Funktion $f(x)$. Daraus folgt, dass die Funktion $f(x)$ restlos durch $(x - x_1)$ und $(x - x_2)$ geteilt werden kann um den Funktionsgrad zu verringern.

$$f(x) \div (x - x_1) \cdot (x - x_2) = l(x) \text{ Rest } 0$$

Aufgrund dessen, dass $f(x)$ den Grad 3 hat, muss $l(x)$ den Grad 1 haben und durch $l(x) = ux + v$ beschrieben werden können. Da

$$l(x) \cdot (x - x_1) \cdot (x - x_2) = f(x),$$

muss auch gelten

$$u \cdot x \cdot x \cdot x = x^3,$$

weshalb $u = 1$ gelten muss. Daraus ergibt sich

$$l(x) = x + v.$$

Somit ist $x_3 = -v$ eine weitere Nullstelle von $f(x)$. Es folgt also

$$f(x) = (x - x_1) \cdot (x - x_2) \cdot (x - x_3),$$

woraus durch ausmultiplizieren

$$f(x) = x^3 - (x_1 + x_2 + x_3) \cdot x^2 + (x_1x_2 + x_1x_3 + x_2x_3) \cdot x - x_1x_2x_3$$

entsteht. Vergleicht man nun diese Darstellung mit der obigen also

$$f(x) = x^3 - s^2x^2 + (a - 2sm) \cdot x - m^2 + b,$$

so folgt durch Koeffizientenvergleich

$$s^2 = x_1 + x_2 + x_3$$

also

$$x_3 = s^2 - x_1 - x_2.$$

Um die Formel für y_3 zu ermitteln kann nun der Punkt R in die Formel für s eingesetzt werden. Dies ist möglich, da der Punkt R auf der gleichen Geraden wie P und Q liegt, weshalb auch die Steigung s identisch ist. Es gilt also

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{y_3 - y_1}{x_3 - x_1}.$$

Aufgelöst nach y_3 ergibt sich

$$y_3 = s \cdot (x_3 - x_1) + y_1.$$

Da der Schnittpunkt an der x-Achse gespiegelt wird um R zu erhalten müssen die Vorzeichen in der Formel für y noch gedreht werden. Im Fall der Punktverdopplung kann genauso argumentiert werden, wobei hier $x_1 = x_2$ ist. Final ergeben sich für den Punkt $R = (x_3, y_3)$ also die Formeln:

$$x_3 = s^2 + x_1 - x_2$$

$$y_3 = s \cdot (x_1 - x_3) - y_1$$

Formel: Punktaddition und -verdopplung auf elliptischen Kurven[6, 278]:

$$x_3 = s^2 - x_1 - x_2$$

$$y_3 = s(x_1 - x_3) - y_1$$

, wobei

$$s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{, falls } P \neq Q \text{ (Punktaddition)} \\ \frac{3x_1^2 + a}{2y_1} & \text{, falls } P = Q \text{ (Punktverdopplung)} \end{cases}$$

Um sicherzugehen, dass die Berechnung korrekt durchgeführt wurde kann im Nachgang geprüft werden, ob der errechnete Punkt überhaupt auf der Kurve liegt, indem man diesen in die Gleichung der elliptischen Kurve einsetzt. Ist die Gleichung erfüllt, wurde wahrscheinlich richtig gerechnet.

Nachfolgend wollen wir ein Beispiel zur Punktaddition auf der elliptischen Kurve rechnen. Wir betrachten dazu die elliptische Kurve

$$E : y^2 \equiv x^3 + 2x + 2 \bmod 17.$$

Die Koeffizienten sind dementsprechend $a = 2$ und $b = 2$. Es gilt den Punkt $P = (5, 1)$ zu verdoppeln:

$$2P = P + P = (5, 1) + (5, 1) = (x_3, y_3)$$

Zunächst wird s mittels der Formel für die Punktverdopplung berechnet.

$$s = \frac{3x_1^2 + a}{2y_1} = (2 \cdot 1)^{-1} \cdot (3 \cdot 5^2 + 2) = 2^{-1} \cdot 9 \equiv 9 \cdot 9 \equiv 13 \bmod 17$$

Zur Berechnung von $2^{-1} \bmod 17$, also dem multiplikativem Inversen von 2 mod 17, wird der in Kapitel 2.2.6 behandelte erweiterte euklidische Algorithmus verwendet. Im letzten Schritt werden x_3 und y_3 berechnet.

$$\begin{aligned} x_3 &= s^2 - x_1 - x_2 = 13^2 - 5 - 5 = 159 \equiv 6 \bmod 17 \\ y_3 &= s(x_1 - x_3) - y_1 = 13(5 - 6) - 1 = -14 \equiv 3 \bmod 17 \\ 2P &= (5, 1) + (5, 1) = (6, 3) \end{aligned}$$

3.2. Gruppeneigenschaften

Zur Erfüllung der Gruppeneigenschaften wird außerdem ein neutrales Element $\mathcal{O}$ benötigt. Alle Punkte P der elliptischen Kurve müssen die Eigenschaft $P + \mathcal{O} = P$ erfüllen. Da kein Punkt der elliptischen Kurve diese Eigenschaft erfüllen kann, wird der imaginäre *unendlich ferne Punkt* als neutrales Element $\mathcal{O}$ definiert. Dieser Punkt liefert außerdem den *Schnittpunkt* im oben genannten Sonderfall, also falls ein Punkt P und der bezüglich der x-Achse gegenüberliegende Punkt $-P$ addiert werden. Die hier verwendete Terminologie $-P$ ist schon ein wenig voraus gegriffen, denn die Existenz eines neutralen Elements $\mathcal{O}$ ermöglicht nämlich die Definition eines Inversen

$-P$ für jeden Punkt P auf der Kurve. Laut der allgemeine Gruppengesetze gilt

$$P + (-P) = \mathcal{O}.$$

Folgend aus der im vorigen Kapitel 3.1 erläuterten Arithmetik auf elliptischen Kurven ist das Inverse eines Punktes $P = (x_p, y_p)$ der Punkt $-P = (x_p, -y_p)$, also der bezogen auf die x-Achse gegenüberliegende Punkt. In einem Primkörper berechnet sich die negative y-Koordinate durch $-y_p = p - y_p$.[6, 279]

3.3. Punktbestimmung

Die Arithmetik für Elliptische Kurven wurde bereits besprochen. Nun wird erarbeitet, wie man die Punkte einer elliptischen Kurve bestimmt. Doch was ist ein Punkt einer elliptischen Kurve? Im Foglenden wird erklärt, was ein Punkt auf einer elliptischen Kurve ist und wie diese berehcnet werden können. Die Erklärungen werden anschließend anhand einiger Beispiele näher erläutert. Anschließend wird Python-Code präsentiert, welcher die Punktbestimmung für eine elliptische Kurve mit $p > 3$ durchführt und die Punkte anschließend in der Konsole ausgibt.

3.3.1. Rechnerische Grundlagen

Viele Mathematiker suchten eine Formel, mit denen sich die Anzahl der Punkte einer elliptischen Kurve schätzen lässt, ohne dass man diese vorher aus- oder berechnen muss. Joseph H. Silverman beweist in seinem Buch einen mathematischen Satz aus der Zahlentheorie, welcher eine allgemeine Aussage über die Anzahl der rationalen Punkte auf einer elliptischen Kurve trifft [8, S. 138]. Dieser Satz kann also herangezogen werden, um eine ungefähre Abschätzung über die Anzahl der Punkte auf einer elliptischen Kurve über einem Primkörper p zu treffen. Bei dem mathematischen Satz handelt es sich um die Hasse–Weil–Schranke. Diese wird im allgemeinen dafür benutzt, um die Anzahl der Lösungen der Gleichung und der Bedingungen aber auch für die Einschränkung des Lösungsraums. Die Hasse-Weil-Schranke wird angelehnt an [4, S. 181] wie folgt beschrieben. Sei $k = \mathbb{F}_p$ ein endlicher Körper und $\overline{E}$ eine elliptische Kurve über k, dann gilt

$$p + 1 - 2 \cdot \sqrt{p} \leq |\overline{E}| \leq p + 1 + 2 \cdot \sqrt{p}$$

Was ist jedoch die Hauptaussage der Hasse-Weil-Schranke? Sie besagt, dass sich bei großen p die Anzahl der Elemente der elliptischen Kurve in der Größenoprdnung von p bewegen. Diese Schranke bildet hierbei eine obere und untere Grenze. Die Anzahl

der Punkte bewegt sich also innerhalb dieser Schranke. Dies ist für elliptische Kurven mit kleinem p uninteressant, jedoch wird diese Schranke für elliptische Kurven mit großem gewählten p, was in der Kryptographie gängig ist, relevant.

Doch wie lassen sich die Punkte konkret berechnen? Um diese Frage zu beantworten, müssen wir noch einmal die Grundlagen für elliptische Kurven aufgreifen. Wie Reinhold Hübl in seinem Manuskript der Kryptologie [4, S. 157] erläutert, ist eine elliptische Kurve mit den Charakteristiken $char(k) = 0$ oder $char(k) > 3$ eine Kurve, die durch ein Polynom der Form

$$F(X, Y) = Y^2 - X^3 - aX - b$$

mit $a, b \in k$ dargestellt ist, für die $4a^3 + 27b^2 \neq 0 \in k$ gilt.

Stellt man die Gleichung der Funktion nach y^2 um, dann erhält man folgende Gleichung in zwei Variablen:

$$y^2 = x^3 + ax + b \bmod p,$$

wobei $a, b \in \mathbb{F}_p$. Diese Gleichung ist der Schlüssel für die im Voraus aufgeworfene Frage, was ein Punkt einer elliptischen Kurve ist. Diese sind nämlich alle Punkte (x, y), welche die Gleichung lösen.

Um die Punkte auf einer Kurve zu berechnen, prüft man zunächst, ob es sich bei der betrachteten Kurve um eine elliptische Kurve mit $p > 3$ handelt. Dafür arbeitet man mit der Formel $4a^3 + 27b^2 \neq 0$. Man setzt die Parameter a und b der vermeintlichen elliptischen Kurve ein. Wenn die linke Seite $\neq 0$ ist, dann handelt es sich bei der besagten Kurve tatsächlich um eine elliptische Kurve mit $p > 3$. Rechnen wir dies nun einmal schematisch durch. Gegeben ist eine Funktion F mit

$$F(X, Y) = Y^2 - X^3 + 3X - 3 \in \mathbb{F}_{13}$$

Wie man der Funktion entnehmen kann ist $a = -3 = 10$ und $b = 3$ in $\mathbb{F}_{13}$. Diese setzen wir nun in die Formel ein.

$$4 \cdot 10^3 + 27 \cdot 3^2 = 6 \neq 0 \bmod(13)$$

Da $6 \neq 0$ ist definiert die Funktion eine elliptische Kurve in $\mathbb{F}_{13}$. Die Abbildung 3.7 zeigt die Zeichnung der Funktion dieser elliptischen Kurve über den reellen Zahlen im kartesischen Koordinatensystem:

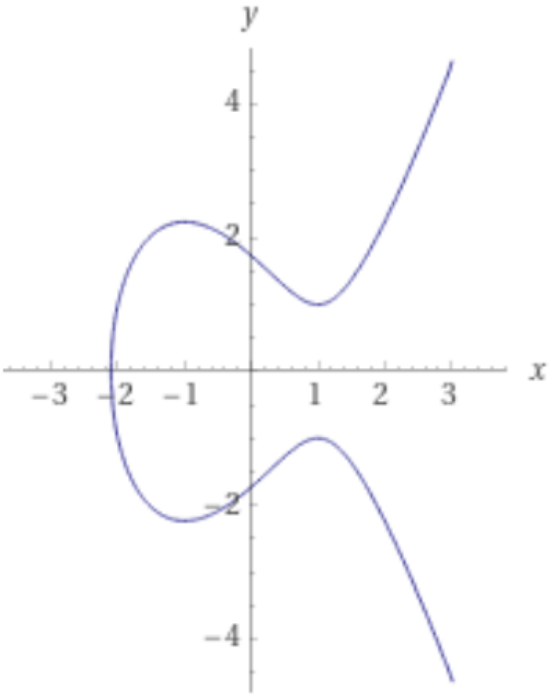

Abbildung 3.7.: Zeichnung der elliptischen Kurve
Quelle: Wolframalpha

Nachdem man allgemein geprüft hat, ob es sich bei der besagten Funktion um eine elliptische Kurve handelt, geht es jetzt um die Findung der Lösungen der umgestellten Gleichung, um alle Punkte zu finden. Dafür gibt es mehrere Möglichkeiten. Die Möglichkeiten haben unterschiedliche Zeit- und Rechenkomplexitäten, wodurch sich die Lösungsmöglichkeiten differenzieren lassen. Je nach Anwendungsfall lohnt sich die Implementierung einer anderen Lösung. Im Folgenden sind drei Möglichkeiten zur Berechnung aller Punkte auf elliptischen Kurven aufgezählt:

- Brute-Force
- Punktaddition und Punktverdopplung
- Algorithmischer Brute-Force

Neben dieser drei gängigen Methoden werden in der Mathematik und in der Kryptografie auch die Barrett-Reduktion und weiterhin auch eine Methode, bei welcher sich die Symmetrie zur x-Achse der elliptischen Kurve zur Berechnung zunutze gemacht wird, genutzt. Um diese soll es jedoch nicht gehen. In den folgenden zwei Unterkapiteln wird die Berechnung der Punkte über Brute-Force mit zwei Methoden sowie über die Punktaddition und -verdopplung erläutert und mittels Beispielen verständlich erklärt.

