ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Projektbericht

Projektarbeit
an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Praktische Informatik
der Fakultät für Ingenieurwissenschaften

Implementierung elliptischer Kurven für die Kryptographie

vorgelegt von
Annick Aboa
Hendrik Haas
Mai Manh-Khang

betreut und begutachtet von
Prof. Dr. Peter Birkner

Saarbrücken, 31. März 2021

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

In den letzten Jahren hat die dramatische Zunahme von elektronisch übertragenen Informationen zu einer zunehmenden Abhängigkeit von kryptographischen Verfahren geführt. In unserer modernen vernetzten Welt ermöglicht Kryptographie es Menschen, nicht nur geheime Nachrichten über öffentliche Kanäle auszutauschen, sondern auch Online-Banking, Online-Handel und Online-Einkäufe zu tätigen, ohne befürchten zu müssen, dass die persönlichen Informationen kompromittiert werden [5].
Daher ist unter dem Begriff **Kryptographie**, die Lehre mathematischer Techniken in Bezug auf Aspekte der Informationssicherheit wie Vertraulichkeit, Datenintegrität, Datensauthentifizierung, zu verstehen [3]. Die Dringlichkeit eines sicheren Austauschs digitaler Daten zu gewähren, hat daher in den letzten Jahren zu großen Mengen unterschiedlicher Verschlüsselungsverfahren geführt. Diese können in zwei Gruppen eingeteilt werden nähmlich: symmetrische (mit privaten Schlüsselalgorithmen) und asymmetrische Verschlüsselungsverfahren (mit öffentlichen Schlüsselalgorithmen) [3].

In dieser Arbeit wird der Fokus hauptsächlich auf eine asymmetrische Verschlüsselungstechnik liegen: die **Elliptische-Kurven-Kryptographie** (engl. **Elliptic Curve Cryptography: ECC**) aufgrund ihrer zahlreichen Vorteile gegenüber herkömmlichen kryptographischen Algorithmen. Gemäß den Richtlinien des Nationalen Instituts für Standards und Technologie (NIST) kann eine ECC-Schlüsselgröße von 163 Bit eine gleichwertige bzw. höhere Sicherheit wie ein 1024-Bit des RSA-Algorithmus gewährleisten. Mit guten ECC-Schlüsselgrößen sind nur eine geringere Rechenleistung, sowie ein geringerer Speicher- und Stromverbrauch erforderlich ([10]; [34]). Zudem kann die Technologie in Verbindung mit den meisten Verschlüsselungsmethoden mit öffentlichen Schlüsseln wie RSA und Diffie-Hellman verwendet werden. ECC ist ideal für den Einsatz in eingeschränkten Umgebungen wie Personal Digital Assistenten, Mobiltelefonen und Smartcards.

Im Allgemeinen lässt sich die Elliptische-Kurven-Kryptographie als ein Public-Key- bzw. ein asymmetrisches Verschlüssselungsverfahren definieren, das auf der elliptischen Kurventheorie basiert und zur Erstellung schnellerer, kleinerer und effizienterer kryptografischer Schlüssel verwendet werden kann. Im asymmetrischen Verfahren mit öffentlichen Schlüsseln verfügt jeder Benutzer oder das Gerät, das an der Kommunikation teilnimmt, über ein Schlüsselpaar: einen öffentlichen Schlüssel und einen privaten Schlüssel sowie eine Reihe von Operationen, die den Schlüsseln zugeordnet sind, um kryptographische Operationen wie die Verschlüsselung einer Nachricht auszuführen. Nur der bestimmte Benutzer kennt den privaten Schlüssel, während der öffentliche Schlüssel an alle an der Kommunikation beteiligten Benutzer verteilt wird. Zudem können die Daten, die mit öffentlichen Schlüsseln verschlüsselt sind, nur mit dem privaten Schlüssel entschlüsselt werden ([10]; [13]).

1.2 Problematik

Da ECC dazu beiträgt, eine gleichwertige Sicherheit bei geringerer Rechenleistung und geringerem Ressourcenverbrauch zu erreichen, hat sich ECC zu einem attraktiven und sehr effizienten Public-Key-Krytosystem entwickelt [34]. Ihre Sicherheit basiert jedoch auf die Komplexität, das diskrete Logarithmusproblem in der Gruppe von Punkten auf einer elliptischen Kurve zu berechnen [11], da dieses Problem in nur exponentieller Zeit gelöst werden kann.

Außerdem ist auch die Art der zu verwendeten elliptischen Kurven unter Betrachtung der verwendeten Parameter (z.B. den Koeffizienten der Kurve) optimal auszuwählen [16]. Es existiert bereits mehrere Kurven die vom amerikanischen Standardinstitut NIST festgelegt wurden, obwohl deren Erzeugung allerdings nicht vollständig nachvollziehbar ist, was in amerikanischen Krypto-Standards zu erheblicher Kritik geführt hat [1].
Zudem gibt es eine erhebliche Anzahl potenzieller Schwachstellen für elliptische Kurven, wie z. B. Seitenkanalangriffe und Twist-Security-Angriffe, die bedrohen, die Sicherheit von angebotenen ECC privaten Schlüsseln, ungültig zu machen [36].
Also unabhängig davon, wie sicher ECC theoretisch ist, muss der Algorithmus ordnungsgemäß implementiert werden, da fehlgeschlagene Implementierung von ECC-Algorithmen zu erheblichen Sicherheitslücken in der kryptografischen Software führen können [36].

1.3 Zielsetzung

Ziel dieser Projektarbeit ist es einen Überblick über elliptischen Kurven in der Kryptographie zu geben. Zudem wird eine auf Modular- und Kurvenarithmetik basierende Implementierung der elliptischen Kurven für die kryptographische Anwendung bereitgestellt, die dann zur Erzeugung von Schlüsseln eines ECC-basierten Kryptosystems verwendet wird. Java wurde als bevorzugte Sprache in dieser Arbeit für die Implementierung von elliptischen Kurven gewählt, weil sie gut lesbar und mit einfachen Mitteln eine gute Schnittstelle für viele Webanwendungen bietet, wodurch unsere Implementierung auch von anderen Programmen genutzt werden kann.

1.4 Projektaufbau

Die vorliegende Arbeit ist wie folgt aufgebaut: Nach diesem einleitenden Abschnitt, gibt der zweite Abschnitt einen Überblick über das Thema Im Abschnitt 2 bietet eine Einfuhrung in elliptische Kurven und deren Arithmetik. //TODO

2 Grundlagen von elliptischen Kurven

2.1 Begriffsabgrenzung

Nach Dietmer ist unter dem Begriff **Kryptographie** „die Wissenschaft von geheimen Schreiben zu verstehen." Grundkonzept eines kryptografischen Systems ist also die Verschlüsselung von Informationen bzw. von Daten, um die Vertraulichkeit der Informationen zu gewährleisten [39].

Daten, die über einen unsicheren Kanal wie das Internet übertragen werden, werden mit Hilfe moderner Kryptosysteme so verschlüsselt, dass Unbefugte in einem Szenario, in dem sie auf die Informationen zugegriffen haben, diese nicht frei lesen und verstehen können [24].
Die unverschlüsselte Information wird als *Klartext* (Plaintext, Cleartext), die Verschlüsselte als *Chiffretext* (Ciphertext, Cryptotext) bezeichnet und der Verschlüsselungsprozess des Klartextes wird *chiffrieren* (engl. encryption) genannt. Umgekehrt wird unter *dechriffrieren* (engl. decryption), der Entschlüsselungsprozess eines Chiffretextes, verstanden.

Im Allgemeinen werden die beiden Prozesse durch eine Reihe von Regeln erreicht, sogenannte Verschlüsselungs- und Entschlüsselungsalgorithmen. Der Verschlüsselungsprozess basiert auf einem Schlüssel, der dann zusammen mit den Informationen als Eingabe an einen Verschlüsselungsalgorithmus übergeben wird. Danach können unter Verwendung eines Entschlüsselungsalgorithmus die Informationen mit dem entsprechenden Schlüssel abgerufen werden. Wer einen geheimen Schlüssel besitzt, kann die Informationen in Klartext entschlüsseln [24]. Die folgende Abbildung 2.1 verdeutlicht die Zusammenhänge.

In den 70er und 80er Jahren war die Kryptographie vor allem auf dem militären und diplomatischen Sektor beschränkt. Um geheime Nachrichten zu verschlüsseln, wurden sogenannte symmetrische Verschlüsselungsverfahren (z.B. Caesar-Verschlüsselung ([24])) verwendet, wo nur ein gemeinsamer geheimer Schlüssel sowohl für die Ver- als auch für die Entschlüsselung benutzt wird.

Aus diesem Grund ist es bei der symmetrischen Verschlüsselung sehr wichtig, dass der geheime Schlüssel auf einem sicheren Übertragungsweg an den Empfänger weitervermittelt wird, bevor die verschlüsselten Nachrichten übermittelt werden können. Früher wurde der Schlüssel meist persönlich, in Form eines Botens, übergeben. Da das persönliche Übergeben des Schlüssels sehr umständlich ist und das Risiko besteht, dass der

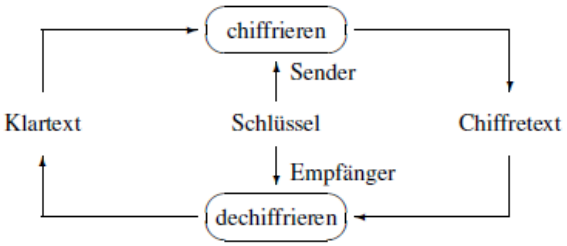

Abbildung 2.1: Grundkonzept der Kryptographie

Schlüssel belauscht oder gestohlen werden könnte, wurden weitere Verschlüsselungsverfahren vorgeschlagen. Dies sind die asymmetrischen Verschlüsselungsverfahren (auch Public-Key-Verfahren genannt) [5].

Im Gegensatz zu einem symmetrischen Verschlüsselungsverfahren erfordern asymmetrische Verschlüsselungsverfahren, nicht nur einen Schlüssel, sondern ein Schlüsselpaar bestehend aus einem öffentlichen Schlüssel und einem privaten Schlüssel [39]. Die Kommunikation erfolgt hier ohne vorhergehenden Schlüsselaustausch und ist jedoch viel langsamer als die Kryptografie mit privaten Schlüsseln. Mit dem privaten Schlüssel werden Daten entschlüsselt oder digitale Signaturen erzeugt, während mit dem öffentlichen Schlüssel Daten verschlüsselt und die Authentizität von erzeugten Signaturen überprüft werden ([32]; [39]). Die Abbildung 2.2 veranschaulicht diese Schlüssel.

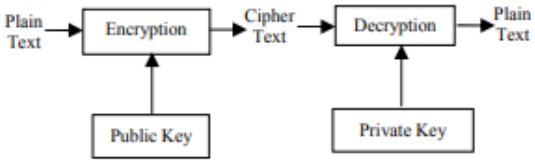

Abbildung 2.2: Public-Key Verschlüsselung

Das erste asymmetrische Kryptosystem, Rivest-Shamir-Adleman-Verfahren (RSA-Verfahren) wurde im Jahr 1977 von Ronald Rivest, Adi Shamir und Leonard Adleman gefunden, danach wurden weitere Kryptosysteme wie Rabin, Elgamal und Elliptische Kurven-Kryptogrphie vorgestellt.
Für unsere Arbeit liegt jedoch der Schwerpunkt auf der Kryptographie mit elliptischen Kurven und die Erzeugung von Schlüsseln, die für das Diffie-Hellman-Verfahren notwendig sind.

Die **elliptische Kurven-Kryptographie (ECC)** ist eine Verschlüsselungstechnik mit öffentlichem Schlüssel, die auf der algebraischen Struktur elliptischer Kurven über endlichen Körper basiert [22] und zur Erstellung schnellerer, kleinerer und effizienterer kryptografischer Schlüssel verwendet werden kann [6].

Die Verwendung einer elliptischen Kurve in der Kryptographie wurde im Jahr 1985 unabhängig voneinander von Miller [37] und Koblitz [26] vorgeschlagen. In den späten 1990er Jahren wurde ECC von einer Reihe von Organisationen wie ANSI [28], IEEE [35], ISO [14], NIST [2] standardisiert und erhielt kommerzielle Akzeptanz [12]. Das bekannteste Verschlüsselungsschema ist das Elliptic Curve Integrated Encryption Scheme (ECIES), das in IEEE- und auch in SECG SEC 1-Standards enthalten ist [30]. Beispiele für die Anwendung von ECC sind unter anderen Mehrzweck-Smartcards wie die neuen deutschen Ausweisdokumente [16].