3.3.2. Punktbestimmung: Brute-Force-Methode

Wie bereits erläutert wurde, ist die umgestellte Gleichung der elliptischen Kurve in zwei Variablen folgende:

$$y^2 = x^3 + ax + b \in \mathbb{F}_p$$

Da die linke Seite mit y^2 offensichtlich quadratisch ist, muss am Ende auf beiden Seiten der Gleichung die Quadratwurzel gezogen werden, um die Gleichung zu lösen und im weiteren Sinne Punkte auf der Kurve zu finden. Da dies umständlich und im endlichen Körper $\mathbb{F}_p$ schwer zu realisieren ist, wird die rechte Seite bis zu einem Quadrat aufgelöst. Dafür muss man als erstes alle Punkte $x \in \mathbb{F}_p$ bestimmen, für die $x^3 + ax + b$ ein Quadrat in $\mathbb{F}_p$ sind. Die Quadratprüfung ist dabei simpel: Man geht jedes Element in $\mathbb{F}_p$ von 0 bis $p-1$ durch und prüft, ob das quadrierte Element in der Restklasse ein Quadrat ist. Hierbei geht es um keine komplexe Rechnung, sondern lediglich um stupides ausprobieren. Zur Verdeutlichung folgendes Beispiel: Wir haben den Körper $\mathbb{F}_5$. Der Körper beinhaltet die Elemente $\mathbb{F}_5 = \{0, 1, 2, 3, 4\}$. Die Quadrate mitsamt ihrer Wurzeln sind folgende:

$$0 = 0^2 \qquad 1 = 1^2 = 4^2 \qquad 4 = 2^2 = 3^2$$

Die Zahlen 0, 1 und 4 sind demnach in $\mathbb{F}_5$ Quadrate. Die Wurzeln wurden der Vollständigkeit halber ebenfalls obig rechts des Quadrates notiert. Man berechnet die Quadrate nach folgendem Schema:

1. Man hat k mit k $\in \mathbb{F}_p$
2. Man beginnt bei $k = 0$
3. Es wird quadriert mit k^2
4. Der Wert von k^2 in $\mathbb{F}_p$ wird ermittelt, dafür rechnet man k^2 mod(p)
5. Die ermittelte Zahl ist ein Quadrat in $\mathbb{F}_p$
6. Man wiederholt alle Schritte mit allen k von 0 bis $p-1$
7. Am Ende hat man alle Quadrate und ihre zugehörigen Wurzeln

Dies wird anhand eines ausführlichen Beispiels deutlich. Sei $p = 11$ und somit $\mathbb{F}_p = \mathbb{F}_{11}$.

- $0^2 \equiv 0 \bmod 11$

- $1^2 \equiv 1 \bmod 11$
- $2^2 \equiv 4 \bmod 11$
- $3^2 \equiv 9 \bmod 11$
- $4^2 \equiv 5 \bmod 11$
- $5^2 \equiv 3 \bmod 11$
- $6^2 \equiv 3 \bmod 11$
- $7^2 \equiv 5 \bmod 11$
- $8^2 \equiv 9 \bmod 11$
- $9^2 \equiv 4 \bmod 11$
- $10^2 \equiv 1 \bmod 11$

Wenn man die Null dazuzählt, hat $\mathbb{F}_{11}$ die Quadrate 0, 1, 3, 4, 5, 9. Im Folgenden sind die Quadrate mitsamt ihrer Wurzeln aufgeschrieben:

- Quadrat: 0 Wurzel 1: 0
- Quadrat: 1 Wurzel 1: 1 Wurzel 2: 10
- Quadrat: 3 Wurzel 1: 5 Wurzel 2: 6
- Quadrat: 4 Wurzel 1: 2 Wurzel 2: 9
- Quadrat: 5 Wurzel 1: 4 Wurzel 2: 7
- Quadrat: 9 Wurzel 1: 3 Wurzel 2: 8

Die berechneten Quadrate und Wurzeln sind nötig, um die Punkte zu bestimmen. Um dies zu bewerkstelligen, wird die Gleichung der elliptischen Kurve in zwei Variablen benötigt. Um die benötigte Formel zu erhalten, überführen wir die Ausgangsgleichung in zwei Variablen in eine Funktionsgleichung mit einer Variablen:

$$y^2 = x^3 + ax + b \in \mathbb{F}_p \qquad \Longrightarrow \qquad f(x) = x^3 + ax + b \in \mathbb{F}_p$$

Diese Funktionsgleichung ist der Ausgangspunkt für die Berechnung der Punkte. Man setzt nun alle x-Werte von 0 bis $p - 1$ in die Funktionsgleichung ein. Jene x-Werte, welche die Quadrate ergeben, werden für die Punktbestimmung benötigt. Nehme als Beispiel $p = 11$. Wir haben die Quadrate und die Wurzeln oben bereits ausgerechnet. Sei weiterhin die Gleichung einer elliptischen Kurve E gegeben durch

$$F(X,Y) = Y^2 - X^3 - 1 \cdot X - 3 \in \mathbb{F}_{11}$$

Damit ist die umgestellte Gleichung in zwei Variablen und die Funktionsgleichung mit einer Variablen

$$y^2 = x^3 + x + 3 \qquad \Longrightarrow \qquad f(x) = x^3 + x + 3 \in \mathbb{F}_{11}$$

Setzen wir nun alle Elemente von 0 bis $p - 1$ in die Funktionsgleichung ein und betrachten die Ergebnisse:

- $f(0) = 0^3 + 0 + 3 = 3 \in \mathbb{F}_{11} \qquad \Longrightarrow \qquad$ Quadrat
- $f(1) = 1^3 + 1 + 3 = 5 \in \mathbb{F}_{11} \qquad \Longrightarrow \qquad$ Quadrat
- $f(2) = 2^3 + 2 + 3 = 2 \in \mathbb{F}_{11}$
- $f(3) = 3^3 + 3 + 3 = 0 \in \mathbb{F}_{11} \qquad \Longrightarrow \qquad$ Quadrat
- $f(4) = 4^3 + 4 + 3 = 5 \in \mathbb{F}_{11} \qquad \Longrightarrow \qquad$ Quadrat
- $f(5) = 5^3 + 5 + 3 = 1 \in \mathbb{F}_{11} \qquad \Longrightarrow \qquad$ Quadrat
- $f(6) = 6^3 + 6 + 3 = 5 \in \mathbb{F}_{11} \qquad \Longrightarrow \qquad$ Quadrat
- $f(7) = 7^3 + 7 + 3 = 1 \in \mathbb{F}_{11} \qquad \Longrightarrow \qquad$ Quadrat
- $f(8) = 8^3 + 8 + 3 = 6 \in \mathbb{F}_{11}$
- $f(9) = 9^3 + 9 + 3 = 4 \in \mathbb{F}_{11} \qquad \Longrightarrow \qquad$ Quadrat
- $f(10) = 10^3 + 10 + 3 = 1 \in \mathbb{F}_{11} \qquad \Longrightarrow \qquad$ Quadrat

Die Ergebnisse zeigen dass für die folgenden x-Werte $f(x)$ ein Quadrat ist: 0, 1, 3, 4, 5, 6, 7, 9, 10. Bis jetzt wurde der einfache Brute-Force durchgeführt. Man muss nur noch wenige Schritte durchführen, um die Punkte auf der elliptischen Kurve zu bestimmen. Wir haben jetzt die Quadrate, die dazugehörigen Wurzeln sowie die x-Werte, welche Quadrate sind. Wie kombiniert man diese, um die Punkte auf der elliptischen Kurve zu erhalten? Es gilt, die Informationen der Punkte P mit $P = (x, y)$ der elliptischen Kurve zu erhalten. Für den x-Wert der Punkte nimmt man das Ergebnis des eingesetzten x von 0 bis $p - 1$ in die Funktion. Man nimmt als x-Wert das vorhandene Quadrat. Die y-Werte der Punkte ergeben sich aus dem Ergebnis des eingesetzten x in die umgestellte Funktion mit einer Variablen. Man nimmt die Wurzeln des zugehörigen Quadrates zum Ergebnis. Daraus ergeben sich die folgenden Punkte auf der obig eingeführten elliptischen Kurve E:

- $f(0) = 3 \implies$ Punkt 1: $(0, 5)$ Punkt 2: $(0, 6)$
- $f(1) = 5 \implies$ Punkt 1: $(1, 4)$ Punkt 2: $(1, 7)$
- $f(3) = 0 \implies$ Punkt 1: $(3, 0)$
- $f(4) = 5 \implies$ Punkt 1: $(4, 4)$ Punkt 2: $(4, 7)$
- $f(5) = 1 \implies$ Punkt 1: $(5, 1)$ Punkt 2: $(5, 10)$
- $f(6) = 5 \implies$ Punkt 1: $(6, 4)$ Punkt 2: $(6, 7)$
- $f(7) = 1 \implies$ Punkt 1: $(7, 1)$ Punkt 2: $(7, 10)$
- $f(9) = 4 \implies$ Punkt 1: $(9, 2)$ Punkt 2: $(9, 9)$
- $f(10) = 1 \implies$ Punkt 1: $(10, 1)$ Punkt 2: $(10, 10)$

Die Berechnung der Punkte über die Brute-Force-Methode ist sehr umständlich. Insbesondere, wenn das gewählte p groß ist, dann ist diese Brute-Force-Methode sehr aufwendig. Es müssen alle Quadrate in $\mathbb{F}_p$ und deren zugehörigen Wurzeln berechnet werden. Weiterhin entsteht Zeit- und Rechenaufwand beim Einsetzen in die Funktionen. In der Kryptografie ist jedoch die Wahl besonders großer p gängig, um ein hohes Sicherheitsniveau zu erreichen. Aus diesem Grund werden im Folgenden die beiden anderen angeführten Lösungsmöglichkeiten angeführt.

3.3.3. Punktbestimmung: Punktaddition und -verdopplung

Eine weitere Methode zur Berechnung von Punkten auf einer elliptischen Kurve ist die Punktaddition und die Punktverdopplung. Dies ist eine elegante Methode, da man über einen simplen Algorithmus alle Punkte auf einer elliptischen Kurve bestimmen kann, sofern der Startpunkt ein primitives Element der elliptischen Kurve ist. Der Nachteil dieser Methode ist jedoch, dass im Vorfeld ein Punkt auf der elliptischen Kurve bekannt sein muss, der gleichzeitig ein Erzeuger der Gruppe ist oder eine hohe Ordnung auf der Kurve hat. Dieser Punkt muss über die Brute-Force-Methode oder einem anderen Ansatz berechnet werden oder ein ist im Voraus bekannt. Die offene Frage ist jedoch, ob jeder beliebige Punkt als Ausgangsobjekt genutzt werden kann, um auf seiner Grundlage alle anderen Punkte zu erzeugen.

Um dies herauszufinden, müssen noch einmal die Gruppeneigenschaften von elliptischen Kurven über $\mathbb{F}_p$ betrachtet werden. Hankerson, Menezes und Vanstone

beschreiben in ihrem Buch „*Guide to Elliptic Curve Cryptography*“diese Gruppeneigenschaften und erklären damit, wie man herausfindet, ob es sich um eine zyklische Gruppe handelt und damit in welterem Sinne, welche Punkte auf der elliptischen Kurve Generatoren sind [9, S. 82-84]. Sei E eine elliptische Kurve über $\mathbb{F}_p$. Die Anzahl der Punkte in E($\mathbb{F}_p$), bezeichnet als $\overline{E}(\mathbb{F}_p)$, wird als die Ordnung von E über $\mathbb{F}_p$ bezeichnet. Dabei ist der imaginäre Punkt im Unendlichen inkludiert. Im Folgenden ist es wichtig herauszufinden, ob E eine zyklische Gruppe ist. Eine elliptische Kurve E über $\mathbb{F}_p$ ist eine zyklische Gruppe, wenn $\overline{E}(\mathbb{F}_p)$ eine Primzahl ist. Dabei gilt die Regel: Definiert $\overline{E}(\mathbb{F}_p)$ eine zyklische Gruppe, so ist die Anzahl der Punkte der Kurve mit hoher Wahrscheinlichkeit eine Primzahl. Dadurch ergibt sich ein Vorteil. Wenn die Anzahl der Punkte prim ist, dann sagt das Lagrange Theorem aus, dass jeder Punkt auf der Kurve ein Erzeuger der Gruppe ist. Dadurch kann aus jedem Punkt alle anderen Punkte berechnet werden. Um dies weiter auszuführen, wird die bereits angeführte Hasse-Weil-Schranke genutzt, um eine Abschätzung über die Anzahl der Punkte abzugeben.

$$p + 1 - 2 \cdot \sqrt{p} \leq |\overline{E}| \leq p + 1 + 2 \cdot \sqrt{p}$$

Diese gibt eine obere und untere Schranke für $\overline{E}$ an. Liegt innerhalb dieser Schranke eine Primzahl, so ist die Wahrscheinlichkeit dafür, dass E eine zyklische Gruppe über $\mathbb{F}_p$ definiert, sehr hoch. Ist eine elliptische Kurve E eine zyklische Gruppe und die Anzahl der Punkte ist nicht prim, dann hat sie mindestens ein Element, welches ein Erzeuger der Gruppe ist. Mit diesem Erzeugerelement lassen sich alle anderen Elemente der Gruppe durch Punktverdopplung bzw. Punktaddition generieren. Der Punkt hat damit die gleiche Ordnung wie E. Ist der Erzeuger und ein weiterer Punkt gegeben, so lassen sich auch über die Punktaddition weitere Punkte finden. Weiterhin sei gesagt, dass der Punkt im Unendlichen kein Erzeuger sein kann. Die Berechnung aller Punkte auf der Kurve ist trivial, wenn der Erzeuger bekannt ist. Nehmen wir als Beispiel eine elliptische Kurve E gegeben durch

$$y^2 = x^3 + 2x + 2 \in \mathbb{F}_{17}$$

Der Erzeuger der Gruppe ist der Punkt P = (5, 1). Um zu zeigen, dass dieser Punkt ein Erzeuger oder Generator ist, muss er mit sich selbst so oft addiert werden, bis er den Punkt im Unendlichen ergibt. Bei dem gegebenen Punkt ist dies bei $19 \cdot$ P der Fall. Gegeben sei außerdem $18 \cdot P = (5, -1)$. Da $19 \cdot P = 18 \cdot P + P = (5, 1) + (5, -1)$ $= \mathcal{O}$. Dieses kurze Beispiel macht deutlich, dass es aufwendig sein kann zu prüfen, ob ein Punkt P ein Erzeuger ist. In der Praxis werden in Kryptosystemen Punktgeneratoren mit großer Ordnung gewählt, um die Sicherheit in den verwendeten kryptografischen System zu gewährleisten. Der Erzeuger der Gruppe ist so nicht

einfach berechenbar. In den folgenden Beispielen ist jedoch ein Erzeuger gegeben, damit die Punktaddition respektive die Punkterdopplung trivial ist.

Bevor das Beispiel durchgegangen wird, werden noch einmal die benötigten Formeln und die Vorgehensweise erläutert. Es sei ein Erzeugerpunkt P gegeben mit $P = (x_1, y_2)$. Da nur dieser gegeben ist, wird er mit sich selbst addiert. Somit gilt: $2 \cdot P = P + P$. Durch die Punktverdopplung entsteht ein neuer Punkt. P kann jetzt mit $2 \cdot P$ addiert werden. Dies wird solange fortgeführt, bis der Punkt im Unendlichen herauskommt. Anschließend hat man alle Punkte auf E berechnet. Im Folgenden wird die Vorgehensweise über Pseudocode in Python erläutert, wenn ein Erzeugerpunkt gegeben ist. Es soll gesagt sein, dass es sich hierbei um eine starke Vereinfachung handelt. Beispielsweise wird nicht darauf eingegangen, dass das p des Primkörpers für die Durchführung der Addition von Punkten benötigt wird:

```
1  function punktbestimmung(P, Q):
2      Punkte = []
3      if P = UNENDLICH:
4          return FEHLER
5      zaehler = 0
6      while 1:
7          if P = Q
8              Punkte[zaehler] = P + P
9              Q = Punkte[zaehler]
10             if Q = UNENDLICH
11             zaehler = zaehler + 1
12         if P != Q
13             Punkte[zaehler] = P + Q
14             Q = Punkte[zaehler]
15             if Q = UNENDLICH
16             zaehler = zaehler + 1
17     return Punkte
```

Listing 3.1: Punktberechnung in Python

Damit die Theorie anschaulich wird, folgt nun ein Beispiel für die Punktaddition und -verdopplung, um alle Punkte auf einer elliptischen Kurve zu berechnen. Das Beispiel ist angelehnt an [6, S. 279-280]. Gegeben sei eine elliptische Kurve E mit

$$y^2 = x^3 + 2x + 2 \in \mathbb{F}_{17}$$

Weiterhin sei ein Punkt P auf der elliptischen Kurve gegeben durch $P = (5, 1)$. Da

nur ein Punkt gegeben ist, muss dieser verdoppelt werden. Es gilt $2 \cdot P = P + P = (x_1, y_1) + (x_2, y_2) = (5, 1) + (5, 1) = (x_3, y_3)$. Es gilt

$$s = \frac{3x_1^2 + a}{2y_1}, \text{ da } P = Q$$

Setzt man die Werte ein, erhält man

$$s = \frac{3x_1^2 + a}{2y_1} = (2 \cdot 1)^{-1}(3 \cdot 5^2 + 2) = 2^{-1} \cdot 9 \equiv 9 \cdot 9 \equiv 13 \bmod 17$$

Da nun s berechnet wurde, kann man nun die folgenden Formeln lösen und x_3 sowie y_3 bestimmen.

$$x_3 = s^2 - x_1 - x_2 = 13^2 - 5 - 5 = 159 \equiv 6 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 13(5 - 6) - 1 = -14 \equiv 3 \bmod 17$$

Damit ist $2 \cdot P = (6, 3)$. Da jetzt zwei (unterschiedliche) Punkte vorhanden sind, kann nun die im Vergleich zur Punktverdopplung simplere Punktaddition durchgeführt werden, um weitere Punkte zu finden. Dafür addieren wir nun alle aufeinander folgenden Punkte zusammen. $3 \cdot P = P + Q = (5, 1) + (6, 3)$, wobei $Q = 2 \cdot P$.
Für s berechnet man

$$s = \frac{y_2 - y_1}{x_2 - x_1}, \text{ da } P \neq Q$$

Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{3 - 1}{6 - 5} = 2 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 4 - 5 - 6 = -7 \equiv 10 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 2(5 - 10) - 1 = -11 \equiv 6 \bmod 17$$

Damit ist $3 \cdot P = (10, 6)$. $4 \cdot P = P + Q = (5, 1) + (10, 6)$, wobei $Q = 3 \cdot P$. Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{6 - 1}{10 - 5} = 1 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 1 - 5 - 10 = -14 \equiv 3 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 1(5 - 3) - 1 = 1 \bmod 17$$

Damit ist $4 \cdot P = (3, 1)$. $5 \cdot P = P + Q = (5, 1) + (3, 1)$, wobei $Q = 4 \cdot P$.
Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{1 - 1}{3 - 5} = 0 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 0 - 5 - 3 = -8 \equiv 9 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 0(5 - 9) - 1 = -1 \equiv 16 \bmod 17$$

Damit ist $5 \cdot P = (9, 16)$. $6 \cdot P = P + Q = (5, 1) + (9, 16)$, wobei $Q = 5 \cdot P$. Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{16 - 1}{9 - 5} = 15 \cdot 13 \equiv 8 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 13 - 5 - 9 = -1 \equiv 16 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 8(5 - 16) - 1 = -89 \equiv 13 \bmod 17$$

Damit ist $6 \cdot P = (16, 13)$. $7 \cdot P = P + Q = (5, 1) + (16, 13)$, wobei $Q = 6 \cdot P$. Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{13 - 1}{16 - 5} = 12 \cdot 14 \equiv 15 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 4 - 5 - 16 = -17 \equiv 0 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 15(5 - 0) - 1 = 6 \bmod 17$$

Damit ist $7 \cdot P = (0, 6)$. $8 \cdot P = P + Q = (5, 1) + (0, 6)$, wobei $Q = 7 \cdot P$. Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{6 - 1}{0 - 5} = -1 \equiv 16 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 1 - 5 - 0 = -4 \equiv 13 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 16(5 - 13) - 1 = -129 \equiv 7 \bmod 17$$

Damit ist $8 \cdot P = (13, 7)$. $9 \cdot P = P + Q = (5, 1) + (13, 7)$, wobei $Q = 8 \cdot P$. Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{7 - 1}{13 - 5} = 6 \cdot 15 = 5 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 25 - 5 - 13 = 7 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 5(5 - 7) - 1 = -11 \equiv 6 \bmod 17$$

Damit ist $9 \cdot P = (7, 6)$. $10 \cdot P = P + Q = (5, 1) + (7, 6)$, wobei $Q = 9 \cdot P$.

Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{6-1}{7-5} = 5 \cdot 9 \equiv 11 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 2 - 5 - 7 = -10 \equiv 7 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 11(5-7) - 1 = -23 \equiv 11 \bmod 17$$

Damit ist $10 \cdot P = (7, 11)$. $11 \cdot P = P + Q = (5, 1) + (7, 11)$, wobei $Q = 10 \cdot P$. Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{11-1}{7-5} = 5 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 25 - 5 - 7 = 13 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 5(5-13) - 1 = -41 \equiv 10 \bmod 17$$

Damit ist $11 \cdot P = (13, 10)$. $12 \cdot P = P + Q = (5, 1) + (13, 10)$, wobei $Q = 11 \cdot P$. Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{10-1}{13-5} = 9 \cdot 15 \equiv 16 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 1 - 5 - 13 = -17 \equiv 0 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 16(5-0) - 1 = 79 \equiv 11 \bmod 17$$

Damit ist $12 \cdot P = (0, 11)$. $13 \cdot P = P + Q = (5, 1) + (0, 11)$, wobei $Q = 12 \cdot P$. Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{11-1}{0-5} = -2 \equiv 15 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 4 - 5 - 0 = -1 \equiv 16 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 15(5-16) - 1 = -166 \equiv 4 \bmod 17$$

Damit ist $13 \cdot P = (16, 4)$. $14 \cdot P = P + Q = (5, 1) + (16, 4)$, wobei $Q = 13 \cdot P$. Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{4-1}{16-5} = 3 \cdot 14 \equiv 8 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 13 - 5 - 16 = -8 \equiv 9 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 8(5-9) - 1 = -33 \equiv 1 \bmod 17$$

Damit ist $14 \cdot P = (9, 1)$. $15 \cdot P = P + Q = (5, 1) + (9, 1)$, wobei $Q = 14 \cdot P$.

Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{1-1}{9-5} = 0 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 0 - 5 - 9 = -14 \equiv 3 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 0(5-3) - 1 = -1 \equiv 16 \bmod 17$$

Damit ist $15 \cdot P = (3, 16)$. $16 \cdot P = P + Q = (5, 1) + (3, 16)$, wobei $Q = 15 \cdot P$. Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{16-1}{3-5} = -15 \cdot 9 \equiv 1 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 1 - 5 - 3 = -7 \equiv 10 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 1(5-10) - 1 = -6 \equiv 11 \bmod 17$$

Damit ist $16 \cdot P = (10, 11)$. $17 \cdot P = P + Q = (5, 1) + (10, 11)$, wobei $Q = 16 \cdot P$. Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{11-1}{10-5} = 2 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 4 - 5 - 10 = -11 \equiv 6 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 2(5-6) - 1 = -3 \equiv 14 \bmod 17$$

Damit ist $17 \cdot P = (6, 14)$. $18 \cdot P = P + Q = (5, 1) + (6, 14)$, wobei $Q = 17 \cdot P$. Daraus ergibt sich

$$s = \frac{y_2 - y_1}{x_2 - x_1} = \frac{14-1}{6-5} = 13 \bmod 17$$

$$x_3 = s^2 - x_1 - x_2 = 16 - 5 - 6 = 5 \bmod 17$$

$$y_3 = s(x_1 - x_3) - y_1 = 13(5-5) - 1 = -1 \equiv 16 \bmod 17$$

Damit ist $18 \cdot P = (5, 16)$. $19 \cdot P = P + Q = (5, 1) + (5, 16) = \infty$, wobei $Q = 18 \cdot P$. Der Punkt im unendlichen wurde erreicht, womit nun alle Punkte gefunden wurden. Da die Ordnung der Untergruppe von P auf der elliptischen Kurve ein Teiler der Gruppenordnung ist, muss die Anzahl der Elemente der Kurve ein Vielfaches von 19 sein. Da wir insgesamt 19 Punkte mit dem Punkt im Unendlichen berechnet haben und $2 \cdot 19$ zu groß für die Hasse-Weil-Schranke ist da es das Intervall überschreitet, beudeutet dies, dass alle Punkte auf der elliptischen Kurve gefunden wurden.

Ähnlich wie bei der Brute-Force-Methode in 3.3.2 ist die Berechnung über die Punktaddition und -verdopplung sehr rechenaufwendig. Die Schritte können in dieser Methode nicht ohne weiteres vereinfacht werden. Dadurch bleiben nur die Möglichkei-

ten offen, diese Methode in Code abzubilden, wie in **??** oder es händisch auszurechnen, was sehr zeit- und rechenaufwendig ist. Berechnet man die Punkte mit der Hand, dann ist die Fehleranfälligkeit hoch, da die vielen Rechnungen schnell unübersichtlich werden. Dadurch bleibt die Implementierung dieser Methode über Programmcode die einzige logische und effiziente Möglichkeit. Die nächste angeführte Methode zur Berechnung von Punkten auf einer elliptischen Kurve ist die algorithmische Brute-Force-Methode. Durch sie kann im Vergleich zu den zwei vorausgehenden Methoden der Zeit- und Rechenaufwand verringert werden.

3.3.4. Punktbestimmung: Algorithmische Brute-Force-Methode

Die algorithmische Brute-Force-Methode ist der klassischen Brute-Force-Methode in 3.3.2 sehr ähnlich. Sie unterscheidet sich jedoch in einem wichtigen Aspelt. Während man bei der klassichen Methode die Quadrate und ihre Quadratwurzeln durch Ausprobieren aller Möglichkeiten herausfindet, wird bei der algorithmischen Methode die Findung der Quadrate und ihrer Wurzeln über einen Algorithmus durchgeführt. Der Algorithmus beinhaltet eine Quadratprüfung sowie eine Quadratwurzelprüfung über Schlussfolgerungen aus dem Satz von Fermat. Bei großen p ist das Finden von Quadraten und der Quadratwurzeln durch Ausprobieren sehr aufwendig. Der Vorteil der algorithmischen Methode ist, dass diese auch bei einem großen p noch effizient berechnet werden können.

Satz von Fermat Ist $\mathbb{F}_q$ ein endlicher Körper mit $q = p^e$ Elementen und p ist eine Primzahl, so ist die Einheitengruppe $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}$ zyklisch von der Ordnung $q - 1$. Aus diesem Satz lassen sich zwei Folgerungen herleiten, welche Grundlage für die algorithmische Brute-Force-Methode sind. Die genaue Herleitung der Folgerungen soll kein Teil dieser Arbeit sein. Die Folgerungen wurden dem Script von Hübl entnommen [4, S. 269].

Folgerung 1 Ist $\mathbb{F}_p$ ein endlicher Primkörper mit einer Primzahl $p \geq 3$, so ist ein $x \in \mathbb{F}_p^*$ genau dann ein Quadrat, wenn $x^{\frac{p-1}{2}} = 1$. Genau die Hälfte der von 0 verschiedenen Elemente von $\mathbb{F}_p^*$ sind Quadrate.

Folgerung 2 Ist p eine Primzahl, für die 4 ein Teiler von $p + 1$ ist und ist $x \in \mathbb{F}_p^*$ ein Quadrat, so ist

$$a = x^{\frac{p+1}{4}}$$

eine Quadratwurzel von x.

Dies wird nun an einem Beispiel verdeutlicht. Gegeben sei ein Polynom einer elliptischen Kurve E der Form

$$F(X,Y) = Y^2 - X^3 - X - 1 \in \mathbb{F}_7$$

Stellt man die Gleichung der Funktion nach y^2 um, dann erhält man folgende Gleichung:

$$y^2 = x^3 + x + 1 \bmod 7$$

Um nun zu prüfen, welche Elemente in $\mathbb{F}_7$ Quadrate sind, wird nun für jedes Element geprüft, ob $x^{\frac{p-1}{2}} = 1$ in $\mathbb{F}_7$. Die Elemente sind $\{0, 1, ..., p-1\}$. Die Elemente werden nun in die Funktion der Gleichung eingesetzt.

- $f(0) = 0^3 + 0 + 1 = 1 \in \mathbb{F}_7$
- $f(1) = 1^3 + 1 + 1 = 3 \in \mathbb{F}_7$
- $f(2) = 2^3 + 2 + 1 = 4 \in \mathbb{F}_7$
- $f(3) = 3^3 + 3 + 1 = 3 \in \mathbb{F}_7$
- $f(4) = 4^3 + 4 + 1 = 6 \in \mathbb{F}_7$
- $f(5) = 5^3 + 5 + 1 = 5 \in \mathbb{F}_7$
- $f(6) = 6^3 + 6 + 1 = 6 \in \mathbb{F}_7$

Nun muss geprüft werden, ob es sich bei den Ergebnissen um Quadrate handelt.

- $1^{\frac{7-1}{2}} = 1 \bmod(7)$
- $3^{\frac{7-1}{2}} = 6 \neq 1 \bmod(7)$
- $4^{\frac{7-1}{2}} = 1 \bmod(7)$
- $5^{\frac{7-1}{2}} = 6 \neq 1 \bmod(7)$
- $6^{\frac{7-1}{2}} = 6 \neq 1 \bmod(7)$

Da $1^{\frac{7-1}{2}} = 1 \bmod(7)$ und $4^{\frac{7-1}{2}} = 1 \bmod(7)$ handelt es sich um Quadrate. Aus diesen Quadraten ergeben sich die folgenden Punkte auf der elliptischen Kurve E:

- $f(0) = 1 \quad \Longrightarrow \quad$ Punkt 1: $(0, 1) \quad$ Punkt 2: $(0, 6)$
- $f(2) = 4 \quad \Longrightarrow \quad$ Punkt 1: $(2, 2) \quad$ Punkt 2: $(2, 5)$

Wie man sieht, ist die Anwendung relativ einfach umzusetzen. Es gibt jedoch einen weiteren Fall, welcher betrachtet werden muss, um die Wurzeln der Quadrate zu finden. Der Fall 1 wurde soeben betrachtet und anhand eines beispiels durchgerechnet. In diesem Fall ist 4 ein Teiler von $p+1$ ist. Wenn es diesen Fall gibt kann es auch sein, dass $4 \nmid (p+1)$ ist. Für diesen Fall 2 gibt es einen probabilistischen Ansatz, um die Quadratwurzeln zu finden. Dafür wird eine Induktion durchgeführt. Hübl beschreibt in seinem Script die Vorgehensweise der Induktion um die Quadratwurzeln zu finden, wenn $4 \nmid (p+1)$ [4, S. 270-272].

Dafür wählt man ein $b \in \mathbb{F}_p^*$, das kein Quadrat ist. Dafür muss $b^{\frac{p-1}{2}} \neq 1$ sein in $\mathbb{F}_p^*$. Da die Hälfte der Elemente von $\mathbb{F}_p^*$ diese Bedingung erfüllt, kann ein Element, welches kein Quadrat ist, relativ schnell gefunden werden. Notwendigerweise gilt dadurch $b^{\frac{p-1}{2}} = -1$. Da $4 \nmid (p+1)$, gilt außerdem auch, dass $4|(p-1)$. Dadurch kann man den Bruch im Exponenten umschreiben:

$$\frac{p-1}{2} = 2^l \cdot t$$

mit einem $l \geq 1$ und einem ungeraden t. Jetzt werden nach dem Ansatz von Hübl induktiv die Zahlen $n_0, n_1, ..., n_l \in \mathbb{Z}$ konstruiert mit folgenden Eigenschaften:

- n_i ist $(l-i)$-mal durch 2 teilbar.
- $x^{2^{l-i} \cdot t} \cdot b^{2n_i} = 1$.

Nun wird die Induktion durchgeführt.

Induktionsanfang $i = 0$:
Setze $n_0 = 0$. Es gilt:

1. n_0 ist beliebig oft, also sicherlich $(l-0)$-mal durch 2 teilbar
2. $x^{2^{l-i} \cdot t} \cdot b^{2n_0} = x^{2^l \cdot t} \cdot b^0 = x^{\frac{p-1}{2}} \cdot 1 = 1$.