Außerdem basieren Kryptosysteme mit öffentlichem Schlüssel auf der Lösung bestimmter mathematischen Probleme, z.B. beruht das RSA-Verfahren auf dem Integer Faktorisierungsproblem, DH und ECC basieren auf dem Diskreten Logarithmus-Problem. Bei der Lösung dieser Probleme stellt man im direkten Vergleich jedoch fest, dass herkömmliche Kryptosysteme mit öffentlichen Schlüsseln wie RSA Nachteile aufweisen.

Das Hauptproblem besteht darin, dass die Schlüsselgröße ausreichend groß sein muss, um die Sicherheitsanforderungen auf hohem Niveau zu erfüllen, was zu einer geringeren Geschwindigkeit und einem höheren Bandbreitenverbrauch führt.

Dies ist nicht den Fall, wenn elliptische Kurven (EC) für das Public Key Verfahren eingesetzt sind, da ECC im Vergleich zu RSA für den Benutzer eine gleichwertige Sicherheit bei kleineren Schlüsselgrößen bietet und für Angreifer schwere exponentielle Zeitherausforderung, um in das System einzudringen [12].

Laut einigen Forschern kann ECC mit einem 164-Bit-Schlüssel ein Sicherheitsniveau erreichen, für dessen Erreichung andere Systeme einen 1.024-Bit-Schlüssel benötigen [6]. Es stellte sich heraus, dass ECC das effizienteste Kryptosystem mit öffentlichem Schlüssel ist [25]. Der Grund dafür ist, dass nach Miller und Koblitz das Diskreter-Logarithmus-Problem für elliptische Kurven schwerer sei, als das klassische Diskreter-Logarithmus-Problem ist und zudem auch schnellere Laufzeiten ermögliche.
Bevor das Diskreter-Logarithmus-Problem vorgestellt wird, ist es wichtig zu definieren, was unter einer elliptischen Kurven zu verstehen ist.
Im Allgemeinen ist eine elliptische Kurve eine projektive, algebraische Kurve **C(K)**, auf der sich ein bestimmter Punkt O befindet, der als Punkt unendlich oder Nullpunkt bezeichnet wird [12].

Eine **elliptische Kurve E** über ein Körper K der Charakteristik $\neq 2$ oder 3, lässt sich formal definieren als die Menge der Punkte $(x, y) \in K$ der Weierstraß-Gleichung:

$$y^2 = x^3 + ax + b \quad \text{mit a, b} \in K \text{ [26].} \tag{2.1}$$

Elliptische Kurven in Form von Gleichungen können in **singuläre** und **nicht-singuläre** Gruppen unterteilt werden. In der ECC werden nicht-singuläre Kurven bevorzugt, damit eine Kurve frei von Spitzen oder Selbstüberschneidungen ist [13].

Eine elliptische Kurve wird als **nicht-singulär** bezeichnet, wenn sie keinen Punkt $P = (x, y)$ enthält, an dem ein mathematisches Objekt nicht definiert ist [5] und für die Werte a und b folgende Bedingung $\Delta = 4a + 27b \neq 0$ erfüllt, um eine endliche Abelsche Gruppe zu bilden.

In der abstrakten Algebra ist eine **abelsche** bzw. eine **kommutative** Gruppe, eine Gruppe G, in der das Ergebnis der Anwendung der Gruppenoperation auf zwei Gruppenelemente nicht von ihrer Reihenfolge abhängt. Also wenn $a \cdot b = b \cdot a \quad \forall a, b \in G$, wobei „ $\cdot$ “ eine Verknüpfung auf G.

Ein **Körper** ist eine Menge K mit zwei Verknüpfungen („ $+$ “, „ $\cdot$ “), sodass gilt:

1. $(K, +)$ ist eine abelsche Gruppe. Das neutrale Element wird mit 0 und das inverse von $a \in K$ mit -a bezeichnet,
2. $(K \setminus 0, \cdot)$ ist eine abelsche Gruppe mit neutralem Element 1 und a^{-1} als inversem Element zu $a \in K$.
3. Distributivgesetze:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c) \quad \forall a, b, c \in K$$
$$(a + b) \cdot c = (a \cdot c) + (a \cdot b) \quad \forall a, b, c \in K$$

Ein Körper mit endlich vielen Elementen heißt endlicher Körper. Ein solcher Körper mit p Elementen wird mit $\mathbf{F_p}$ (auch: $\mathbf{GF(p)}$) bezeichnet.

Die **Charakteristik** eines Körpers **K** ist die kleinste positive Zahl **p** mit

$$\underbrace{1 \cdot 1 \cdot 1 \cdot \ldots \cdot 1}_{k \text{ mal}} = 0 \quad \text{, falls sie existiert.}$$

Dies ist beispielsweise für die Körper der rationalen Zahlen $\mathbb{Q}$, der reelen Zahlen $\mathbb{R}$ und der komplexen Zahlen $\mathbb{C}$ der Fall. Für jeden endlichen Körper ist die Charakteristik immer eine Primzahl [39].
K kann ein beliebiger Körper, also etwa $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{C}$ oder ein endlicher Körper $\mathbb{F}$ sein [5]. Die obige Gleichung entspricht der Definition einer elliptischen Kurve über reelen Zahlen.

Obwohl eine elliptische Kurve über die reellen Zahlen ein guter Ansatz ist, um die Eigenschaften einer elliptischen Kurve zu verstehen, erfordert sie eine höhere Rechenzeit, um verschiedene Operationen auszuführen. Sie ist manchmal aufgrund von Rundungsfehlern ungenau. Kryptographie-Schemata erfordern jedoch eine schnelle und präzise Arithmetik [13]. Folglich werden in kryptografischen Anwendungen zwei Arten von elliptischen Kurven verwendet:

- Primzahlen über einem Feld $\mathbf{Z_p}$, wobei p eine Primzahl und p > 3 ist. Alle Variablen und Koeffizienten werden aus einer Menge von ganzen Zahlen von 0 bis p - 1 entnommen und Berechnungen werden über Modulo p durchgeführt [5].
- Binäre Kurve über dem Galois-Feld 2^m, auch bekannt als $GF(2^m)$, wobei alle Variablen und Koeffizienten in GF sind und Berechnungen über $GF(2^m)$ durchgeführt werden.

Da Primkurven analog zur Binärkurve keine erweiterte Bit-Fidding-Operation haben, sind sie für die Software-Implementierung geeignet [13].

In dieser Arbeit wurde die Implementierung von elliptischen Kurven über endliche Körper mit $\mathbf{K} = \mathbf{Z_p}$ berücksichtigt. Dazu wird in Kapitel 4 mehr erläutert. Eine elliptische Kurve über dem endlichen Feld $\mathbf{Z_p}$ enthält alle Punkte (x, y) in der $\mathbf{Z_p} \times \mathbf{Z_p}$ Matrix, die die folgende elliptische Kurvengleichung erfüllt:

$$y^2 = x^3 + ax + b \quad (mod p) \tag{2.2}$$

Dabei sind x und y Zahlen in $\mathbf{Z_p}$ und ähnlich wie im realen Fall ist $\Delta \neq 0$. Alle Punkte (x, y), die die obige Gleichung erfüllen, liegen auch auf der elliptischen Kurve. Der öffentliche Schlüssel ist ein Punkt auf der Kurve und der private Schlüssel ist eine Zufallszahl. Das Hinzufügen von Punkten auf der elliptischen Kurve ist jedoch kein einfacher Prozess, sondern an ein in Polynomialzeit zu lösendes Problem gebunden [23]: Das diskrete Logarithmusproblem.

2.2 Diskretes Logarithmusproblem

Bevor das Problem des diskreten Logarithmus erläutert wird, muss noch der Begriff der zyklischen Gruppe erklärt werden.

Eine **zyklische Gruppe** ist eine Gruppe, deren Elemente als Potenz eines ihrer Elemente dargestellt werden können. Also wenn es ein $a \in G$ gibt mit $\langle a \rangle = G$.

Nehmen wir nun an, dass (G, $\times$) eine multiplikative zyklische Gruppe mit Domainparametern g, h und n ist. Das **diskrete Logarithmusproblem** ist explizit definiert, als das

Problem der Bestimmung einer eindeutigen Ganzzahl x, die zufällig aus dem Intervall [1, p - 1] ausgewählt wird, so dass gilt:

$$g^x = h \mod p$$

vorausgesetzt, dass eine solche ganze Zahl existiert. Der Parameter g ist die Basis des Logarithmus bzw. ein Erzeuger der Gruppe **G** ; n die Anzahl der Elemente in **G** ; der private Schlüssel bzw. das diskrete Logarithmusproblem von h zur Basis g ist die Ganzzahl x, und der öffentliche Schlüssel ist $h = g^x$ [8].

Das **Elliptic Curve Diskrete Logarithmus Problem (ECDLP)** ist ähnlich definiert, aber betrachtet die Weierstraß-Gleichung für eine elliptische Kurve $\mathbf{E} : y^2 = x^3 + ax + b$ über $\mathbb{Z}$ und zwei Punkte P und $Q \in \mathbf{F_p}$. Zu bestimmen ist, eine Zahl k $\in \mathbf{Z}$ mit

$$Q = kP \quad \text{, falls so ein k existiert.}$$

Die Primzahl p, die Gleichung der elliptischen Kurve E und der Punkt P und seine Ordnung n sind die Domainparameter. Der private Schlüssel ist die ganze Zahl k, die gleichmäßig zufällig aus dem Intervall [1, n - 1] ausgewählt wird, und der entsprechende öffentliche Schlüssel ist $Q = kP$.

Daher ist die Hauptoperation bei der ECC die Punktmultiplikation, also die Multiplikation eines Skalars k mit einem beliebigen Punkt P auf der Kurve, um einen anderen Punkt Q auf der Kurve zu erhalten.

Das Lösen von ECDLP ist viel schwieriger als DLP, da die Komplexität der Punktarithmetik in ECDLP im Vergleich zur Ganzzahlarithmetik in DLP zunimmt. Aus diesem Grund ist ECC in der Lage, ein ähnliches Sicherheitsniveau wie RSA bereitzustellen, jedoch mit einer kürzeren Schlüsselgröße, was es wiederum zur beliebtesten Wahl macht [23].

Damit ein auf $\mathbf{F_p}$ basierendes diskretes Logarithmus-System effizient ist, sollten schnelle Algorithmen zur Berechnung der Gruppenoperation bekannt sein. Aus Sicherheitsgründen sollte das Problem des diskreten Logarithmus in $\mathbf{Z_p}$ unlösbar sein [8].
Es gibt viele Algorithmen zur Lösung von **ECDLP**, aber der erfolgreichste ist die Kombination von Pollards Rho- und Pohlig-Hellman-Angriffen mit vollständig exponentieller Laufzeit ([23];[8]). Angriffe resultieren normalerweise aus den Schwächen bei der Auswahl der elliptischen Kurve und des endlichen Feldes (siehe Kapitel 4 und 5), aber die meisten Angriffe können durch korrekte Auswahl der Parameter der elliptischen Kurve vereitelt werden [23]. Daher sollten diese öffentlichen Parameter sicher ausgewählt werden, um alle bekannten Angriffe zu vermeiden [4]. Der nächste Abschnitt beschäftigt sich näher mit diesen Parametern.

2.3 Domänenparameter für elliptischen Kurven

Vor der Implementierung eines ECC-Systems müssen mehrere Entscheidungen getroffen werden. Dazu gehören die Auswahl von Domänenparametern für elliptische Kurven (zugrunde liegendes endliches Feld, Felddarstellung, elliptische Kurve) sowie Algorithmen für Feldarithmetik, elliptische Kurvenarithmetik und Protokollarithmetik.

Die Auswahl kann durch Sicherheitsaspekte, Anwendungsplattform (Software oder Hardware), Einschränkungen der jeweiligen Computer- und Kommunikationsumgebung

(z. B. Speichergröße, Bandbreite) beeinflusst werden [7].

Bevor der Verschlüsselungsprozess gestartet und der verschlüsselte Text transformiert wird, müssen Domänenparameter von beiden Parteien vereinbart werden, die an der sicheren und vertrauenswürdigen Kommunikation über ECC beteiligt sind [34]. Sie werden von einer Gruppe von Benutzern gemeinsam genutzt und können in einigen Anwendungen jedoch für jeden Benutzer spezifisch sein [17]. Es können zwei Arten von elliptischen Kurvendomänenparametern verwendet werden [31]:

- elliptische Kurvendomänenparameter über $\mathbf{Z_p}$ und
- elliptische Kurvendomänenparameter über $\mathbf{F_2^m}$.

2.3.1 Parameter über Z_p

Die Domänenparameter für die elliptische Kurve über $\mathbf{F_p}$ sind ein Sextuple

$$(p, a, b, G, n, h)$$

bestehend aus einer ganzen Zahl **p**, die das endliche Feld $\mathbf{Z_p}$ spezifiziert; der Kurve aus 2.2, den Parametern a und b, einem Basispunkt $G = (x_G, y_G)$ auf $E(\mathbf{Z_p})$, **n** die Ordnung der elliptischen Kurve und dem Cofaktor h, wobei

$$h = \frac{\#E(F_p)}{n}$$

und $\#E(\mathbf{F_p})$ die Anzahl der Punkte auf einer elliptischen Kurve ist ([31]; [12]).

2.3.2 Parameter über F_2^m

Die Domänenparameter für die elliptische Kurve über $\mathbf{F_2^m}$ sind ein Septuple

$$T = (m, f(x), a; b; G; n; h)$$

bestehend aus einer ganzen Zahl m, einem irreduziblen binären Polynom $f(x)$ vom Grad m, Parameter a, b $\in \mathbf{F_2^m}$ der durch die Gleichung der elliptischen Kurve $\mathbf{E}(\mathbf{F_2^m})$:

$$y^2 + xy = x^3 + ax^2 + b \quad \in F_2^m \tag{2.3}$$

definiert sind, einem Basispunkt $G = (xG; yG)$ auf $E(F_2^m)$, eine Primzahl n, die in der Größenordnung von G liegt und die Cofaktor h mit

$$h = \frac{\#E(F_p)}{n}$$

und $\#E(\mathbf{F_p})$ die Anzahl der Punkte auf einer elliptischen Kurve ist. ([31]; [12])

Um einen Angriff auf ECDLP zu vermeiden, muss $|E|$ eine ausreichend große Primzahl sein. Zumindest wird vorgeschlagen, dass $n > 2^{160}$ ist [17].

Für diese Arbeit wurden Parameter in $\mathbf{Z_p}$ bevorzugt. Die Verwendung der elliptischen Kurvenarithmetik macht ECC unter allen vorhandenen kryptografischen Schemata einzigartig. Je nach gewähltem Feld verwendet ECC für seine Operationen modulare Arithmetik oder Polynomarithmetik. Dies wird im nächsten Kapitel präsentiert.

3 Zahlentheoretische Grundlagen

Hoher Durchsatz und geringe Ressourcen sind in vielen Anwendungen die entscheidenden Entwurfsparameter des ECC-Prozessors [29]. Da die Effizienz von ECC-Prozessoren hauptsächlich von modularen arithmetischen Operationen wie modularer Addition, Subtraktion und Multiplikation abhängt, ist der effiziente Entwurf modularer Arithmetik eine sehr anspruchsvolle Aufgabe für die Implementierung eines Hochleistungs-ECC-Prozessors [29].

In diesem Kapitel werden die wichtigsten Arithmetikoperationen über dem Primfeld $\mathbf{Z_p}$, die für die Kryptographie von Bedeutung sind, zusammen mit Beispielen vorgestellt und beschrieben.

3.1 Modulararithmetik

Die modulare Arithmetik ist:

> „ein Prozess zum Reduzieren einer Zahl modulo einer anderen Zahl, wobei dieser Prozess zahlreiche Multi-Präzisions-Gleitkomma-Divisionsoperationen umfassen kann [9]. “

Die modulare Arithmetik bietet endliche Strukturen, die alle üblichen arithmetischen Operationen der ganzen Zahlen aufweisen und die mit vorhandener Computerhardware problemlos implementiert werden können [33]. Eine wichtige Eigenschaft dieser Strukturen ist, dass sie durch Operationen wie Potenzieren zufällig permutiert zu sein scheinen, aber die Permutation kann oft leicht durch eine andere Potenz umgekehrt werden [33]. In entsprechend ausgewählten Fällen ermöglichen diese Vorgänge die Ver- und Entschlüsselung oder die Generierung und Überprüfung von Signaturen [33].

Die Berechnung hier erfolgt genauso wie bei der normalen Arithmetik und der einzige Unterschied besteht darin, dass alle Operationen in der modularen Arithmetik in Bezug auf eine positive ganze Zahl, das Modul, ausgeführt werden. Ziel ist der kleinste positive Rest zu finden.

Im Allgemeinen lässt sich eine modulare Reduktion durch die folgende Gleichung definieren:

$$r = x \mod p = x - \lfloor \frac{x}{p} \rfloor \cdot p$$

, wobei x die zu reduzierende Anzahl modulo p ist, der gefundene Rest r, der im Bereich von [0, p - 1] liegt [9]. Daraus lässt sich die Äquivalenzrelation definieren und man sagt, dass r kongruent zu x modulo p ist.

Eine Zahl r ist **kongruent** bzw. **äquivalent** zu einer Zahl x modulo p, ausgedrückt durch $r \equiv x \pmod{p}$, falls p ein Teiler von (r - x) ist [39].

Dies bedeutet, dass r und x den gleichen Rest haben, wenn sie durch p geteilt werden und $r = k \cdot p + x$ mit $k \in \mathbf{Z}$.

In dieser Arbeit bildet die Klasse *BasicTheoreticMethod.java*, das Grundgerüst der Implementierung von elliptischen Kurven über das Primfeld $\mathbf{Z_p}$. In dieser Klasse wurden folgende Funktionen implementiert:

- *modCalculation*, die den Rest aus der Division zweier ganzen Zahlen bestimmt;
- *isKongruent*, die prüft ob zweier Zahlen den selben Rest nach der Division;
- *modAddition*
- *modSubtraction*
- *modMultiplikation*
- *gcdExtended*
- *hasInverse*
- *multiplicativeInverse*
- *modDivision*
- *modExponentiation*
- *phiFunction*
- *chineseremainder*

Diese Klasse enthält unter anderen zahlentheoretische Methoden, die eine wichtige Bedeutung für die Kryptographie haben. Die erste wichtige Methode ist die Methode der Berechnung von Modulo: $\mathbf{X} \pmod{P}$

3.1.1 Berechnung von X mod Y

Eine naive Methode wurde implementiert, in dem das Finden von $r = x \pmod{p}$ zuerst durch wiederholte Berechnung des Quotienten $q = \frac{x}{p}$ und dann durch wiederholtes Subtrahieren von x mit dem Ergebnis aus der Multiplikation von p mit q, bis das Ergebnis im Bereich von $[0, p-1]$ liegt. Die Leistung der Modulo-Operation hängt vor allem hier von der Leistung des Divisionsalgorithmus ab.

Die naive Art eine ganzzahlige Division zu implementieren, besteht darin, solange den Divisor p von dem Dividend x zu subtrahieren, bis der Dividend x negativ wird, und dabei den Quotienten q hoch zu zählen. Wenn der Dividend x negativ wird, wird er zum Divisor aufaddiert und der Quotient um eins verringert [15].
Die implementierte Methode ist jedoch sehr langsam und kostspielig, denn wenn eine große durch eine kleine Zahl geteilt wird, muss man sehr oft Iterieren [27].

Aber Methoden wie die **Barrett-Reduktion** können verwendet werden, um die Modulo-Operation zu optimieren ([21], [38]).

Eine **Barrett-Reduktion** ist ein Verfahren zum Reduzieren einer Zahl modulo einer anderen Zahl, wobei die Division durch Multiplikationen und Verschiebungen ersetzt

werden können, da die beiden letzteren Operationen viel billiger sind [9].
Also der Quotient $q = \frac{x}{p}$ wird unter Verwendung kostengünstigerer Operationen mit Potenzen einer geeignet gewählten Basis b geschätzt. Für die Barett-Reduktion wird b häufig als Potenz von 2 gewählt werden.

Eine modulabhängige Größe $m = \left\lfloor \frac{b^{2k}}{p} \right\rfloor$ muss berechnet werden, wodurch der Algorithmus für den Fall geeignet ist, dass viele Reduktionen mit einem einzigen Modul durchgeführt werden [8]. Der Quotient q kann nun nur einmal für jedes Modul gleich dem Kehrwert von p berechnet werden [27].

Dies erklärt sich aus der Tatsache, dass beispielsweise jede RSA-Verschlüsselung für eine Entität ein Reduktionsmodul für den öffentlichen Schlüssel dieser Entität erfordert [3].

Das modulare Reduktionsverfahren umfasst [9]:

- die Eingabe des zu reduzierenden Werts x und des Modulos p, wobei dies eine spezielle Form (z.B. NIST-Primzahl) hat,
- das Durchführen der modularen Reduktion von x mod p, wobei die modulare Reduktion umfasst:
  - Berechnen eines genäherten Basisquotienten m,
  - Berechnen einer Quotienten-Näherung q,
  - Berechnen eines genäherten Rests r, und
  - Berechnen einer geschätzten, reduzierten Form des Wert x auf der Basis der Quotienten-Näherung und des genäherten Rests.

Die Tabelle 4.14 veranschaulicht den Barett-Reduktion-Algortihmus.

Algorithmus: Modulo-Berechnung
Input: b > 3, p, $k = \lfloor log_b p \rfloor + 1, 0 \leq x < b^{2k}, m = \left\lfloor \frac{b^{2k}}{p} \right\rfloor$
Output: $x mod p$
1. Berechne $q = \left\lfloor \left\lfloor \frac{x}{b^{k-1}} \right\rfloor \frac{m}{b^{k+1}} \right\rfloor \cdot p$;
2. $r = (x \bmod b^{k+1}) - (q \cdot p \bmod b^{k+1})$;
3. If r < 0 then $r = r + b^{k+1}$;
4. While $r \geq p$ do r = r - p;
5. Return r.

Tabelle 3.1: Barett-Reduktion[2]

Weitere grundlegenden und wesentlichen Operationen für die Kryptographie sind die modulare Addition, modulare Subtraktion und modulare Multiplikation.

3.1.2 Modulare Addition

Die modulare Addition über $\mathbf{Z_p}$ kann mathematisch geschrieben werden als:

$$x + y \mod p$$

wobei x und y die gegebenen Zahlen und p eine Primzahl sind. In dieser Arbeit wurde für die Implementierung der modularen Addition zuerst x und y addiert, dann wird das Ergebnis modulo p berechnet, wenn dies außerhalb des Bereichs von [0, p-1] liegt, sonst wird die Summe direkt ausgegeben.

Seien zum Beispiel x = 17, y = 4, p = 5, dann ist

$$x + y \pmod{p} = \quad 17 + 4 \pmod{5} = \quad 21 \equiv 1 \mod 5$$

Ein kostengünstiger bzw. schnellerer Algorithmus wäre zuerst x und y binär darzustellen und in einem Array von t-Bits zu speichern. Danach sind x und y wortweise bzw. bitweise zu addieren und vom Ergebnis dann p zu subtrahieren, solange es p - 1 überschreitet.

Jede Wortaddition erzeugt zum Beispiel in einer 32-Bit Plattform-Architektur eine 32-Bit-Summe und eine 1-Bit-Übertragsziffer, die zur nächsthöheren Summe addiert wird [2]. Die einfache Addition und die *Addition mit Übertrag* können schnelle Einzeloperationen sein [29]. Die Tabellen 3.2 und 3.3 veranschaulichen die beiden Algorithmen.

Klassischer Algorithmus: Modulare Addition
Input: Modulo p und Zahlen x, y $\in$ [0, p - 1]
Output: $r = x + y \pmod{p}$
1. Berechne r = x + y;
2. If $r \neq 0$, then finde den Rest r = r mod p else r = 0;
3. return r.