Induktionsschluss $i \to i+1$ (*mit* $i+1 \leq l$)
Als Induktionsvoraussetzung wird angenommen, dass ein n_i gefunden wurde, welches $(l-i)$-mal durch 2 teilbar ist und $x^{2^{l-i} \cdot t} \cdot b^{2n_i} = 1$.
Setze

$$c_i = x^{2^{l-(i+1)} \cdot t} \cdot b^{n_i}$$

Dann gilt

$$c_i^2 = \left(x^{2^{l-(i+1)} \cdot t} \cdot b^{n_i}\right)^2 = x^{2 \cdot 2^{l-(i+1)} \cdot t} \cdot b^{2 \cdot n_i} = x^{2^{l-i}} \cdot b^{2n_i} = 1$$

durch die Induktionsvoraussetzung, wodurch $c_i = 1$ oder $c_i = -1$.
Wenn $c_i = 1$, dann setze $n_{i+1} = \frac{n_i}{2}$. Dies ist möglich, da n_i $(l-i)$-mal durch 2 teilbar und $i < i+1 \leq l$. Hierfür gilt

1. n_{i+1} ist $(l-(i+1)$-mal durch 2 teilbar
2. $x^{2^{l-(i+1)} \cdot t} \cdot b^{2 \cdot n_{i+1}} = x^{2^{l-(i+1)}} \cdot b^{n_i} = c_i = 1$

Falls $c_i = -1$, so setze

$$n_{i+1} = \frac{n_i}{2} + \frac{p-1}{4} = \frac{n_i}{2} + 2^{l-1} \cdot t$$

Hierfür gilt

1. n_{i+1} ist $(l-(i+1)$-mal durch 2 teilbar
2. $x^{2^{l-(i+1)} \cdot t} \cdot b^{2n_{i+1}} = x^{2^{l-(i+1)}} \cdot b^{n_i} \cdot b^{\frac{p-1}{2}} = c_i \cdot b^{\frac{p-1}{2}} = (-1) \cdot (-1) = 1$

Der Induktionsschritt wurde damit erfolgreich beendet und die Konstruktion ist dadurch abgeschlossen. Nun wird $n = n_l$ gesetzt. Nach der Konstruktion gilt hierfür dann

$$x^t \cdot b^{2n} = x \cdot x^t \cdot b^{2n} = x \cdot 1 = x$$

Da t ungerade ist, muss $t+1$ gerade sein. Aus diesem Grund existiert

$$a = x^{\frac{t+1}{2}} \cdot b^n$$

und hierfür gilt

$$a^2 = \left(x^{\frac{t+1}{2}} \cdot b^n\right)^2 = x^{t+1} \cdot b^{2n} = x$$

und damit ist a eine Quadratwurzel von x.

Hübl beschreibt hier sehr anschaulich, wie die Induktion durchzurechnen ist. Da der Fall $4 \not| (p+1)$ sehr kompliziert ist, wollen wir dies nun anhand eines Beispiels vollständig durchrechnen. Gegeben sei ein Polynom einer elliptischen Kurve E der Form

$$F(X,Y) = Y^2 - X^3 - 12 \cdot X - 17 \in \mathbb{F}_{421}$$

Stellt man die Gleichung der Funktion nach y^2 um, dann erhält man folgende Gleichung:

$$y^2 = x^3 + 12 \cdot x + 17 \bmod 421$$

Es soll geprüft werden, ob $x_0 = 200$ und $x_1 = 400$ auf der elliptischen Kurve E Quadrate sind. Weiterhin sollen die Wurzeln und die Punkte auf der elliptischen Kurve bestimmt werden, sofern es sich bei x_0 und x_1 um Quadrate handelt.

Als erstes wird geprüft, ob $x_0 = 200$ ein Quadrat ist.

$$f(200) = 200^3 + 12 \cdot 200 + 17 = 49 \in \mathbb{F}_{421}$$

Nun prüfen wir, ob $49^{\frac{p-1}{2}} = 1 \bmod 421$.

$$49^{\frac{421-1}{2}} = 49^2 10 = 1 \bmod 421$$

Da $49^{\frac{p-1}{2}} = 1 \bmod 421$, handelt es sich bei x_0 um ein Quadrat. Nun müssen die Wurzeln bestimmt werden. 4 ist kein Teiler von $p + 1$. Wir setzen

$$\frac{p-1}{2} = 210 \implies 2^l \cdot t = 2^1 \cdot 105$$

Nun wird ein $b \in \mathbb{F}_{421}^*$ gewählt, das kein Quadrat ist. Dafür gehen wir alle $\mathbb{P}$ durch, die kleiner als 421 sind:

$$2^2 10 \equiv 420 \equiv -1 \bmod 421$$

Die Primzahl 2 ist kein Quadrat und wird für b gewählt.

$$c_0 = x^{2^{l-(0+1)} \cdot t} \cdot b^{n_0} \implies 49^{2^0 \cdot 105} \cdot 2^0 = 1 \bmod 421$$

Da $c_0 = 1$, setze

$$n_1 = \frac{n_0}{2} \implies \frac{0}{2} = 0$$

Wir setzen

$$a = x^{\frac{t+1}{2}} \cdot b^n \implies a = 49^{\frac{105+1}{2}} \cdot 2^0 = 414$$

Die Zahl 414 ist eine Quadratwurzel von x_0. Um beide Quadratwurzeln zu bekommen negieren wir 414:

$$-414 \equiv 7 \bmod 421$$

Das Element x_0 ist ein Quadrat und hat die Wurzeln 414 und 7. Daraus ergeben sich die folgenden Punkte:

$$P_1 = (200, 414) \quad \text{und} \quad P_2 = (200, 7)$$

Als nächstes prüfen wir, ob $x_1 = 400$ ein Quadrat ist.

$$f(400) = 400^3 + 12 \cdot 400 + 17 = 187 \in \mathbb{F}_{421}$$

Nun prüfen wir, ob $187^{\frac{p-1}{2}} = 1 \bmod 421$.

$$187^{\frac{421-1}{2}} = 187^2 10 = 1 \bmod 421$$

Da $187^{\frac{p-1}{2}} = 1 \bmod 421$, handelt es sich bei x_1 um ein Quadrat. Nun müssen die Wurzeln bestimmt werden. 4 ist kein Teiler von $p+1$. Wir setzen

$$\frac{p-1}{2} = 210 \implies 2^l \cdot t = 2^1 \cdot 105$$

Nun wird ein $b \in \mathbb{F}_{421}^*$ gewählt, das kein Quadrat ist. Dafür gehen wir alle $\mathbb{P}$ durch, die kleiner als 421 sind:

$$2^2 10 \equiv 420 \equiv -1 \bmod 421$$

Die Primzahl 2 ist kein Quadrat und wird für b gewählt.

$$c_0 = x^{2^{l-(0+1)} \cdot t} \cdot b^{n_0} \implies 187^{2^0 \cdot 105} \cdot 2^0 = 420 = -1 \bmod 421$$

Da $c_0 = -1$, setze

$$n_1 = \frac{n_0}{2} + \frac{p-1}{4} \implies \frac{0}{2} + \frac{420}{4} = 105$$

Wir setzen

$$a = x^{\frac{t+1}{2}} \cdot b^n \implies a = 187^{\frac{105+1}{2}} \cdot 2^1 05 = 271$$

Die Zahl 271 ist eine Quadratwurzel von x_1. Um beide Quadratwurzeln zu bekommen negieren wir 271:

$$-271 \equiv 150 \bmod 421$$

Das Element x_1 ist ein Quadrat und hat die Wurzeln 271 und 150. Daraus ergeben sich die folgenden Punkte:

$$P_3 = (400, 271) \quad \text{und} \quad P_4 = (400, 150)$$

Das Beispiel ist zwar kompliziert, doch wenn man alle Schritte der Induktion durchgeht, dann kann man das Ergebnis nachvollziehen. Beim algorithmischen Brute-Force muss also immer geprüft werden, ob $4|(p+1)$ oder ob $4 \nmid (p+1)$. Hübl beschreibt in seinem Script die allgemeine Vorgehensweise für den algorithmischen Brute-Force [4,

S. 271-272]. Gegeben sei ein $x \in \mathbb{F}_p^*$.

1. Berechne $x^{\frac{p-1}{2}}$ in $\mathbb{F}_p$. Falls $x^{\frac{p-1}{2}} \neq 1$: **STOPP**, x ist kein Quadrat in $\mathbb{F}_p$.

2. Falls $4|(p+1)$, so setze
$$a = x^{\frac{p+1}{4}}$$
Falls $4 \not| (p+1)$, wähle ein $b \in \mathbb{F}_p$ mit $b^{\frac{p-1}{2}} = -1$, schreibe $\frac{p-1}{2} = 2^l \cdot t$ (mit t ungerade) wie oben und konstruiere
$$a = x^{\frac{t+1}{2}} \cdot b^n$$
wie oben.

3. Die Zahl a ist eine Quadratwurzel von x in $\mathbb{F}_p$.

3.4. Diskreter Logarithmusproblem über elliptischen Kurven

Wie in Kapitel 2.3.1 gezeigt, kann das DLP verallgemeinert werden, sodass über jeder zyklischen Gruppe (mehr oder weniger effektiv) ein DLP definiert erden kann. Im ersten Abschnitts dieses Kapitels wurden die Gruppeneigenschaften der Punktmenge auf der elliptischen Kurve gezeigt. Gruppenelemente können über die Gruppenoperation + *addiert* werden.

Um eine kryptografisch *sicheres* DLP zu konstruieren, ist es wichtig die Gruppenkardinalität zu kennen. Deren exakte Bestimmung ist bei elliptischen Kurven jedoch sehr aufwendig. Für die grobe Bestimmung der Gruppenkardinalität wurde in Kapitel 3.3.1 bereits die Hasse-Weil-Schranke vorgestellt. Diese erlaubt es uns die Anzahl der Punkte einer elliptischen Kurve abzuschätzen. Vereinfacht kann man sagen das die Anzahl der Punkte in etwa p entspricht, wenn die elliptische Kurve über $\mathbb{Z}_p^*$ definiert ist. Es gibt außerdem ein Verfahren von R. Schoof, das relativ effiziernt die Ordnung einer elliptischen Kurve bestimmt. Hat diese Ordnung einen ausreichend großen Primteiler, so ist sie für die Kryptografie geeignet. Der Algorithmus kann unter [10, S. 219-254] nachgeschlagen werden.

Im folgenden wollen wir das DLP über elliptischen Kurven definieren:

Definition: Das DLP über elliptischen Kurven (ECDLP) Gegeben sei eine elliptische Kurve E. Wir betrachten eine zyklische Untergruppe U von E mit hinreichend großer Primzahlordnung, ein primitives Element P von U und ein beliebiges weiteres Element T. Das DLP ist Entscheidung ob $T \in U$ ist und ggf. die Bestimmung der natürlichen Zahl d zwischen $1 \leq d \leq \#E$, sodass gilt:

$$\underbrace{P + P + \cdots + P}_{d-mal} = dP = T$$

Die Bestimmung der natürlichen Zahl d ist wie auch schon beim DLP über $\mathbb{Z}_p^*$ nicht trivial. In einem Kryptosystem wäre dementsprechend die natürliche Zahl d der private Schlüssel und das Element T der öffentliche Schlüssel. Anders als beim DLP über Primkörpern, wo beide Schlüssel natürliche Zahlen waren, ist der öffentliche Schlüssel T hier ein Punkt der elliptischen Kurve. Der in der obigen Definition verwendete Ausdruck dP wird *Punktmultiplikation* genannt, da man schreiben kann $T = dP$. Es ist jedoch zu beachten das es dafür keine unmittelbare Rechenvorschrift gibt, die Zahl d mit P zu multiplizieren. Der Ausdruck bedeutet lediglich, dass die Gruppenoperation $d - 1$ mal auf P angewendet wird.