(a) 1.Tabelle

Tabelle 3.2: Modulare Addition aus der Klasse *BasicTheoreticMethods*

Algorithmus: Modulare Addition
Input: Modulo p und Zahlen x, y $\in$ [0, p - 1]
$t = \lceil m/32 \rceil$ und $m = \lceil log_2 p \rceil$
Output: $c = (x + y) mod p$
1. $c_0 = x_0 + y_0$;
2. For i from 1 to t-1 do: $c = Add\_With\_Carry(x_i, y_i)$;
3. If the carry bit is set, then subtract p from $c = (c_{t\_1}, ..., c_2, c_1, c_0)$;
4. If $c \geq p$ then $c = c - p$;
5. Return c.

(a) 2.Tabelle

Tabelle 3.3: Empfohlene modulare Addition von amerikanischen Standard NIST für eine 32-Bit Architektur-Plattform [2].

3.1.3 Modulare Subtraktion

Mathematisch kann die modulare Subtraktion geschrieben werden als:

$$(x - y) \mod p$$

wobei x und y die gegebenen Zahlen und p die Primzahl sind. Der einfachste Weg diese Operation durch zu führen, besteht darin, die beiden Zahlen x und y zuerst zu subtrahieren und dann der kleinste positive Rest im Bereich von [0, p-1] zu berechnen, in dem das Ergebnis der Subtraktion von (x - y) durch p geteilt wird. Der Algorithmus sieht genauso aus wie der Algorithmus der modularen Addition, wobei im ersten Schritt anstatt eine Addition, eine Subtraktion durchgeführt wird.

Seien zum Beispiel x = 15, y = 23, p = 5, dann ist

$$(x - y) \mod p = \quad (15 - 23) \mod 5 = \quad -8 \equiv 2 \mod 5$$

In der effizienteren Art, Modulo-Subtraktion durchzuführen, wird das Übertragungsbit nicht mehr als `Carry-Bit`, sondern als `Borrow-Bit` (Ausleih-Bit) interpretiert [2]. Die Operation wird dann ähnlich wie die modulare Addition implementiert.

3.1.4 Modulare Multiplikation

Die modulare Multiplikation ist eine der teuersten und zeitaufwändigsten Operationen der Kryptographie über Z_p. Aber um ein höheres, leistungsstarkes Kryptosystem zu entwickeln, muss eine effiziente Implementierung der modularen Multiplikation erfolgen [29]. Mathematisch kann die modulare Multiplikation-Operation ausgedrückt werden als:

$$x \cdot y \mod p$$

Um Modulo-Multiplikationen durchzuführen, wurde in dieser Arbeit die Zahlen x und y einfach multipliziert und dann die ganzzahlige Division durch p zu verwendet, um den Rest zu erhalten. Dies bedeutet auch, dass die Leistung des ganzen Algorithmus, genauso wie bei der modularen Addition und Subtraktion, von Modulo-Operationen bzw. von der Leistung des Divisionsalgorithmus abhängt.
Seien zum Beispiel: x = 35, y = 7, p = 5, dann gilt:

$$x \cdot y \mod p = \quad 35 \cdot 7 \mod 5 = \quad 245 \equiv 0 \mod 5$$

Neben der Barett-Reduktion Methode gibt es auch eine andere Methode, die die Modulo-Operationen viel schneller als die reguläre klassische Methode durchführen kann: Die **Montgomery-Multiplikation**.

Die **Montgomery-Multiplikation** ist ein Verfahren zur modularen Multiplikation, bei der die traditionelle Divisons-Operation vermeidet wird und anschließend nur Multiplikationen, Additionen und Verschiebungen verwendet werden [32]. Die Zahlen x und y werden binär dargestellt. Der modulare multiplikative Algorithmus ist in Tabelle 3.4 angegeben.

Das Verfahren ist für eine einzelne modulare Multiplikation nicht effizient, kann jedoch effektiv bei Berechnungen wie der modularen Potenzierung verwendet werden, bei denen viele Multiplikationen für eine gegebene Eingabe durchgeführt werden [8]. Im nächsten Unterabschnitt werden Methoden zur Berechnung der modularen Potenzierung erläutert.

Algorithmus: Modulare Multiplikation

Input: p, x und y zwei positive k-Bit Ganzzahlen, x_i, y_i: i-te Bit in x und y
Output: $m = x \cdot y \bmod p$

1. m = 0;
2. For i = 0 to k-1
   2.1 $m = m + (x \cdot y_i)$;
   2.2 If $(m_0 = 1)$ then $m = m/2$ else $m = (m + p)/2$;
3. Return m.

Tabelle 3.4: Montgomery-Multiplikation [32]

3.1.5 Modulare Potenzierung

Das Problem der modularen Potenzierung lässt sich mathematisch definieren als:

$$A = x^k \mod p \quad mit \quad x \in \mathbf{Z_p} \quad und \quad 0 \leq k < p$$

Die modulare Potenzierung ist die kostspielige, aber entscheidende und bedeutungsvolle Methode für viele kryptographische Protokolle.

Algorithmen zur schnellen modularen Potenzierung wie das Square-and-Multiply-Verfahren, basieren vor allem auf der Auswertung der Binärdarstellung des Exponenten k [19]. Dort wird der Exponent k binär dargestellt, als k = $\sum_{i=0}^{t} k_i \cdot 2^i$ mit $k_i \in \{0, 1\}$ und die Auswertung kann dabei je nach Algorithmen entweder von links nach rechts oder von rechts nach links erfolgen [19].

Bei einem k-Bit-Exponenten werden dann für die Potenzierung höchstens k Quadrate und k Multiplikationen benötigt [19]. Selbst wenn sie effizient mit dem wiederholten Square-and-Multiply-Verfahren durchgeführt wird, erreicht sie eine Bit-Komplexität von $\mathcal{O}(\log n^3)$ [3].

Gegeben sei beispielsweise $x^k = x^{13}$. Zu berechnen ist, $2^{13} \mod 7$. Dann ist die binäre Darstellung von k = 1101. Der Exponent k lässt sich wie folgt auswerten:

$$((((0) \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 13$$

Daraus lässt sich x^{13} bzw. 2^{13} wie folgt aufteilen:

$$x^{13} = ((((x^0)^2 \cdot x^1)^2 \cdot x^1)^2 \cdot x^0)^2 \cdot x^1$$

$$2^{13} = ((((2^0)^2 \cdot 2^1)^2 \cdot 2^1)^2 \cdot 2^0)^2 \cdot 2^1$$

Es ergibt sich, dass $2^{13} \equiv 2 \mod 7$.

Die schnelle modulare Potenz kann jedoch auch in einfacher Weise realisiert werden [18]. Tatsächlich, kann x^{13} auch dargestellt werden, als

$x^{13} = x^{12} \cdot x$ und x^{12} weiterhin als $x^{12} = (x^6)^2$.

Ausgehend von dieser Beobachtung, ergibt sich die folgende Rekursionsformel [18]:

$$x^k = \begin{cases} 1 & \text{falls } k = 0 \\ x^{k-1}.x & \text{falls } k \quad \textit{ungerade} \\ (x^{k/2})^2 & \text{falls } k \quad \textit{gerade} \end{cases}$$

In dieser Arbeit wurde für die Implementierung der modularen Potenz, die Idee der Rekursionsformel verfolgt. In dieser rekursive Implementierung wurde die Binärdarstellung des Exponenten k nicht explizit benötigt.

Der Algorithmus startet mit Basisfällen, in dem geprüft wird, ob die Zahlen x, k und p in den richtigen Bereichen liegen. A wird am Anfang auf 1 gesetzt (A = 1). Wenn k gleich null ist, wird A = 1 zurückgegeben. Wenn k größer null ist, wird in jedem Schritt geprüft, ob der Exponent k ungerade bzw. ob die aktuelle binäre Zahl eins ist. Wenn ja, wird das Ergebnis A mit x multipliziert und das Modulo berechnet; ansonsten erfolgt zuerst eine Rechtsverschiebung (k wird halbiert) und danach wird x quadriert. Danach wird der Rest bestimmt, damit $x \in [0, p-1]$ bleibt. Diese Schritte werden durchgeführt, solange der Exponent größer 0 ist. Der Algorithmus wird in der Tabelle 3.5 dargestellt. Also die Leistung dieser Methode hängt von der Leistung des Modulo-Algorithmus ab.

Algorithmus: Modulare Potenz
Input: p, x, $k \in \mathbf{Z}$
Output: $x^k \mod p$
1. A = 1;
2. If $(k = 0)$ then return A;
3. If $(k < 0)$ then $k = \lvert k \rvert$;
x = $(x^{-1} \mod p) \mod p$;
4. x = x mod p;
5. while (k > 0)
5.1 If $(k \,\&\, 1) = 1$ then A = A · x *mod* p;
5.2 k $\gg 1$;
5.3 x = x · x mod p;
6. Return A.

Tabelle 3.5: Modulare Potenz aus der Klasse *BasicTheoreticMethods.java*

In den meisten wissenschaftlichen Artikeln, liegt $k \in [0, p-1]$, aber in dieser Arbeit es wurde angenommen, dass k $\in \mathbf{Z}$ liegt. Betrachtet wurde also die Fälle, wo der Exponent k auch negative Werte haben kann, um möglichst alle Sicherheitslücken auszuschließen.

In diesem Fall wird für die Berechnung der modularen Potenz, der Absolutwert des Exponenten genommen, aber vorher nimmt x den resultierenden Wert aus der Berechnung der modularen multiplikative Inverse von x und p an.

3.1.6 Modulare multiplikative Inverse

Das modulare Inverse ist eine weitere wichtige Methode der Kryptographie. Wenn a eine Restklasse aus $\in \mathbf{Z_p}$ (geschrieben $[a]_p$) und p eine Primzahl ist, ist das modulare Inverse a^{-1} eine ganze Zahl, die die Beziehung

$$a \cdot a^{-1} \equiv 1 \mod p$$

erfüllt.

Das modulare Inverse wird mit Hilfe des erweiterten euklidischen (EEA) bestimmt. Dort sind die ganzen Zahlen x und y zu finden, für die die folgende Gleichung erfüllt ist: $ggT(a, p) = 1 = ax + py$.
Also muss der $ggT(a, p) = 1$ sein, da p prim ist. Wenn dies den Fall lässt sich feststellen, dass a und p **Koprime** sind.

Erweiterter Euklidischer Algortihmus

Der erweiterte Euklidische Algorithmus basiert auf dem euklidischen Algorithmus, der den ggT von zwei ganzen Zahlen durch wiederholte Anwendung des Divisionsalgorithmus ermittelt. Aber bei der Berechnung des ggT wird auch der Wert von x verfolgt. Im Vergleich zu anderen Algorithmen, die als Ausgabe sowohl den Wert des ggT als auch die Werte von x und y haben, gibt der für diese Arbeit implementierte Algorithmus 3.7 nur den Wert vom ggT zurück.

In diesem Algorithmus wird die Funktion *gcdExtended* rekursiv aufgerufen, um den ggT (a, p) zu berechnen. Solange der Wert von p größer null ist, werden die Ergebnisse aus dem rekursiven Aufruf vom $ggT(p \mod a, a)$ bestimmt und die Werte von a und p dementsprechend aktualisiert. Danach werden die Koeffizienten x und y berechnet und den ggT zurückgegeben.

Algorithmus: EEA

Input: Positive BigInteger a, p mit a $\geq p$
Output: d = ggT(a, p), für die es gilt: $d = ax + py$

1. If a = 0 then d = p, x = 0, y = 1 und return p;
2. If p = 0 then d = a, x = 1, y = 0 und return a;
3. $x_1 = 1, y_1 = 1$;
4. while (p > 0)
   4.1 $d = p \cdot x_1 + a \cdot (y_1 - (\lfloor p/a \rfloor \cdot x_1)$;
   4.2 a = p, $p = a - \lfloor a/b \rfloor \cdot p$;
5. $x = y_1 - \lfloor p/a \rfloor \cdot x_1$;
6. y = x_1 ;
7. Return d.

Tabelle 3.6: Erweiterter Euklidischer Algorithmus aus der Klasse *BasicTheoreticMethods.java*

Der erste Schritt des euklidischen Algorithmus besteht darin, die größere Ganzzahl durch die Kleinere zu teilen. Dann wird der Divisor wiederholt durch den Rest geteilt, bis der Rest 0 ist. Der ggT ist dann der letzte Rest ungleich Null in diesem Algorithmus.

Allerdings die Voraussetzung für die Bestimmung der Inverse, dass $ggT(a, p) = 1$ sein muss. Wenn dies der Fall ist, können durch Umkehren der Schritte im euklidischen Algorithmus die ganzen Zahlen x und y gefunden werden.

Dies kann erreicht werden, indem die Zahlen als Variablen behandelt werden, bis den Ausdruck

$$1 = x \cdot a + y \cdot p$$

erhalten wird, der eine lineare Kombination unserer Anfangszahlen ist. Daraus es folgt $x \cdot a \equiv 1 \mod p$ und x ist dann das multiplikative Inverse von a. Die Tabelle 3.7 veranschaulicht den Algorithmus zur Berechnung des modularen multiplikativen Inverses.

Algorithmus: Modulares Inverse
Input: $a \in \mathbf{Z_p}$, p eine Primzahl
Output: $a \cdot a^{-1} \mod p$
1. $x = 0, y = 1, x_1 = 1, y_1 = 0, m = p$;
2. Prüfe ob a ein Inverse hat;
3. If a = 1 then x = 1;
4. while (p > 0)
4.1 q = a/p; t = a; a = p;
4.2 $p = t \mod p$;
4.3 t = x;
4.4 $x = x_1 - q \cdot x$;
4.5 x_1 = t;
4.6 t = y;
4.7 $y = y_1 - q \cdot x$;
4.8 $y_1 = t$;
5.If x < 0 then x = x + m;
6. Return x.

Tabelle 3.7: Modulares multiplikatives Inverse aus der Klasse *BasicTheoreticMethods.java*

Zum Beispiel, gesucht wird das Inverse von $[17]_{53}$.
EEA liefert:

$$\begin{aligned} 53 &= 3 \cdot 17 + 2 \\ 17 &= 2 \cdot 2 + \boxed{1} \\ 2 &= 2 \cdot 1 + 0 \end{aligned}$$

Es ergibt sich, dass ggT(17, 53) = 1. Es folgt:

$$\begin{aligned} 1 &= 25 \cdot 17 + (-8) \cdot 53 \\ 1 &\equiv 25 \cdot 17 \pmod 5 3 \end{aligned}$$

d.h. das Inverse von $[17]_{53}$ ist $[25]_{53}$.

Das modulare multiplikative Inverse lässt sich auch durch die modulare Potenz unter Verwendung des Satz von Euler berechnen.
Satz von Euler Sei $n \in \mathbb{N}$, die Menge der natürlichen Zahlen. Dann gilt für alle a $\in \mathbb{N}$, die teilerfremd ($ggT(a, n) = 1$) zu n sind also [20]:

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

mit $\varphi(n)$, die Euler'sche φ-Funktion, die die Anzahl der Elemente der reduzierten Menge der Reste modulo n bezeichnet [39].

Durch modulare Potenzierung lässt sich zwar das Inverse einer Restklasse leicht bestimmen, aber sie ist im Vergleich mit der Berechnung mit dem erweiterten euklidischen Algorithmus etwas langsamer. Außerdem muss der Wert der Variable $\varphi(n)$ und damit die Kenntnis der Primfaktorzerlegung von n vorher bestimmt sein [20].

Bestimmung der Euler'sche Funktion für eine natürliche Zahl n

Nach dem Satz von Euler gilt für jedes $a \in \mathbf{Z_n^*}$

$$a^{\varphi(n)} \mod n = 1$$
$$a^{\varphi(n)-1} \mod n = a^{-1}$$

Die Euler'sche φ-Funktion ist gleich der Anzahl der zu n teilerfremden Zahlen zwischen 1 und n. Man schreibt

$$\varphi(n) = |\{1 \leq a < n, mit \quad ggT(a, n) = 1\}|$$

Für Primzahlen gilt $\varphi(p) = p - 1$.

Zur Bestimmung der Anzahl der positiven ganzen Zahlen $\varphi(n) \leq n$, die relativ prim zu n sind, wurde in dieser Arbeit eine naive Implementierung durchgeführt.
Der Algortihmus startet mit der Initialisierung des Ergebnis $\varphi(n)$ = n. Es wird angenommen, dass sowohl $\varphi(0)$ als auch $\varphi(1)$ gleich eins sind. Dann laufen wir iterativ über alle Zahlen durch, die kleiner oder gleich der Quadratwurzel von n sind. Es wird für jede Zahl p überprüft, ob diese Zahl n teilt. Wenn dies den Fall ist, werden alle Vielfachen der jeweiligen Zahl entfernt, indem diese wiederholt mit n geteilt werden.
Der Grund dafür ist, dass der ggT von Primfaktoren und Ihre Vielfache ungleich eins ist. In jeder Iteration wird zusätzlich das Ergebnis um n/p reduziert, um die Euler'sche $\varphi(n)$-Funktion zu erhalten. Wenn die Zahl p nicht prim ist, dann wird direkt geprüft, ob diese größer eins ist. Wenn ja dann wird das Ergebnis dementsprechend um n/p reduziert. Die Tabelle 3.8 stellt den Algorithmus für die Berechnung der $\varphi(n)$-Funktion dar.

Da wir mit sehr großen Zahlen arbeiten, ist diese Implementierung jedoch nicht effizient.
Eine effiziente Lösung wäre, anstatt auf alle Zahlen durchzulaufen, nur auf alle Primzahlen, die kleiner oder gleich der Quadratwurzel von n sind, durchzulaufen. Diese Primzahlen können vom Anfang an zufällig z.B. mitHilfe des Fermat-Tests bestimmt und gespeichert werden. Im Kapitel 4 wird auf die Idee des Fermat-Test-Algorithmus näher

Algorithmus: Phi-Funktion

Input: $n \in \mathbf{N}$
Output: $\varphi(n)$

1. $\varphi(n) = n$;
2. If $(n = 1 || n = 2)$ then return $\varphi(n) = 1$;
3. For $(p = 2; p * p \leq n; p++)$
   3.1 If $(n \mod p) = 0$
      3.1.1 $\varphi(n) -= \varphi(n)/p$;
      3.1.2 while $(n \mod p) = 0$ n/p ;
4. If (n > 1) then $\varphi(n) -= \varphi(n)/p$;
5. Return $\varphi(n)$.

Tabelle 3.8: Phi-Funktion aus der Klasse *BasicTheoreticMethods.java*

eingegangen.

Zum Beispiel für p = 7, eine Primzahl gilt:

$\varphi(n) = |\{1,2,3,4,5,6\}| = 6 = 7 - 1 = p - 1$

$\varphi(4) = |\mathbf{Z}_4^*| = |\{[1]_4, [3]_4\}| = 2$

Mit Hilfe des modularen multiplikativen Inverses lässt sich leicht die modulare Division zwischen 2 Zahlen bestimmen.

3.1.7 Modulare Division

Die modulare Division lässt sich mathematisch definieren als:

$$x/y \mod p \quad = \quad x \cdot y^{-1} \mod p$$

Bei der Bestimmung der modularen Division wird zuerst y^{-1}, das modulare Inverse von y über p berechnet. Danach wird die modulare Multiplikation von $x \cdot y^{-1}$ durchgeführt. Diese Operation ist definiert bzw. erfolgt, nur wenn das modulare Inverse von y existiert.
Gegeben sei zum Beispiel x = 8, y = 3, p = 5.

$8/3 \mod 5 \quad = \quad 8 \cdot 3^{-1} \mod 5 \quad = \quad 8/3 \equiv 1 \mod 5$

3.2 Chinesischer Restsatz

Eine weitere Methode, die eine wichtige Rolle in der Entschlüsselung bzw. in der Generierung von Schlüsseln für die Kryptographie spielt, ist der chinesische Restsatz. Der chinesische Restsatz (engl. Chinese Remainder Theorem: CRT) besagt, dass wenn 2 Zahlen p und q Koprime sind (also,wenn ggT(p,q) = 1), dann hat das Gleichungssystem:

$$x \equiv a \mod p$$
$$x \equiv b \mod q$$

eine eindeutige Lösung für $x \mod p \cdot q$.

Also wenn p und q Koprime sind, existieren 2 Zahlen m_1 und m_2, so dass $m_1 \cdot$ p + $m_2 \cdot$ q = 1 gilt. Um m_1 und m_2 zu bestimmen, wird der EEA verwendet und daraus ergibt sich, dass

$$x = a \cdot m_2 \cdot p + b \cdot m_1 \cdot q, \quad \text{für 2 Gleichungen ist.}$$

Für beliege Gleichungen:

$$x = \sum_{i=1}^{k} a_i \cdot N_i \cdot R_i \mod N, \quad \text{mit k, die Anzahl der Gleichungen;} \tag{3.1}$$

$$N_i = N/n_i \quad \text{und} \quad R_i = N_i^{-1} \mod N.$$

In der Tat ermöglicht dieser Satz, eine Nachricht zu entschlüsseln, in dem am Anfangs eine geheime Nachricht M in beliebige Hälften (hier in 2 Hälften m_1, m_2) aufgeteilt wird und dann das Modulo jede diese Nachricht berechnet, wenn die Faktorisierung des öffentlichen Schlüssels $N = p \cdot q$ bekannt ist. Danach werden diese Nachrichten in einer Zahl x wieder kombiniert sein und anschließend wird der private Schlüssel berechnet.

Gegeben sind zum Beispiel zwei Zahlen p = 5 und q = 7. Gesucht ist die Zahl x für die gilt:

$$x \equiv 2 \mod 5$$
$$x \equiv 3 \mod 7$$

Berechnung des ggT (5,7) mittels des EEA ergibt 1 = 3 · 5 - 2 · 7, also es existieren 2 Zahlen m_1, m_2, mit $m_1 = 3$ und $m_2 = -2$.
Dann ist $x = 2 \cdot -2 \cdot 7 + 3 \cdot 3 \cdot 5 = -28 + 45 \equiv 17 \mod 35$.

Für diese Arbeit wurde für die Implementierung der Gleichung 3.1 betrachtet. Zwei Array-Listen wurden genutzt, um alle Reste a_i und alle Modulo-Werte N_i zu speichern. K wird als die Größe der Liste genommen und der Algorithmus wird nur laufen, für Listen gleicher Länge. Die Tabelle 3.12 stellt den entsprechenden Algorithmus dar.

Die Gleichung aus 3.1 wird als Gauß-Algorithmus bezeichnet. Die Berechnungen können in $\mathcal{O}(\log n^2)$ Bitoperationen durchgeführt werden [3].

3.3 Polynomarithmetik

3.3.1 Allgemein

Nun folgt die Rechnungen in das Unbekannte. Hiermit ist gemeint, das wir zu jeder Rechnung bis jetzt alle Werte hatten.

$(a_1 \cdot r^n + a_2 \cdot r^{n-1} + ... + a_{n-1} \cdot r + a_n \cdot r^0)\ mod\ m$

Hierbei könnte man r für eine beliebige Zahl im $r \in \{0, 1, ..., m-1\}$ auswählen. Bei dieser Stelle wissen wir aber nicht was genau r ist und wollen r durch das Symbol x , bzw. einem Polynom ersetzen. Auf diese Weise landen wir in die Polynomarithmetik.

Algorithmus: Chinesischer Restsatz

Input:$a = \{a_1, ..., a_k\}, n = \{n_1, ..., n_k\}$, wobei alle n_i co-prime sind
Output: $x = \sum_{i=1}^{k} a_i \cdot N_i \cdot R_i \mod N$

1. x = 0;
2. k = n.size(); 2. If $(k > a.size())$ then k = a.size();
3. For $i = 0$ bis k-1 compute product of all moduli N =$n_1 * ... * n_k$;
4. For $i = 0$ bis k-1
   4.1 compute $N_i = N/n_i$;
   4.2 compute Inverse $R_i = N_i^{-1} \mod n_i$;
   4.3 x = x + $a_i \cdot R_i \cdot N_i$
5. Return $x \mod N$.

Tabelle 3.9: Chinesischer Restsatz aus der Klasse *BasicTheoreticMethods.java*

Natürlich darf man die Schreibformen nicht vernachlässigen, da man sonst durcheinander kommt. Bei den Polynomen gibt es Potenzen $x^0 = 1, x^1 = x, x^2, x^3, ...$ Jede Potenz hat seinen eigenen Koeffizienten.
Man reiht die Polynome meistens mit fallender Potenzen $(2x^2 + x + 1)$ oder aufsteigender Potenz $(1 + x + 2x^2)$.