Der Erzeuger eines Schlüsselpaars muss allerdings nicht einzeln $d - 1$ mal P auf sich selbst addieren, sondern kann sich eines *Tricks* bedienen. Wie beim DLP über dem Primkörper $\mathbb{Z}_p^*$, wo der Square-And-Multiply-Algorithmus zur effizienten Berechnung des öffentlichen Schlüssels verwendet wurde, kann hier der sog. *Double-And-Add-Algorithmus* verwendet werden. Dieser ist von der Idee her zum Square-And-Multiply-Algorithmus ähnlich, und unterscheidet sich hauptsächlich durch die angewandten Operationen. Statt einer Potenzierung durch iteriertes Quadrieren, wird bei diesem Algorithmus eine Vielfachenbildung durch iteriertes Verdoppeln erreicht. Der folgende Code zeigt eine Beispielhafte Implementierung:

```
1 def doubble_and_add(kPriv, start_point):
2     binary = bin(kPriv)
3     binary = binary[3:]
4     cur_point = start_point
5     for digit in binary:
6         # Doubble
7         cur_point = curve.add(cur_point, cur_point)
8         if digit == "1":
```

```
9               # Add
10              cur_point = curve.add(cur_point, start_point)
11      return cur_point
```

Listing 3.2: Double-And-Add-Algorithmus in Python

Zuerst wird die Binärdarstellung des private Schlüssels d in *binary* gespeichert. Nun wird diese Binärdarstellung von links nach rechts durchlaufen, wobei die vorderste 1 ignoriert wird, da diese quasi schon in $1\cdot$ Startpunkt P steckt. Für jede Ziffer wird zunächst der aktuelle Punkt verdoppelt und im Falle einer 1 wird zudem noch der Startpunkt addiert. Dieses Vorgehen wird angewandt bis die komplette Binärdarstellung von d abgearbeitet ist. Der Output entspricht dP.
Nachfolgend wollen wir verstehen, warum dieser Algorithmus funktioniert. Dazu betrachten wir das folgende Beispiel:
Das Ziel ist es den Punkt $19P$ zu berechnen:

$$21P = (10101_2)P = (d_4d_3d_2d_1d_0)_2P$$

Die Bits d_4 bis d_0 werden in absteigender Reihenfolge verarbeitet:

$d_4 = 1:$	$P = \mathbf{1}_2\ P$	Startwert 1
$d_3 = 0:$	$P + P = 2\ P = \mathbf{10}_2\ P$	double
$d_2 = 1:$	$2\ P + 2\ P = 4P = \mathbf{100}_2\ P$	double
	$4\ P + P = 5\ P = \mathbf{101}_2\ P$	add, da $d_2 = 1$
$d_1 = 1:$	$5\ P + 5\ P = 10\ P = \mathbf{1010}_2\ P$	double
$d_0 = 1$	$10\ P + 10\ P = 20\ P = \mathbf{10100}_2\ P$	double
	$20\ P + P = 21\ P = \mathbf{10101}_2\ P$	add, da $d_0 = 1$

Das darstellen von d als binäre Zahl ermöglicht es, diese ausgehend von der ersten 1 ausgehend *zusammenzubauen*. Jede Stelle im Binärsystem ist doppelt so wertig wie die vorherige, weshalb durch Verdoppeln die ganze Zahl um eine Stelle nach links geschoben wird. Durch diese Operation wird dann sozusagen eine 0 *erzeugt*. Wenn nun eine 1 benötigt wird kann diese durch die Addition von 1 bzw. dem Startpunkt *erzeugt* werden.

Mittels dieses Algorithmus von die Komplexität der Berechnung des Punktes $T = dP$

von $O(d)$ auf $O(1,5 \cdot \log_{10} d)$ reduziert werden. Wodurch die Berechnung von T auch für große d verhältnismäßig schnell durchzuführen ist. Der Angreifer hat keine Möglichkeit die Berechnung von d derart zu beschleunigen, was das DLP auch hier zu einer Einwegfunktion macht [6, 280-284].

Das DLP über elliptischen Kurven bilden die Grundelage für viele moderne kryptografischen Protokolle, wie den Signaturalgorithmus ECDSA, das Verschlüsselungsystem EC-ElGamal oder das Schlüsselaustauschverfahren ECDHKE. Letzteres wird im folgenden Kapitel genauer erläutert.

3.5. Elliptic-Curve-Diffie-Hellman Key Exchange (ECDHKE)

Der ECDHKE ist eine Anpassung des klassischen DHKE, wobei hier als zyklische Gruppe nicht $\mathbb{Z}_p^*$ sondern die Punktemenge einer elliptischen Kurve verknüpft durch die in Kapitel 3.1 definierte Gruppenoperation. Wie in Kaptiel 3.4 gezeigt, kann auch in einer solchen zyklischen Gruppe bzw. Untergruppe ein DLP definiert werden, welches sogar deutlich schwieriger zu lösen ist, als das DLP über $\mathbb{Z}_p^*$.

Die Konstruktion des ECDHKE funktioniert analog zu jener des DHKE, welche in Kapitel 2.3.2 erläutert wurde. zunächst müssen wieder die Domain-Parameter festgelegt werden:

ECDHKE-Domain-Parameter

1. Wähle eine Primzahl p und eine elliptische Kurve

$$E : y^2 \equiv x^3 + a \cdot x + b \bmod p$$

2. Wähle ein primitives Element $P = (x_P, y_P)$

Die Primzahl p, die Kurvenkoeffizienten a, b und der Punkt P bilden die sog. Domain-Parameter.

Die Wahl dieser Parameter hat großen Einfluss auf die Anzahl der Gruppenelemente und damit auf die Sicherheit des Protokolls. Die Bestimmung geeigneter Parameter ist relativ rechenaufwändig, weshalb dies in der Praxis nicht jedes mal neu gemacht

wird, sondern z.B. von der NIST vorgegebene Parameter verwendet werden. Der Schlüsselaustausch erfolgt nun analog zum DHKE über endlichen Körpern:

Diffie-Hellman-Schlüsselaustausch mit elliptischen Kurven (ECDHKE)

Alice		**Bob**
Wähle $a = k_{pr,A} \in \{2, ..., \#E - 1\}$		Wähle $b = k_{pr,B} \in \{2, ..., \#E - 1\}$
Berechne $k_{pub,A} = aP = A = (x_A, y_A)$		Berechne $k_{pub,B} = bP = B = (x_B, y_B)$
	$\xleftarrow{k_{pub,A}=A}$	
	$\xrightarrow{k_{pub,B}=B}$	
Berechne $aB = T_{AB}$		Berechne $bA = T_{AB}$

Gemeinsames Geheimnis zwischen Alice und Bob: $T_{AB} = (x_{AB}, y_{AB})$

Das Alice und Bob den gleichen Punkt berechnet haben lässt sich einfach zeigen.

Alice berechnet:

$$aB = a(bP)$$

Bob berechnet:

$$bA = b(aP)$$

Da die Punktaddition assoziativ ist, berechnen Alice und Bob den gleichen Punkt $T_{AB} = abP$.

Alice und Bob wählen zunächst jeweils eine große zufällige Primzahl a und b, welche als private Schlüssel dienen. Beide multiplizieren ihren privaten Schlüssel mit dem durch die Domain-Parameter gegebenen Generatorpunkt P um die öffentlichen Schlüssel A und B zu erhalten. Diese werden gegenseitig ausgetauscht. Im folgenden multiplizieren beide ihren privaten Schlüssel (a bzw. b) mit dem empfangenen öffentlichen Schlüssel (B bzw. A) des anderen um T_{AB} zu erhalten. Aus diesem gemeinsamen Geheimnis kann nun ein Schlüssel für eine symmetrische Verschlüsselung abgeleitet werden. In der Regel wird dazu der Wert der x-Koordinate verwendet. Folgend wird der ECDHKE beispielhaft mit kleinen Zahlen durchgeführt.

Alice		**Bob**
Wähle $a = k_{pr,A} = 3$		Wähle $b = k_{pr,B} = 10$
$k_{pub,A} = A = 3P = (10, 6)$		$k_{pub,B} = B = 10P = (7, 11)$
	$\xleftarrow{k_{pub,A}=A}$	

$$\xrightarrow{k_{pub,B}=B}$$

$T_{AB} = aB = 3(7, 11) = (13, 10)$ $\qquad$ $T_{AB} = bA = 10(10, 6) = (13, 10)$

Gemeinsames Geheimnis zwischen Alice und Bob: $T_{AB} = (13, 10)$

Für die Punktmultiplikationen wird der in Kapitel 3.4 erläuterte Double-and-Add-Algorithmus verwendet.

4. Implementierung in Python

In diesem Kapitel wird die Umsetzung der theoretische Behandelten Konzepte in Python thematisiert. Im Zuge dessen, werden auch die verwendeten Bibliotheken und die Programmiersprache Python kurz thematisiert.

Python ist eine höhere Programmiersprache, die in den späten 1980er Jahren entwickelt wurde. Sie wurde entworfen, um die Programmierung einfach, lesbar und zugänglich zu machen. Python ist bekannt für seine klare Syntax und seine Fähigkeit, komplexe Aufgaben mit wenigen Codezeilen zu erledigen. Die Programmiersprache ist zudem sehr flexibel und kann in vielen Bereichen eingesetzt werden, z.B. Webentwicklung, Datenanalyse, wissenschaftliche Programmierung, maschinelles Lernen, Automatisierung, Spieleentwicklung und vieles mehr. Python hat eine große Gemeinschaft von Entwicklern und bietet eine breite Palette von Bibliotheken und Frameworks, die für verschiedene Zwecke verwendet werden können[11]. Aufgrund der genannten Eigenschaften wurde für die Umsetzung dieses Projekts Python eingesetzt.

Die Bibliothek Numpy wurde verwendet um innerhalb des Programms Berechnungen durchzuführen, welche nicht durch die Standard-Pythonbibliothek bereitgestellt werden. Numpy ist eine Python-Bibliothek, die für numerische Berechnungen und Datenanalyse verwendet wird. Numpy bietet leistungsstarke Funktionen für Arrays und Matrizen sowie für mathematische Funktionen wie lineare Algebra, Fourier-Transformationen und Statistik. Numpy wurde für die Arbeit mit großen Datensätzen und für die schnelle Berechnung von mathematischen Operationen entwickelt. Numpy verwendet C-Code im Hintergrund, um eine höhere Leistung zu erreichen, was sie zu einer beliebten Wahl für wissenschaftliche Berechnungen macht. Numpy wird oft mit anderen Bibliotheken wie Pandas, Scipy und Matplotlib kombiniert, um umfassende Datenaufbereitungs- und Datenanalyse-Tools bereitzustellen [12]. In dieser Arbeit wurde nur ein Bruchteil der durch Numpy gebotenen Funktionalitäten genutzt.

Zur grafischen Darstellung der elliptischen Kurven wurde die Bibliothek Matbplotlib verwendet. Dies ist eine Python-Bibliothek für die Erstellung von Plots und Diagrammen. Matplotlib ist sehr flexibel und kann für eine Vielzahl von Anwendungen eingesetzt werden, z.B. für die Erstellung von wissenschaftlichen Diagrammen, Finanzdiagrammen, 2D- und 3D-Diagrammen und animierten Diagrammen. Matplotlib ist sehr einfach zu verwenden und bietet eine Vielzahl von Optionen für

die Gestaltung von Diagrammen, z.B. Farben, Linienstilen und Textformatierung. Matplotlib wird oft zusammen mit Numpy und anderen Bibliotheken verwendet, um Datenvisualisierung und explorative Datenanalyse zu ermöglichen [13].
Das entwickelte Programm ermöglicht es ermöglicht es elliptische Kurven zu definieren und grafisch darzustellen. Darüber hinaus kann auf Basis einer elliptischen Kurve eine zyklische Gruppe erzeugt werden, welche wiederum die Basis für einen Diffie-Hellman-Key-Exchange darstellt. In der *main.py* werden einige Beispiele für elliptische Kurven mit verschiedener Punkteanzahl gegeben. Zudem werden die implementierten Klassen beispielhaft genutzt um einen DHKE durchzuführen.

4.1. Klasse: ellipticCurveInFp

Die Klasse *ellipticCurveInFp* ermöglicht es eine elliptische Kurve der Form

$$y^2 = x^3 + ax + b \in \mathbb{F}_p$$

zu definieren. Im Zuge dessen werden verschiedene Methoden bereitgestellt.

4.1.1. Methode: get_all_points_on_curve()

Mittels des in Kapitel 3.3.2 vorgestellte Verfahrens ermittelt diese Funktion alle auf der elliptischen Kurve befindlichen Punkte.

```
1  def get_all_points_on_curve(self):
2      if not self.is_elliptic_curve_correct():
3          return False
4      # Berechnen der Quadrate und zugehoerige Wurzeln in F_p
5      squares_with_roots = supportAlgos.get_squares_with_roots(
       range(self.p))
6      # Liste mit Quadraten erstellen
7      squares = []
8      for tupel in squares_with_roots:
9          if tupel[0] not in squares:
10             squares.append(tupel[0])
11     # Pruefe fuer jedes x in F_p, ob es eingesetzt ein
       Quadrat ergibt
12     x_values = []
13     for x in range(self.p):
```

```
14          if (x ** 3 + self.a * x + self.b) % self.p in squares
    :
15              x_values.append(x)
16      # Ermitteln der Punkte auf der Kruve
17      points = []
18      for x in x_values:
19          y_quad = (x ** 3 + self.a * x + self.b) % self.p
20          for tupel in squares_with_roots:
21              if y_quad == tupel[0]:
22                  points.append((x, tupel[1]))
23      # Neutrales Element hinzufuegen
24      points.append(("N", "N"))
25      return points
```

Listing 4.1: Methode: get_all_points_on_curve()

Zunächst wird geprüft, ob die elliptische Kurve korrekt definiert ist. Durch die Funktion *get_squares_with_roots()* wird zu jeder Zahl von 0 bis $p-1$ das Quadrat in $\mathbb{F}_p$ berechnet. Die errechneten Quadrate werden in einer Liste gespeichert. Anschließend wird jedes $x > p$ in die Gleichung der elliptischen Kurve eingesetzt um zu prüfen ob das Ergebnis einem der errechneten Quadrate entspricht. Jene x-Werte auf die das zutrifft, werden in einer Liste gespeichert. Anschließend werden für jeden dieser x-Werte die zugehörigen y-Werte ermittelt und die Kombination als Punkte in einer Liste gespeichert. Im letzten Schritt wird der Liste der unendlich ferne Punkt hinzugefügt und die List zurückgegeben.

4.1.2. Methode: is_elliptic_curve_correct()

Diese Methode prüft alle Eigenschaften, die zur Definition einer Elliptischen Kurve erforderlich sind und gibt entsprechend *True* oder *False* zurück.

```
1  def is_elliptic_curve_correct(self):
2      # p muss eine Primzahl sein
3      if not is_prime(self.p):
4          return False
5      # p muss groesser als 3 sein
6      if self.p <= 3:
7          return False
```

```
8       # 4a^3 + 27b^2 darf nicht durch p teilbar sein bzw. nicht
        Null sein in F_p
9       if (4 * (self.a ** 3) + 27 * (self.b ** 2)) % self.p ==
        0:
10          return False
11      # Die Kurve ist korrekt
12      return True
```

Listing 4.2: Methode: is_elliptic_curve_correct()

Im ersten Schritt wird überprüft ob es sich beim Parameter p um eine Primzahl handelt, da $\mathbb{F}_p$ ein Primkörper sein muss. Im zweiten Schritt wird geprüft ob $p > 3$, da der Arithmetik auf Kurven mit $p <= 3$ andere Formeln zur Punktaddition zu Grunde liegen, mit welchen hier nicht Umgegangen werden kann. Als letzten Schritt wird die Bedingung

$$4x^3 + 27x^2 \neq 0 \in \mathbb{F}_p$$

geprüft. Diese muss, wie in Kapitel 3 erwähnt, gelten sein um Singularitäten auszuschließen.

4.1.3. Methode: is_point_on_curve()

Die Methode überprüft ob ein gegebener Punkt auf der definierten elliptischen Kurve liegt.

```
1  def is_point_on_curve(self, P):
2      x, y = P
3      # y^2 = x^3 + ax + b muss erfuellt sein in F_p
4      return (y ** 2 - x ** 3 - self.a * x - self.b) % self.p
       == 0
```

Listing 4.3: Methode: is_point_on_curve()

Dazu wird geprüft ob nach Einsetzten des gegebenen Punktes die Kurvengleichung

$$y^2 = x^3 + ax + b \in \mathbb{F}_p$$

erfüllt ist. Entsprechend wird *True* oder *False* zurückgegeben.

4.1.4. Methode: add()

Die *add()* Methode dient dazu zwei Punkte auf der elliptischen Kurve zu addieren. Zur Abbildung des unendlich fernen Punkts bzw. des neutralen Elements wurde die Notation (N, N) gewählt. Aufgrund der Ähnlichkeit zu $(0, 0)$, wurde nicht von (O, O) Gebrauch gemacht, was wegen der Bezeichnung des neutralen Elements $\mathcal{O}$ naheliegend wäre.

```
1  def add(self, P, Q):
2      x1, y1 = P
3      x2, y2 = Q
4      # Addition des neutralen Elements mit sich selbst ergibt
       das neutrale Element
5      if x1 == "N" and x2 == "N":
6          R = ("N", "N")
7          return R
8      # Addition eines Punktes und des neutralen Elements
       ergibt den Punkt
9      elif x1 == "N":
10         R = (x2, y2)
11         return R
12     elif x2 == "N":
13         R = (x1, y1)
14         return R
15     # Addition inverser Punkte ergibt neutrales Element
16     elif x1 == x2 and y1 != y2:
17         R = ("N", "N")
18         return R
19
20         # Addition nach bekannten Formeln
21         if x1 == x2 and y1 == y2:
22             # Punkt ist Nullstelle --> Tangente ist parallel
               zur y-Achse --> ergibt neutrales Element
23             if y1 == 0:
24                 R = ("N", "N")
25                 return R
26             # Punktaddition mit sich selbst
27             s = (3 * x1 ** 2 + self.a) * supportAlgos.
               inverse_mod(2 * y1, self.p) % self.p
28         else:
```

```
29              # Punktaddition von unterschiedlichen Punkten
30              s = (y2 - y1) * supportAlgos.inverse_mod(x2 - x1,
    self.p) % self.p
31          x3 = (s ** 2 - x1 - x2) % self.p
32          y3 = (s * (x1 - x3) - y1) % self.p
33          R = (x3, y3)
34          return R
```

Listing 4.4: Methode: add()

Zunächst werden alle Sonderfälle behandelt:

- $\mathcal{O} + \mathcal{O} = \mathcal{O}$
- $P + \mathcal{O} = \mathcal{O} + P = P$
- $P + -(P) = \mathcal{O}$
- Falls $P = (x_P, 0)$, dann gilt $P + P = \mathcal{O}$

Nach Behandlung dieser Sonderfälle erfolgt die Berechnung von $R = (x_3, y_3)$ gemäß den in Kapitel 3.1 hergeleiteten Formeln:

$$x_3 = s^2 - x_1 - x_2$$

$$y_3 = s(x_1 - x_3) - y_1$$

, wobei

$$s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{, falls } P \neq Q \text{ (Punktaddition)} \\ \frac{3x_1^2 + a}{2y_1} & \text{, falls } P = Q \text{ (Punktverdopplung)} \end{cases}$$

4.2. Klasse: CyclicGroup

Diese Klasse bietet Funktionalitäten bezüglich der Gruppeneigenschaften elliptischer Kurven. Ein Objekt der Klasse *EllipticCurveInFp* muss im Konstruktor übergeben werden. *CyclicGroup* stellt Methoden zur Addition und Skalar-Multiplikation, als auch jene zur Bestimmung der Gruppen- und Elementordnung etc. zur Verfügung.

4.2.1. Methode: add_elements()

Diese Methode stellt die Funktionalität bereit, zwei Gruppenelemente zu addieren und das Ergebnis zurückzugeben. Dazu wird intern die *add()* Methode der *EllipticCurveInFp*-Klasse genutzt.

```
1 def add_elements(self, A, B):
2     C = self.elliptic_curve.add(A, B)
3     return C
```

Listing 4.5: Methode: add_elements()

4.2.2. Methode: scalar_dot_element

Die Methode *scalar_dot_element()* ermöglicht es, einen Punkt auf der Kurve mit einem Skalar zu multiplizieren. Dabei wird der in Kapitel 3.4 erläuterte Doubble-And-Add-Algorithmus verwendet.

```
1  def scalar_dot_element(self, scalar, element):
2      binary = bin(scalar)
3      binary = binary[3:]
4      # double and add algorithm
5      current_element = element
6      for digit in binary:
7          # double
8          current_element = self.add_elements(current_element,
       current_element)
9          if digit == "1":
10             # add
11             current_element = self.add_elements(
       current_element, element)
12     return current_element
```

Listing 4.6: Methode: scalar_dot_element()

Nach Durchlauf des Algorithmus wird der zuletzt berechnete Punkt zurückgegeben.

4.2.3. Methode: get_sub_group_elements()

Unter Anwendung des Wissens aus dem Kapitel zu Untergruppen zyklischer Gruppen 2.2.4 findet diese Methode alle Element der durch einen Punkt bzw. Gruppenelement generierten Untergruppe. Dazu wird der Punkt so lange zu sich selbst addiert, bis das neutrale Element erreicht wird. Alle dabei berechneten Punkte gehören zur durch den gegebenen Punkt erzeugten Untergruppe.

```
1 def get_sub_group_elements(self, primitive_element):
2     current_element = primitive_element
3     sub_group_elements = []
4     while True:
5         sub_group_elements.append(current_element)
6         current_element = self.add_elements(current_element,
    primitive_element)
7         if current_element == primitive_element:
8             break
9     return sub_group_elements
```

Listing 4.7: Methode: get_sub_group_elements()

In jedem Schleifendurchlauf wird der errechnete Punkt einer List hinzugefügt, welche nach Erreichen des neutralen Elements zurückgegeben wird.

4.2.4. Methode: get_element_order()

Diese Methode gibt die Ordnung eines gegebenen Gruppenelements wieder. Dazu wird mittels der Methode *get_sub_group_elements()* die durch das Element erzeugte Untergruppe ermittelt. Die Anzahl der ermittelten Elemente entspricht der Ordnung des Elements.

```
1 def get_element_order(self, element):
2     sub_group_elements = self.get_sub_group_elements(element)
3     order = len(sub_group_elements)
4     return order
```

Listing 4.8: Methode: get_element_order()

4.2.5. Methode: get_group_order()

Mittels dieser Methode kann die Gruppenordnung bestimmt werden. Diese entspricht der Anzahl der Punkte auf der elliptischen Kurve, weshalb intern auf die Methode *get_all_points_on_curve()* der Klasse *EllipticCurveInFp* zurückgegriffen wird. Jene liefert alle Punkte auf der Kurve, deren Anzahl anschließend ermittelt und zurückgegeben wird.

```
1 def get_group_order(self):
2     group_elements = self.elliptic_curve.
    get_all_points_on_curve()
3     group_order = len(group_elements)
4     return group_order
```

Listing 4.9: Methode: get_group_order()

4.2.6. Methode: get_group_elements()

Durch diese Methode werden alle Gruppenelemente zurückgegeben. Diese entsprechen den Punkten auf der elliptischen Kurve, weshalb wieder die *get_all_points_on_curve()*-Methode der Klasse *EllipticCurveInFp* genutzt wird.

```
1 def get_group_elements(self):
2     return self.elliptic_curve.get_all_points_on_curve()
```

Listing 4.10: Methode: get_group_elements()

4.2.7. Methode: get_all_sub_groups()

Diese Methode gibt alle Untergruppen der zyklischen Gruppe in einer Liste wieder. Dazu wird der Satz *Konstruktion einer zyklischen Untergruppe* aus Kapitel 2.2.4 angewandt. Dieser besagt im Kern, dass nur die Gruppenkardinalität sowie ein primitives Element benötigt wird, um alle Untergruppen zu ermitteln.

```
1 def get_all_sub_groups(self):
2     sub_groups = []
3     sub_group_generators = []
```

```
4       divisors = []
5       order = self.get_group_order()
6       primitive_elements = self.get_primitive_elements()
7       # Finden von echten Teilern der Gruppenordnung
8       for number in range(int(order/2 + 1)):
9           if order % number == 0:
10              divisors.append(number)
11      # Berechnen der Untergruppen-Generatoren
12      for div in divisors:
13          sub_group_generators.append(self.scalar_dot_element((
        order/div), primitive_elements[0]))
14      # Berechnen der Untergruppen
15      for gen in sub_group_generators:
16          sub_groups.append(self.get_sub_group_elements(gen))
17      return sub_groups
```

Listing 4.11: Methode: get_all_sub_groups()

Zunächst werden unter Verwendung der Methoden *get_group_order()* und *get_primitive_elements()* die Gruppenordnung und die primitiven Elemente ermittelt. Im Grunde wird aber nur ein primitives Element benötigt. Im folgenden Schritt werden die echten Teiler der Gruppenordnung identifiziert und gespeichert. Um die Generatoren der Untergruppen zu finden, wird anschließend für jeden echten Teiler das primitive Element mit $\frac{Gruppenordnung}{echterTeiler}$ multipliziert. Nach Erhalt der Generatoren, wir mittels der Methode *get_sub_group_elements* für jeden Generator die erzeugte Untergruppe berechnet und diese in einer List gespeichert, welche schlussendlich zurückgegeben wird.

4.2.8. Methode: get_primitive_elements()

Primitive Elemente zeichnen sich dadurch aus, dass ihre Ordnung der Gruppenordnung entspricht, sie also bei Addition zu sich selbst die gesamte Gruppe generieren. Die Methode *get_primitive_elements()* prüft für jedes Element der Gruppe dessen Ordnung und gibt eine Liste jener Elemente zurück deren Ordnung der Gruppenordnung entspricht.

```
1   def get_primitive_elements(self):
2       elements = self.get_group_elements()
3       group_order = self.get_group_order()
4       primitive_elements = []
```

```
5       for element in elements:
6           if self.get_element_order(element) == group_order:
7               primitive_elements.append(element)
8       return primitive_elements
```

Listing 4.12: Methode: get_primitive_elements()

4.3. Klasse: DHKE

Die Klasse *DHKE* nutz die Methoden der Klasse *CyclicGroup* um die Funktionalität für einen Elliptic-Curve-Diffi-Hellman-Key-Exchange durchzuführen. Die Implementierung könnte genauso gut für einen klassischen DHKE genutzt werden, was im Grunde von der Übergebenen zyklischen Gruppe abhängt, daher der Name DHKE anstatt ECDHKE. Wie schon a ngedeutet wird der Klasse im Konstruktor die zugrundeliegende zyklische Gruppe übergeben.

4.3.1. Methode: gen_key_pair()

Basierend auf einem gegebenen Generator erzeugt die Methode ein Schlüsselpaar bestehend aus einem zufälligen privaten Schlüssel und einem daraus berechneten öffentlichen Schlüssel.

```
1  def gen_key_pair(self, start_element):
2      group_order = self.cyclic_group.get_element_order(
       start_element)
3      # k_priv = random.randrange(int(np.sqrt
       (10000000000000000000)), 10000000000000000000)
4      k_priv = np.random.randint(np.sqrt(group_order),
       group_order)
5      # kPub = kPriv * start_point
6      k_pub = self.cyclic_group.scalar_dot_element(k_priv,
       start_element)
7      self.k_priv = k_priv
8      self.k_pub = k_pub
9      return k_priv, k_pub
```

Listing 4.13: Methode: get_primitive_elements()

Im ersten Schritt wird mittels der Methode *get_element_order()* der Klasse *CyclicGroup* die Ordnung des Generators g ermittelt. Anschließend wird ein zufälliger privater Schlüssel $k_{priv} \in \{\sqrt{ord(g)}, ..., ord(g)\}$ erzeugt. Dieser private Schlüssel wird nun unter Verwendung der Methode *scalar_dot_element()* aus der Klasse *CyclicGroup* mit dem gegebenen Generator multipliziert um den öffentlichen Schlüssel k_{pub} zu erzeugen. Beide Schlüssel werden in Objektattributen gespeichert um sie weiteren verwenden zu können und anschließend zurückgegeben.

4.3.2. Methode: calc_common_key()

Nach Erhalt des öffentlichen Schlüssel des Kommunikationspartners ermöglicht es diese Methode den gemeinsamen geheimen Schlüssel zu berechnen.

```
1 def calc_common_key(self, k_pub):
2     common_key = self.cyclic_group.scalar_dot_element(self.
    k_priv, k_pub)
3     return common_key
```

Listing 4.14: Methode: calc_common_key()

Dies geschieht durch die Multiplikation des eigenen privaten Schlüssel mit dem gegebenen öffentlichen Schlüssel des Kommunikationspartners. Der berechnete Schlüssel wird folgend zurückgegeben.

4.4. Support Algorithmen

Neben den oben gezeigten Klassen wurde ein Modul mit verschiedenen Algorithmen entwickelt, welche von den anderen Klassen verwendet werden.

4.4.1. Funktion: is_prime()

Die *is_prime*-Funktion ermöglicht es festzustellen ob eine gegebene Zahl eine Primzahl ist. Im Falle einer Primzahl wird *True*, ansonsten *False* zurückgegeben.

```
1 def is_prime(n):
2     if n <= 1:
3         return False
4     for i in range(2, int(math.sqrt(n)) + 1):
```

```
5           if n % i == 0:
6               return False
7       return True
```

Listing 4.15: Funktion: is_prime()

Zunächst wird geprüft ob die gegebene Zahl n kleiner oder gleich 1, da es in dem Fall sicher keine Primzahl ist. Danach wird für jede Zahl von 2 bis $\sqrt{n}$ geprüft ob sie ein Teiler von n ist. Falls ein Teiler gefunden wird, ist n keine Primzahl, falls nicht, ist n eine Primzahl.

4.4.2. Funktion: extended_euclidean_algorithm()

Diese Funktion ist eine rekursive Implementierung des erweiterten euklidischen Algorithmus. Dieser liefert neben dem GGT der beiden Inputparameter a und b auch die Koeffizienten der Gleichung

$$ax + by == GGT$$

. In der Praxis wird die häufig eingesetzt um das multiplikative Inverse einer Zahl in einem endlichen Körper zu berechnen.

```
1  def extended_euclidean_algorithm(a, b):
2      if a == 0:
3          return b, 0, 1
4      else:
5          gcd, x, y = extended_euclidean_algorithm(b % a, a)
6          return gcd, y - (b // a) * x, x
```

Listing 4.16: Funktion: extended_euclidean_algorithm()

4.4.3. Funktion: inverse_mod()

Mittels des erweiterten euklidischen Algorithmus berechnet diese Funktion das multiplikative Inverse einer Zahl a modulo m.

```
1  def inverse_mod2(a, m):
2      gcd, x, y = extended_euclidean_algorithm(a, m)
```

```
3       if gcd != 1:
4           return None
5       else:
6           return x % m
```

Listing 4.17: Funktion: inverse_mod()

Im Fall, das der *GGT* von a und m ungleich 1 ist, was gleichbedeutend damit ist, dass kein Inverses existiert, wird *None* zurückgegeben, ansonsten das multiplikative Inverse.

4.4.4. Funktion: eratosthenes(limit)

Mit Hilfe des Siebes des Eratosthenes können alle Primzahlen bis zu einem Parameter *limit* berechnet, zu der Liste *primes* hinzugefügt und auch angezeigt werden.

```
1  def eratosthenes(limit):
2      primes = []
3      for number in range(2, limit + 1):
4          primes.append(number)
5      # print(primes)
6
7      for prime in primes:
8          for number in primes[primes.index(prime) + 1:]:
9              if number % prime == 0:
10                 primes.remove(number)
11
12     print(primes)
13     return primes
```

Listing 4.18: Funktion: eratosthenes(limit)

Für das Durchlaufen der Zahlen bis zum eingegebenen Limit werden mehrere Schleifen benötigt.

5. Fazit und Ausblick

In dieser Studienarbeit wurden elliptische Kurven in der Charakteristik $p > 3$ untersucht. Die erläuterten Grundlagen in Kapitel 2 halfen dabei, einen Grundstein zu legen. Die Arithmetik wurde anschaulich in Kapitel 3 erläutert. Neben dreien Algorithmen zur Punktbestimmung wurde ebenfalls das diskrete Logarithmusproblem auf elliptischen Kurven sowie der Diffie-Hellmann-Schlüsselaustausch auf elliptischen Kurven je mittels eines Beispiels erklärt. Im letzten Kapitel 4 wurde die Implementierung der Arithmetik und der Rechengesetze in Python implementiert. Weiterhin wurden Hilfsalgorithmen sowie der DHKE über elliptischen Kurven programmiert.

Bei den gewählten Beispielen wurden verschiedenste elliptische Kurven und Primzahlen aus unterschiedenen Quellen verwendet. Innerhalb dieser Studienarbeit wurde kein Algorithmus oder eine andere Möglichkeit gezeigt, wie elliptische Kurven gefunden oder generiert werden können. Dies könnte in einer künftigen Arbeit noch ergänzt werden. Es könnte zum Beispiel ein oder mehrere Algorithmen oder etwas ähnliches vorgestellt werden, welche helfen die Parameter a, b und p der elliptischen Kurve zu berechnen. Im Teil der Grundlagen könnte man das Unterkapitel um die Bestimmung von Primzahlen ergänzen, indem man die Riemannsche Zeta-Funktion $\zeta(x)$ anführt. Man könnte in einer fortsetzenden Studienarbeit die Umsetzung der Arithmetik und der Rechengesetze von elliptischen Kurven in einer anderen Programmiersprache umzusetzen. Beispielsweise könnte man eine maschinennahe Sprache wie C wählen welche kompiliert wird, damit man die Rechnungen effizienter gestalten kann als in einer Programmiersprache wie Python, welche vor dem kompilieren noch interpretiert werden muss. Weiterhin könnte man auf die Geschichte der elliptischen Kurven und mehr auf deren Nutzen in der Kryptographie eingehen.

Literaturverzeichnis

[1] Paulo Ribenboim. *Die Welt der Primzahlen: Geheimnisse und Rekorde*. Springer-Lehrbuch. Springer, Heidelberg, 2., vollst. überarb. und aktualisierte aufl. edition, 2011.

[2] Nikomachos. Das sieb des eratosthenes. In Kai Brodersen, editor, *Einführung in die Arithmetik : Griechisch – deutsch*, page 7. De Gruyter (A), Berlin, Boston, 2021.

[3] Wätjen. *Kryptographie*. Springer Fachmedien Wiesbaden, 2018.

[4] Dr. Reinhold Hübl. *Kryptologie: Manuskript zur Vorlesung*. Fakultät für Technik DHBW Mannheim, 2022.

[5] Rudolf Berghammer. *Mathematik für die Informatik: Grundlegende Begriffe, Strukturen und ihre Anwendung*. Lehrbuch. Springer Vieweg, Wiesbaden, 4., erweiterte und verbesserte auflage edition, 2021.

[6] Christof Paar and Jan Pelzl. *Kryptografie verständlich*. Springer Berlin Heidelberg, 2016.

[7] Waldecker Stroth. *Elementare Algebra und Zahlentheorie*. Mathematik Kompakt. Birkhäuser Cham, Cham, 2. aufl. edition, 2019.

[8] Joseph H. Silverman. *Arithmetic of Elliptic Curves*. Scholars Portal, 2009.

[9] Hankerson, Menezes, Vanstone. *Guide to Elliptic Curve Cryptography*. Springer New York, New York, NY, 2004.

[10] J. Theor. Nombres Bordaux. *Counting Points on elliptic curves over finite fields*.

[11] Guido van Rossum. Python tutorial, o.J.

[12] Travis E. Oliphant. A guide to numpy, o.J.

[13] John D. Hunter. Matplotlib: A 2d graphics environment, 2007.