Daraus ergibt sich die allgemeine Formel:
$(a_1 \cdot x^n + a_2 \cdot x^{n-1} + ... + a_{n-1} \cdot x + a_n \cdot x^0)\ mod\ m$

Wenn $a_i > m$ oder $a_i < 0$ sind, werden diese durch mod m einzeln berechnet.

Beispiel:

$(4x^4 - 5x^3 + 24x^2 + 5x + 10) mod 13 = 4x^4 + 8x^3 + 11x^2 + 5x + 10$

Mit diesen Gedanken geht man rüber zu den Grundrechenoperationen Addition und Multiplikation

Addition

Allgemeine Definition Addition
$((a_0x^n + ... + a_{n-1}x^1 + a_n) + (b_0x^n + ... + b_{n-1}x^1 + b_n))\ mod\ m = \sum_{i=0}^{n}(a_i + b_i) \cdot x^{n-i}\ mod\ m$

Tabelle 3.10: Allgemeine Definition Addition

Beispiel:

$((4x^2 + 2x + 5) + (x^2 + 6x + 1))\ mod\ 7 = 5x^2 + x + 6$

Multiplikation

Allgemeine Definition Multiplikation
$((a_0x^n + ... + a_{n-1}x^1 + a_n) * (b_0x^n + ... + b_{n-1}x^1 + b_n))\ mod\ m = \sum_{i=0}^{n}\prod_{j=0}^{n}(a_i * b_j) \cdot x^{2n-i-j}\ mod\ m$

Tabelle 3.11: Allgemeine Definition Multiplikation

Beispiel:

$((4x^2 + 2x + 5) + (x^2 + 6x + 1))\ mod\ 7 = 5x^2 + x + 6$

Subtraktion

Allgemeine Definition Subtraktion
$((a_0x^n + ... + a_{n-1}x^1 + a_n) - (b_0x^n + ... + b_{n-1}x^1 + b_n))\ mod\ m = \sum_{i=0}^{n}(a_i - b_i) \cdot x^{n-i}\ mod\ m$

Tabelle 3.12: Allgemeine Definition Subtraktion

Beispiel:

$((4x^2 + 2x + 5) - (x^2 + 6x + 1))\ mod\ 7 = 3x^2 + 3x + 4$

Division

Die Polynomdivision kann man anhand eines Beispiels am besten verstehen.

$$
\begin{array}{rl}
\big(& 2x^5 - 13x^4 + 17x^3 \ - x^2 + 10x + 8 \big) : \big(2x^2 - 3x\big) = x^3 - 5x^2 + x + 1 + \dfrac{13x + 8}{2x^2 - 3x} \\
\underline{-\,2x^5 \ + 3x^4} & \\
-\,10x^4 + 17x^3 & \\
\underline{\ \ 10x^4 - 15x^3} & \\
2x^3 \ - x^2 & \\
\underline{-\,2x^3 + 3x^2} & \\
2x^2 + 10x & \\
\underline{-\,2x^2 \ + 3x} & \\
13x + 8 &
\end{array}
$$

4 Endliche Körper

4.1 Primzahl-Generator

In der Kryptographie arbeitet man mit Primzahlen, jedoch nicht mit kleinen Zahlen.
Aber woher weißt man, ob es sich um eine Primzahl handelt? Bei relativ kleinen Zahlen, kann man dies sehr schnell herausfinden. Jedoch bei höheren Primzahlen wird es schwieriger die Zahlen zu überprüfen, ob es sich auch um eine Primzahl handelt. Es gibt mehrere Primzahl-Test-Algorithmen, der bekannteste von allen ist dieser hier:

Algorithmus: Probedivision
Input: Eine Zufällige Zahl $n \in \mathbb{N}$
Output: true = Primzahl, false = Keine Primzahl
1. Beginne bei i = 2
2. ist n ein vielfaches von i?
true: return false
3. ist $i > \sqrt{n}$?
false: inkrementiere i und gehe zu 2
4. return true

Tabelle 4.1: Einfacher Primzahl-Test

Man erkennt hier deutlich, das es bei kleinen Zahlen relativ zügig gehen kann.
Jedoch bei größeren Zahlen wird es eine ziemlich lange Zeit dauern.

Mit diesen Gedanken, hat man nach Algorithmen gesucht, welche diesen einfachen Primzahl-Test ersetzen sollen.

Die bekanntestens sind:

- Sieb des Eratosthenes
- Sieb von Atkin
- Probabilistische Primzahltests
- und viele weitere.

Von all den aufgelisteten Tests, wird der Fermat-Test (probabilistischer Primzahltest) angeschaut.

4.1.1 Fermat-Test

Der Fermat-Test beruht auf dem kleinen fermatischen Satz. Dieser lautet wie folgt:
$a^{n-1} \equiv 1 \; mod \; n$

Zuerst schauen wir den Algorithmus an.

Algorithmus: Fermat-Test
Input: Eine zufällige Zahl $n \in \mathbb{N}$
Output: true = Pseudoprimzahl, false = keine Primzahl
1. Wähle eine Zahl $a \in 2, 3, 4, ..., n-2$
2. ggT berechnen. Überprüfe ob die Zahlen teilerfremd sind.
-> Nicht teilerfremd => gemeinsamer Teiler => return false
3. Falls die Zahlen teilerfremd sind, so führe folgende Formel aus:
$a^{n-1} \equiv 1 \; mod \; n$
4. Wiederhole für eine gewisse Anzahl an Versuchen.
Merke Anzahl bestandener Versuche.
5. F-Zeuge > F-Lügner? -> true: return true
-> false: return false

Tabelle 4.2: Fermat-Test

Diese Lösungsmöglichkeit sieht zum ersten schön aus. Jedoch beinhaltet dieser Algorithmus einen Fehler.
Es gibt bestimmte Kombinationen, in der die untersuchte Zahl als Primzahl anerkannt wird, jedoch in Wahrheit keine Primzahl ist.

Anhand folgendem Beispiel können wir erkennen, wieso das ist:
Es wird die Zahl 15 genommen und führt von 2-15 die Zahlen durch und listet es in einer Liste auf.

Vielfache von 3
3, 6, 9, 12

Tabelle 4.3: Vielfache von 3

Vielfache von 5
5,10

Tabelle 4.4: Vielfache von 5

F-Zeuge
4, 11, 14

Tabelle 4.5: F-Zeuge

F-Lügner
2,7, 8, 13

Tabelle 4.6: F-Lügner

Würde man nur F-Zeugen bekommen, wird die Zahl 15 als Primzahl angedeutet. Die Zahl 15 ist aber keine Primzahl. Aus dem Grund ist es wichtig den Test mehrmals durchzuführen. Dadurch verringert sich die Fehlerwahrscheinlichkeit, aber es erhöht sich die Laufzeit.

Algorithmus im Programm

Um eine zufällige Zahl zu generieren, gibt man die Länge in Bits mit. In diesem Projekt wurde eine Primzahl, mit einer Bit-Länge von 8192 erwartet.

Algorithmus: Primzahl-Generator
Input: Länge der Primzahl Output: Primzahl 1. Zahl generiert 2. Fermat-Test anwenden, mit einer gewissen Anzahl an Versuchen false: beginne bei 1. 3. return Primzahl

Tabelle 4.7: Primzahl-Generator

Führt man den Algorithmus aus, so kommt man eventuell auf die Primzahl :
70 6815 0405 5962 8624 0116 5582 8814 1890 4999 7822 5925 9076 9503 5507 1619 2423 0111 6620 5520 8288 3610 7731 9769 4969 0297 4888 4070 3257 5833 8064 6528 7972 3291 7661 0446 8045 1640 0051 6702 9150 4361 6802 8389 3226 0098 4356 6161 1330 4262 7433 1919 7858 0365 6198 8262 0250 9889 0007 0954 7574 9383 1886 5361 3190 7877 2205 3670 4087 2171 7091 6374 3838 8716 3932 5296 3638 2140 4949 1393 3782 5072 4723 8804 6806 7614 3346 5804 5628 0683 2443 5101 6267 9171 2074 5458 0355 7929 7744 9444 1421 5092 6350 1162 8496 6988 9236 0338 3073 1279 6721 0801 9128 7263 5150 8342 4633 7071 8726 3737 9368 5065 8952 9713 4329 7856 0992 6224 2713 6350 4753 0207 8413 7888 8748 0655 9121 6942 9384 5841 1784 2617 6752 2643 0070 3556 3276 6686 6523 6072 2767 0478 8204 3629 2550 8690 0935 0632 7571 2249 8673 5817 9463 3439 3486 6162 1529 9621 3988 9362 9652 6810 5885 7633 5373 9720 0378 2639 0125 7017 6324 9307 8768 5224 1751 1885 3081 3584 1722 1685 4433 3817 4480 6016 5101 8674 2180 2737 9668 4275 7149 1326 9800 4087 1250 2661 4749 0951 7078 6570 0339 9965 4623 0540 0957 6548 0165 1284 4902 1116 0441 2929 0051 0857 9995 9240 9802 0822 9474 7571 0623 7772 5932 1672 7224 4630 0980 9029 2147 5787 7331 0174 1858 9750 2773 6215 9885 3344 8006 3253 0604 6003 4384 2821 4673 2124 8686 8459 9423 0273 6968 4661 7844 5424 0255 2547 4792 4467 5374 9401 1206 7899 9391 0968 8939 7038 7423 6856 6727 7350 0374 2514 6856 8184 5213 8164 4258 9305 5193 7144 0916 7515 0860 7244 0174 0141 8084 6042 5181 0634 1126 9636 4500 2538 5977 1624 4032 2865 6823 7439 8977 1815 6120 7864 1853 6229 5787 0600 4382 5331 3365 1603 5651 6751 3330 6865 9910 0308 0063 4913 2569 2019 3094 4494 1725 4427 0941 9613 7271 6810 4820 9910 8363 3502 9174 0854 8677 7734 7568 0720 4455 8979 7970 5022 1557 6524 0814 1278 0843 8718 2630 3908 1154 0097 0824 7859 4337 5885 7050 0393 6496 8972 4092 1182 0078 5656 0493 9605 3816 6074 4358 4332 6214 8690 7009 6825 5029 8782 7415 1540 5975 1381 2588 1923 3731 2939 7576 0031 7781 9263 9960 8730 9804 0411 6651 5594 1936 1198 8630 9016 9448 2244 9085 7228 5217 1963 4461 0055 0330 2220 6229 6277 4812 8053 7639 0871 8791 4674 2159 7828 9380 7599 1756 9444 0479 9931 8911 5307 7384 0105 3550 1120 6437 2188 8508 6491 4124 7819 0968 8775 3958 5616 3999 7092 4953 6260 8121 3276 9665 6900 9134 2768 6308 8067 1280 5023 8310 4702 1587 4989 6780 0320 7794 3957 4394 0139 7926 4275 5192 6627 4494 8587 9184 5545 5525 6075 3829 0180 0868 1184 4304 9256 9086 0751 5701 2096 2199 9694 2375 1322 0102 3865 9208 3656 9461 5931 3256 9318 8649 6485 2471 1023 7452 6270 1542 3029 3085 6406 4691 0217 4783 4265 3842 1920 4077 0900 8805 7200 6880 2156 9270 3522 4746 6514 7127 1080 7195 0979 9595 9529 3246 9726 3472 4012 0939 3505 3249 8908 8835 0540 0900 3795 9129 0667 8220 6335 1467 9553 0848 7085 5458 0141 1058 7791 3737 2972 0536 9327 7136 3663 3385 1034 2973 6790 2383 6961 9534 6174 4388 2640 5243 9154 9655 7946 0048 8789 7953 8528 3864 1039 7464 3673 5184 1846 6819 6396 5498 5830 4678 7511 1233 6299 5444 8304 9448 5627 1167 1761 3370 4359 6385 2667 9925 6225 6901 0792 6579

Diese Zahl wurde von anderen Programmen wie `https://www.alpertron.com.ar/ECM.HTM` nachgeprüft. Weitere Zahlen findet man im Anhang

Fazit zum Fermat-Test

Der Fermat-Test läuft um einiges schneller als die Probedivision.
Jedoch sagt dieser Test nicht aus, ob es sich tatsächlich um eine Primzahl handelt.
Je mehr Versuche man macht, desto geringer wird die Fehlerwahrscheinlichkeit.
Will man diese Fehlerwahrscheinlichkeit verringern, wird die Laufzeit des Programmes, sich deutlich erhöhen.

4.2 Was ist ein endlicher Körper (Galois-Feld)?

Ein endlicher Körper oder Galois-Körper ist ein algebraisches System, welches eine endliche Menge mit definierten Addition und Multiplikation.
Der Endliche Körper hat folgende Eigenschaften:

Eigenschaften der Endlichen Körper
- Die Endliche Menge ist eine abelsche Gruppe, in der die Rechennoperation Addition berechnet werden kann.
- Endliche Menge ohne dem Nullelement ist eine abelsche Gruppe, in der die Multiplikation stattfinden kann.
- Endliche Körper sind auch zyklische Gruppen mit der Schreibweise: $\mathbb{Z}_p$.

Tabelle 4.8: Beispiele für Irreduzible Polynome

Der Unterschied zwischen x-Beliebige Zyklische Gruppen und einem Endlichen Körper ist, das der Endliche Körper durch einer Primpotenz p^n beschrieben werden kann.

Die Anzahl an Elementen in einem endlichen Körper nennt man Ordnung des Endlichen Körper. Die Ordnung des Endlichen Körper kann nur als Primpotenz dargestellt werden.

Allgemein: Die endlichen Körper $\mathbb{F}_{p^n}$ beinhalten p-Elemente und n. Diese werden durch die Ganzzahlen $0, 1, 2, ..., p-1$ repräsentiert.

Im Zusammenhang mit den Grundrechenoperationen wird es folgendermaßen definiert:

Addition in der Endlichen Körper
Wenn $a, b \in \mathbb{Z}_p$, dann gilt $a + b = r$ in $\mathbb{Z}_p$, wobei $r \in [0, p-1]$ ist.

Tabelle 4.9: Addition in der Endlichen Körper

Multiplikation in der Endlichen Körper
Wenn $a, b \in \mathbb{Z}_p$, dann gilt $a \cdot b = r$ in $\mathbb{Z}_p$, wobei $r \in [0, p-1]$ ist.

Tabelle 4.10: Multiplikation in der Endlichen Körper

Subtraktion und Division sind inverse Operationen. Diese Rechenarten sind jedoch nicht anders, wie man normal die Zahlen in $\mathbb{R}$, $\mathbb{Q}$.

Diese sind wie folgt definiert:

Additive Inverse (Subtraktion): Wenn $a \in \mathbb{Z}_p$, dann ist (-a) die additive Inverse von a in $\mathbb{Z}_p$. Dies ist auch die einzige Lösung zur Gleichung : $a + x \equiv 0$ (mod p) Multiplikative Inverse (Division): Wenn $a \in \mathbb{Z}_p, a \neq 0$, dann ist a^{-1} die multiplikative Inverse von a in $\mathbb{Z}_p$. Dies ist auch die einzige Lösung zur Gleichung : $a \cdot x \equiv 1$ (mod p)

Nehmen wir als Beispiel die Primzahl 2.

$\mathbb{Z}_2 = 0, 1$

Hier hat man lediglich nur 2 Elemente. Rechnet man beispielsweise bei Addition alle Möglichkeiten, sieht es so aus: $0 + 0;\ 0 + 1 = 1;\ 1 + 1 = 0$ Um es einfacher und schöner zu gestalten erstellen wir eine Tabelle für Addition und Multiplikation.

Tabelle 4.11: Addition und Multiplikation in $\mathbb{Z}_2 = 0, 1$

+	0	1
0	0	1
1	1	2

+	0	1
0	0	0
1	0	1

Tabelle 4.12: Addition und Multiplikation in $\mathbb{Z}_2 = 0, 1$

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

·	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Algorithmus im Programm

Das Programm verläuft folgendermaßen. Zuerst erstellt man ein Objekt mit der gewählten Primzahl. Das Objekt besitzt alle möglichen Rechenoperationen in dem Umfeld. Führt man eine Rechnung, wird überprüft ob die Zahl im jeweiligen Menge ist. Nach der Rechnung wird das Ergebnis mit modulo Prim ausgerechnet und zurückgeschickt.

Die Rechenoperationen werden von der Modulare Arithmetik bereitgestellt und im Grunde werden die Werte weitergegeben.

4.3 Primzahlpotenz

Irreduzibles Polynom

Bevor man die Verbindung zu endlichen Körper macht, muss man die irreduziblen Polynome verstehen.
Damit Polynome irreduzibel sind, müssen die Irreduziblitätskriterien erfüllt sein.

Lemma
Sei K ein Körper und $f \in K[T]$. Dann gilt
a) Jedes Polynom $f \in K[T]$ vom Grade 1 ist irreduzibel.
b) Ein Polynom $f \in K[T]$ vom Grade ≥ 2, das eine Nullstelle in K besitzt, ist reduzibel
c) $grad(f) \in 2, 3$: f irreduzibel über K $\leftrightarrow$
d) Besitzt ein Polynom

Tabelle 4.13: Beispiele für irreduzible Polynome

Kurz zusammengefasst: Irreduzible Polynome sind Polynome, welche man nicht mehr faktorisieren kann. Somit haben diese Polynome keine Nullstellen im $\mathbb{F}_p$

Beispiele für Irreduzible Polynome
$x^2 + x + 1$
$x^3 + x^2 + x + 1$
$x^4 + x^3 + x^2 + x + 1$
...

Tabelle 4.14: Beispiele für Irreduzible Polynome

4.3.1 Zusammenhang mit endlichen Körpern

Der Körper K wird folgendermaßen aufgeschrieben $K := \mathbb{F}_{p^m}$. p für Primzahl und m für den Gradienten. Der Gradient sagt aus, welches Polynom genommen wird. Beispielsweise nimmt man für den Gradienten 2 das Polynom $x^2 + x + 1$
Alle Berechnungen werden mit Polynomen ausgeführt und die X-Werte werden nicht ersetzt.
Die Elemente die in $K := \mathbb{F}_{p^m}$ repräsentieren sind:

Allgemein: $\mathbb{F}_{p^m} = \{p_{m-1} \cdot x^{m-1} + p_{m-2} \cdot x^{m-2} + ... + p_1 \cdot x + p_0; p_i \in \{0, 1, ..., p-1\}\}$
Man kann $\mathbb{F}_{p^m}$ auch so schreiben : $\mathbb{F}_{p^m}[x] = \frac{\mathbb{F}_p[x]}{Polynom}$

Wie sieht dann die Polynom-Addition und Multiplikation aus? Um es einfacher zu verstehen, nutzen wir ein Beispiel.
Wir nehmen den Körper := $\mathbb{F}_{2^2}$
Hierbei wird unser Polynom $x^2 + x + 1$ benutzt, da der Gradient = 2 ist. Wenn es ein

Polynom mit einem gleichen oder höheren Gradienten gibt, wird dieses durch Polynomdivision verkleinert und der Rest davon genommen.
Bei $x^2 + x + 1$ kann es nur 4 Möglichkeiten geben. $:= \frac{\mathbb{F}_2[x]}{x^2+x+1} = \{0, 1, x, x+1\}$

Macht man eine Tabelle für diese Elemente, kommt folgendes dabei heraus:

Tabelle 4.15: Tabellen Grundrechenoperationen

+	0	1	x	x+1
0	0	1	x	x+1
1	1	0	x+1	x
x	x	x+1	0	1
x+1	x+1	x	1	0

·	0	1	x	x+1
0	0	0	0	0
1	0	1	x	x+1
x	0	x	x+1	1
x+1	0	x+1	1	x

Algorithmus im Programm

Im Prinzip arbeitet das Programm, wie normal bei einer Primzahl. Jedoch wird es mit Polynomen arbeiten, welche in einer ArrayList<BigInteger> abgespeichert werden.
Zuerst wird der Gradient bestimmt und es wird aus einer Liste ein Beispiel-Polynom genommen. Wenn eine ArrayList übergeben wird, bestimmt die Länge den höchsten Gradienten.
Angenommen die Länge der Liste sei q, so ist der Gradient q-1. Zuerst werden die Polynome mit den Grundrechenoperationen berechnet und anschließend wird mit einem irreduziblen Polynom der Restwert bestimmt.

5 Elliptische Kurven

5.1 Darstellungen von Punkten

Um Punkte auf einer elliptischen Kurve darzustellen oder anzugeben gibt es mehrere Möglichkeiten, die unterschiedliche Vor- und Nachteile haben. Jede Darstellung wird daher auch anders miteinander verrechnet.

5.1.1 Affine Darstellung

Die affine Darstellung ist die gebräuchlichste Art und Weise einen Punkt und eine elliptische Kurve darzustellen, da es nur die x und y Koordinaten gibt.
In diesem Fall gibt es keine einheitliche Darstellung der Unendlichkeit und man muss prüfen, ob die Koordinaten auf der Kurve

$$y^2 = x^3 + Ax + B$$

liegen.

5.1.2 Projektive Darstellung

Bei der projektiven Darstellung kommt die z - Koordinate hinzu, sodass aus der zweidimensionalen eine drei-dimensionale Kurve entsteht. Die Umrechnung eines affinen Punktes in einen projektiven Punkt ist dabei einfach durch das Hinzufügen von z = 1 durchzuführen. Bei der umgekehrten Richtung muss man die x und y Koordinaten durch z teilen, woraus folgt, dass alle Punkte mit z = 0 nicht auf der Kurve liegen und unendlich sind. Zur Erinnerung, die Formel der Kurve ändert sich von $y^2 = x^3 + Ax + B$ zu

$$y^2z = x^3 + Axz^2 + Bz^3$$

Eine Punktaddition benötigt zwölf Multiplikationen und zwei Quadrierungen, während eine Punktverdopplung nur sieben Multiplikationen und fünf Quadrierungen braucht.

5.1.3 Jacobi Darstellung

Um einen noch größeren Speedup zu erreichen, kann man Jakobi Koordinaten verwenden, da diese besonders effizient beim Verdoppeln sind. Hier ist die Darstellung (x,y,z) bei der Kurvenform

$$y^2 = x^3 + Axz^4 + Bz^6$$

gewählt. Zum Konvertieren nach affin muss man folgende Rechnung $(x/z^2, y/z^3)$ anwenden. Der Punkt unendlich ist mit (1,1,0) definiert. Natürlich werden alle anderen Punkte, die nicht auf der Kurve liegen auch als unendlich behandelt.

5.1.4 k-fache Punktmultiplikation

Wenn ein Punkt mit k multipliziert wird, unterscheidet man, dass bei $k > 0$

$$P + P + ... + P = kP$$

und bei $k < 0$

$$(-P) + (-P) + ... + (-P) = kP$$

berechnet wird. Durch wiederholtes Verdoppeln erreicht man bei großen k die beste Performance und da die elliptischen Kurven auf endlichen Feldern basieren, muss man sich auch keine Gedanken um immer größere Koordinaten machen, da diese immer $mod p$ gerechnet werden.

5.1.5 Negation

Die Negation wird durch einen Vorzeichenwechsel durchgeführt bei der y-Koordinate. Da die Zahl dann negativ ist, muss noch das positive Gegenstück im endlichen Zahlenraum gefunden werden, was durch die einfache Addition der Zahl p geht. Vorausgesetzt ist hierbei eine affine Darstellung. Kommt man von einer anderen Darstellung muss man zuerst zur affinen Darstellung, dann negieren und abschließend in die ursprüngliche Darstellung umrechnen.

5.2 Punktaddition

5.2.1 affin

Die Addition zweier Punkte bedeutet, dass man eine Gerade durch die Punkte zieht und diese Gerade schneidet die Kurve in einem dritten Punkt, dieser ist das Ergebnis.
Bei zwei unterschiedlichen Punkten, wovon einer nicht im unendlichen Bereich liegt, wird die Steigung durch den Differenzquotienten berechnet:

$$m = x_1 - x_2 / y_1 - y_2$$

Nun braucht man nur noch einzusetzen

$$x = m^2 - x1 - x2$$

$$y = m(x - x_1) - y_1$$

und erhält den neuen Punkt.
Ansonsten gilt $P_1 + \infty = P_1$, wobei ∞ ein Punkt außerhalb der Kurve ist.
Diese Addition ist für den Rechner bei großen Zahlen sehr aufwendig, da man dividieren muss und das die anspruchsvollste Rechenoperation ist. Man benötigt zwei Multiplikationen, eine Quadrierung und eine Division.

5.2.2 projektiv

Bei $P1 \neq P2$ sieht die Addition wie folgt aus:

$$u = y_2 z_1 - y_1 z_2$$

$$v = x_2 z_1 - x_1 z_2$$

$$w = u^2 z_1 z_2 - v^3 - 2v^2 x_1 z_2$$
$$x_3 = 2uw$$
$$y_3 = t(4v - w) - 8y_1^2 u^2$$
$$z_3 = 8u^3$$

5.2.3 jakobi

Bei zwei unterschiedlichen Punkten wird wie folgt addiert:

$$r = x1z_2^2$$
$$s = x_2 z_1^2$$
$$t = y1z_2^3$$
$$v = s - r$$
$$w = u - t$$
$$x_3 = -v^3 - 2rv^2 + 2^2$$
$$y_3 = -tv^3 + (rv^2 - x_3) * w_1$$
$$z_3 = vz_1 z_2$$

Dadurch ergeben sich zwölf Multiplikationen und vier Quadrierungen.
Ein weiterer Vorteil dieser Darstellung ist die Kompatibilität zu affinen Punkten. Hier benötigt man für die Addition von einem Jacobi Punkt und einem affinen Punkt nur acht Multiplikationen und drei Quadrierungen.

5.3 Punktverdopplung

5.3.1 affin

Wenn man den gleichen Punkt mit sich selbst addiert, dann muss wie immer bei der Addition die Steigung berechnet werden.

$$m = 3x^2 + A/2y$$

woraus sich wiederum die x Koordinate

$$m^2 - 2x$$

und y Koordinate berechnet wird

$$y = m(x - x_0) + y$$

5.3.2 projektiv

Wenn man zwei gleiche Punkte addiert, ändert sich die gerade genannte Formel zu:

$$t = Az_1^2 + 3x_1^2$$

$$u = y_1 z_1$$

$$v = ux_1 y_1$$

$$w = t^2 - 8v$$

$$x_3 = 2uw$$

$$y_3 = t(4v - w) - 8y_1^2 u^2$$

$$z_3 = 8u^3$$

mit dem Sonderfall von $P1 = -P2$ folgt $P1 + P2 = \infty$.

5.3.3 jakobi

Wenn man nun aber gleiche Punkte addiert, erkennt man direkt im Vergleich die Einfachheit der Formel:

$$v = 4x_1 y_1^2$$

$$w = 3x_1^2 + Az_1^4$$

$$x_3 = -2v + w^2$$

$$y_3 = -8y_1^4 + (v - x_3)w_1$$

$$z_3 = 2y_1 z_1$$

. Wodurch man die Operationen auf drei Multiplikationen und sechs Quadrierungen reduziert. Das ist besonders schnell berechenbar für einen Computer.

6 ECDH – Elliptic Curve Diffie Hellman

6.1 Diffie Hellman Schlüsselaustausch

Das Diffie-Hellman-Protokoll wird benutzt um auf einem unsicheren Kanal einen gemeinsamen geheimen Schlüssel zu vereinbaren um damit eine verschlüsselte sichere Kommunikation zu betreiben.
Dafür einigen sich beide Teilnehmer über den öffentlichen Kanal auf eine ausreichend große Primzahl p und eine natürliche Zahl g, die zwischen 1 und $p-1$ liegt und ein Erzeuger von Z_P ist.
Nun muss jeder Teilnehmer eine geheime Zufallszahl generieren, die kleiner p ist. Danach berechnet jeder seinen öffentlichen Schlüssel

$$A = g^a mod p$$

und schickt diesen an den anderen Teilnehmer.
Wenn jeder mit seinem privaten Schlüssel und dem öffentlichen Schlüssel des Gegenübers den gemeinsamen Schlüssel

$$K_1 = B^a mod p$$

$$K_2 = A^b mod p$$

$$K_1 = K_2$$

berechnen.
Um die Kommunikation abzuhören müsste ein Angreifer das diskrete Logarithmusproblem lösen, was bei großen Zahlen nicht effizient möglich ist.

6.2 Schlüsselaustausch bei Elliptischen Kurven

Wie beim klassischen Diffie Hellman Schlüssel Austausch wird bei der Variante mit elliptischer Kurve ein Schlüsselaustausch vorangetrieben. Dabei braucht jeder Teilnehmer einen privaten Schlüssel d, eine zufällige Zahl die zwischen 1 und $n-1$ gewählt wird und einen öffentlichen Schlüssel Punkt Q. Q entsteht durch die folgende Formel

$$Q = d * G$$

mit G als Erzeuger der Kurve.
Mit dem eigenen d und dem G vom Gegenüber kann man den Punkt

$$K = d_a * Q_b$$

$$K = d_b * Q_a$$

berechnen. Von K benötigt man in den meisten Verfahren nur die x-Koordinate.
Sollte ein Teilnehmer einen ungültigen Punkt, der nicht auf der Kurve liegt, wählen und der Andere validiert diesen Punkt nicht, ist es möglich den geheimen Schlüssel des echten Punktes herauszufinden.

7 Fazit

Literatur

[1] .

[2] *Advanced Encryption Standard*.

[3] Menezes Alfred, Oorschot Paul und Vanstone Scott. *Handbuch of Applied Cryptography*. CRC-Press, 1997.

[4] Nada Ali und Esaa Kawther. „Security Improvement in Elliptic Curve Cryptography“. In: *International Journal of Advanced Computer Science and Applications* 9 (Jan. 2018), S. 122–133.

[5] Werner Annette. *Elliptische Kurven in der Kryptographie*. Hrsg. von Springer. Springer-Verlag Berlin Heidelberg GmbH, 2002.

[6] Khan Arif. *Comparative Analysis of Elliptic Curve Cryptography*. Jan. 2017. ISBN: 978-3-330-01788-7.

[7] Hankerson Darrel, López Julio und Menezes Alfred. „Software Implementation of Elliptic Curve Cryptography over Binary Fields“. In: Jan. 2000, S. 1–24. ISBN: 978-3-540-41455-1. DOI: 10.1007/3-540-44499-8_1.

[8] Hankerson Darrel, Vanstone Scott und Menezes Alfred. „Guide to Elliptic Curve Cryptography“. In: *Springer Professional Computing*. 2004.

[9] Michel Douguet und Vincent Dupaquis. „Modulare Reduktion unter Verwendung einer speziellen Form des Modulo“. Deutsch. Pat. DE112009000152T5. 2008. URL: https://patents.google.com/patent/DE112009000152T5/de.

[10] Kossi D. Edoh. „Elliptic curve cryptography: Java implementation“. In: *Information Security Curriculum Development Conference, InfoSecCD 2004*. 2004, S. 88–93.

[11] *Elliptic Curve Crytography*. Technical Guideline. Bonn, Germany: Federal Office for Information Security, Juni 2018.

[12] Samta Gajbhiye, Monisha Sharma und Samir Dashputre. „A Survey Report On Elliptic Curve Cryptography“. In: *International Journal of Electrical and Computer Engineering (IJECE)* 1 (Okt. 2011). DOI: 10.11591/ijece.v1i2.86.

[13] Rahman Hazifur, Azad Saiful und Khan Pathan Al-Sakib. *Practical Cryptography Algorithms and Implementations using C++*. CRC Press, 2015. ISBN: 13: 978-1-4822-2890-8.

[14] *Information Technology Security Techniques- Digital Signatures., with Appendix- part 3: Certificatebased mechanism*.

[15] Steffen Daniel Jensen, Brian Melin Iversen, Rasmus Feldthaus und Markus Neubrand. *Cryptography: Fast Modular Arithmetic*. Aarhus University, 2009.

[16] Merkle Johannes und Lochter Manfred. „Ein neuer Standard für Elliptische Kurven“. In: Mai 2009.

[17] Sheetal Kalra und Sandeep Sood. „Elliptic curve cryptography: Survey and its security applications“. In: *Proceedings of the International Conference on Advances in Computing and Artificial Intelligence, ACAI 2011* (Jan. 2011), S. 102 –106. DOI: 10.1145/2007052.2007073.

[18] Hans Werner Lang. *Modulare Exponentiation*. Hochschule Flensburg. 2014. URL: https://www.inf.hs-flensburg.de/lang/krypto/algo/modexp.htm.

[19] Hans Werner Lang. *Montgomery Multiplikation*. Hochschule Flensburg. 2014. URL: https://www.inf.hs-flensburg.de/lang/krypto/algo/modexp-iterativ.htm.

[20] Hans Werner Lang. *Multiplikativ Inverses Element*. Hochschule Flensburg. 2014. URL: https://www.inf.hs-flensburg.de/lang/krypto/grund/inverses-element.htm.

[21] Anoop MS. „Elliptic curve cryptography-an implementation guide“. In: (2007).

[22] Mihailescu Marius und Nita Stefania. „Elliptic-Curve Cryptography“. In: Jan. 2021, S. 189–223. ISBN: 978-1-4842-6585-7. DOI: 10.1007/978-1-4842-6586-4_9.

[23] Mohamad Afendee Mohamed. „A survey on elliptic curve cryptography“. In: *Applied mathematical sciences* 8 (2014), S. 7665–7691.

[24] Abdalbasit Mohammed und Nurhayat Varol. „A Review Paper on Cryptography“. In: Juni 2019, S. 1–6. DOI: 10.1109/ISDFS.2019.8757514.

[25] Nayak und Rajput. „Cryptography Algorithms – The Science of Information Security: Review Paper“. In: 2017.

[26] Koblitz Neal. „Elliptic Curve Cryptosystems“. In: Bd. 48. 177. 1987. Kap. Mathematics of Computation, S. 203–209.

[27] Barett Paul. „Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor“. In: 1986. Kap. Advances in Cryptology — CRYPTO' 86, S. 311 –323. ISBN: ISBN 978-3-540-18047-0. DOI: doi:10.1007/3-540-47721-7_24.

[28] *Public Key Cryptography for the financial Services Industry: The Elliptic curve Digital SignatureAlgorithm (ECDSA)*. ANSI X9.62, 1999.

[29] Hossain Rownak und Hossain Selim. „Efficient FPGA Implementation of Modular Arithmetic for Elliptic Curve Cryptography“. In: *International Conference on Electrical, Computer and Communication Engineering (ECCE)* (2019), S. 1–6.

[30] Markan Ruchika und Kaur. „Literature Survey on Elliptic Curve Encryption Techniques“. In: 2013.

[31] *STANDARDS FOR EFFICIENT CRYPTOGRAPHY SEC 2: Recommended Elliptic Curve Domain Parameters*. Jan. 2010. URL: https://www.secg.org/sec2-v2.pdf.

[32] Sushanta Sahu und Manoranjan Pradhan. „Implementation of Modular Multiplication for RSA Algorithm“. In: Juli 2011, S. 112 –114. DOI: 10.1109/CSNT.2011.30.

[33] Contini Scott, Çetin Kaya Koç und Colin Walter. „Modular Arithmetic“. In: *Encyclopedia of Cryptography and Security*. Hrsg. von Henk C. A. van Tilborg und Sushil Jajodia. Boston, MA: Springer US, 2011, S. 795–798. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_49. URL: https://doi.org/10.1007/978-1-4419-5906-5_49.

[34] Ankita Soni und Nisheeth Saxena. „Elliptic Curve Cryptography: An Efficient Approach for Encryption and Decryption of a Data Sequence“. In: *International Journal of Science and Research (IJSR)* 2 (2013), S. 203–208.

[35] *Standard Specifications for Public Key Cryptography*.

[36] Stolbikova Veronika. „Can Elliptic Curve Cryptography Be Trusted? A Brief Analysis of the Security of a Popular Cryptosystem“. In: *ISACA Journal* 3 (Mai 2016), S. 1–5.

[37] Miller Victor. „Use of elliptic curves in cryptography“. In: LNCS 218, Springer Verlag, 1986. Kap. Advancesin Cryptography-Crypto ’85, S. 417 –426.

[38] Hasenplaugh William, Gaubatz Gunnar und Gopal Vinodh. „Fast Modular Reduction“. In: 2007. DOI: doi:10.1109/ARITH.2007.18..

[39] Dietmar Wätjen. *Kryptographie Grundlagen, Algorithmen, Protokolle*. Bd. 3. Springer-Verlag, 2018. DOI: https://doi.org/10.1007/978-3-658-22474-5.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis