Fakultät für Elektrotechnik, Informatik und Mathematik
Institut für Informatik

Diplomarbeit

Smart Card basierte Berechnung einer Gruppensignatur als Teil einer biometrischen Authentisierung

von

Tim Postler

30. März 2010

Betreuer:
Prof. Dr. rer. nat. Johannes Blömer
Prof. Dr. rer. nat. Franz Josef Rammig

Erklärung

Ich versichere, dass ich die vorliegende Diplomarbeit selbständig und ohne unerlaubte Hilfe Dritter angefertigt habe. Alle Stellen, die inhaltlich oder wörtlich aus Veröffentlichungen stammen, sind kenntlich gemacht. Diese Arbeit lag in gleicher oder ähnlicher Weise noch keiner Prüfungsbehörde vor und wurde bisher noch nicht veröffentlicht.

Paderborn, den 30. März 2010

Tim Postler

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei dieser Arbeit unterstützt und mir so die Fertigstellung ermöglicht haben.
Als erstes möchte ich mich bei Herrn Prof. Dr. Johannes Blömer und Jonas Schrieb für die intensive Betreuung der Arbeit bedanken.
Ein weiterer Dank geht an Gerhard Schmidt, Carsten Rust und Thomas Hübner für die Betreuung innerhalb der Firma Sagem Orga.
Nicht zuletzt möchte ich mich bei meinen Eltern und meiner ganzen Familie bedanken, die mir die ganze Zeit zur Seite gestanden haben.

Inhaltsverzeichnis

1. Einführung

Die Biometrie hält immer mehr Einzug in unseren Alltag, sei es durch die Speicherung von Fingerabdrücken und Gesichtserkennungsmerkmalen im Reisepass oder als Zugangsmittel zu geschützten Computersystemen oder gesicherten Bereichen in Gebäuden. Mit dem vermehrten Einsatz entstehen Probleme im Bereich des Datenschutzes und dem Schutz der Privatsphäre. Gängige biometrische Systeme speichern Referenzabnahmen des biometrischen Merkmals in einer Datenbank oder auf einer Chipkarte ab, um beim späteren Verifizieren frische Abnahmen des biometrischen Merkmals gegen diese Referenzdaten zu vergleichen.

In je mehr Datenbanken solche Daten gespeichert werden, desto mehr erhöht sich die Gefahr, dass die Daten in falsche Hände geraten und sich Betrüger mit den biometrischen Daten als andere Personen ausgeben können. Zusätzlich könnten durch Abgleich mehrerer Datenbanken Persönlichkeitsprofile anhand der biometrischen Identität erstellt werden. Ist ein biometrisches Merkmal einmal kompromittiert, kann man es auch nicht einfach ändern wie eine PIN oder ein Passwort.

Unter anderem diese Probleme werden im Projekt TURBINE adressiert. Dort werden Lösungen gesucht, um aus einem Fingerabdruck sogenannte Pseudo-Identitäten zu erzeugen. Der Benutzer kann durch Überprüfung des Fingerabdrucks immer wieder beweisen, dass er der rechtmäßige Eigentümer einer Pseudo-Identität ist. Umgekehrt soll es nicht möglich sein, aus der Pseudo-Identität Informationen über den Fingerabdruck zu erhalten. Die Kompromittierung einer Pseudo-Identität führt damit nicht automatisch zur Kompromittierung des Fingerabdrucks. Kann zusätzlich aus dem einen Fingerabdruck für jede Anwendung eine eigene Pseudo-Identität erzeugt werden, so ist auch kein Abgleich zwischen verschiedenen Anwendungen möglich und die Privatsphäre bleibt gewahrt.

In der vorliegenden Arbeit sollen einige Aspekte des TURBINE-Projektes vorgestellt werden. Hauptaugenmerk liegt dabei auf einem Verfahren von Bringer et al. [BCPZ08] zur biometrischen Authentisierung, bei dem die biometrischen Daten mit einem Gruppensignaturschema kombiniert werden, mit dem anschließend auf eine anonyme Art und Weise Signaturen erstellt werden können.

Dieses leitet über zum Hauptteil der Diplomarbeit, einer prototypischen Implementierung des Gruppensignaturschemas auf einer Smart Card. Bringer et al. benutzen die Smart Card in ihrer Arbeit in erster Linie als sicheren Speicher für den privaten Schlüssel des Gruppensignaturschemas, die eigentlichen Berechnungen werden vom Terminal ausgeführt. Nur in einer Randnotiz wird erwähnt, dass die Smart Card eventuell mehr leisten kann. Dies soll im Rahmen der Diplomarbeit näher untersucht werden.

Es wird eine vollständige Berechnung der Gruppensignatur auf einer Smart Card realisiert. Dies erhöht die Sicherheit, da der private Schlüssel niemals die sichere Umgebung der Smart Card verlässt. Untersucht werden soll in erster Linie die prinzipielle Machbarkeit der Berechnung der Gruppensignatur auf der Chipkarte. Von besonderem Interesse ist weiterhin die benötigte Laufzeit einer Berechnung. Eine Optimierung der Hardware

ist nicht Gegenstand dieser Diplomarbeit, die Hardware der Smart Card wird als gegeben angesehen.

Neben einer ausführlichen Beschreibung der Gruppensignatur werden in dieser Arbeit auch alle Komponenten detailliert erläutert, die für die konkrete Berechnung der Gruppensignatur notwendig sind. Eine der größten Herausforderungen ist die Konstruktion und effiziente Berechnung einer speziellen bilinearen Abbildung. Neben den mathematischen Zusammenhängen werden für die wichtigsten Komponenten auch Algorithmen für ihre Berechnung inklusive diverser Optimierungen vorgestellt.

Smart Cards

Chipkarten unterteilt man allgemein in die Kategorien Speicherkarten und Smart Cards (auch Mikroprozessorkarten genannt) [RE99]. Wie der Name schon impliziert dienen Speicherkarten dazu, Daten zu speichern und zu lesen. Die bekanntesten Vertreter sind vermutlich die Telefonkarte und die bisherige Krankenversicherungskarte. Sie beinhalten ein geringes Maß an Sicherheitslogik, um die Daten vor Manipulation zu schützen.

Smart Cards hingegen sind deutlich leistungsfähiger, sie sind im Prinzip kleine Computer. Sie besitzen einen Hauptprozessor, RAM, ROM und auch einen nichtflüchtigen Speicher, entweder EEPROM oder in letzter Zeit immer häufiger Flash-EEPROM, und das alles auf kleinstem Raum. Der eigentliche Halbleiter hat nur etwa die Größe eines Stecknadelkopfes und ist damit nochmals deutlich kleiner als die goldfarbene Kontaktfläche, die von vielen fälschlicherweise als „Chip“ angesehen wird.

Mit die größten Vorteile von Smart Cards sind die sichere Speicherung geheimer Daten und die Möglichkeit zur Ausführung einiger kryptografischer Algorithmen. Der reguläre Zugriff auf eine Smart Card erfolgt über die I/O-Schnittstelle, über die spezielle Kommandos gesendet und die zugehörigen Antworten empfangen werden. Zum Lesen und Schreiben der Datenfelder auf der Karte existieren festgelegte Kommandos. Das Kartenbetriebssystem stellt sicher, dass nur dann auf Datenfelder zugegriffen werden darf, wenn der Anwender auch die nötige Berechtigung besitzt. So kann ein Zugriff zum Beispiel eine PIN-Eingabe erfordern oder eine andere Art der Authentisierung. Der Zugriff von außen auf ein Datenfeld kann auch komplett verboten werden. So ist es möglich, initial geheime Schlüssel in der Karte abzuspeichern, auf die später von außen nicht mehr zugegriffen werden kann, die aber intern in der Chipkarte zur Anwendung in kryptografischen Algorithmen genutzt werden können.

Neben dieser Absicherung auf logischer Ebene schützt sich eine Smart Card auch auf Hardware-Ebene gegen Manipulationen von außen. So enthalten Hochsicherheitschips eine Reihe von Sensoren, um Strom-, Spannungs- oder Temperaturschwankungen zu erkennen und darauf reagieren zu können. Das RAM und der nichtflüchtige Speicher können verschlüsselt werden. Zusätzlich enthalten kryptografische Algorithmen Schutzmaßnahmen gegen Seitenkanalangriffe, hier insbesondere gegen SPA- und DPA-Attacken (Simple/Differential Power Analysis).

Die Vorteile führten letztlich auch zur großen Verbreitung von Smart Cards. So begegnen sie einem heute zum Beispiel als SIM-Karte im Handy, als EC-Karte oder in naher Zukunft als elektronische Gesundheitskarte (siehe Abbildung 1.1). Außerdem werden sie häufig als Mittel zur Zugangskontrolle benutzt. Und die Entwicklung schreitet voran, ständig entstehen neue Anwendungsszenarien, in denen eine Smart Card zum Einsatz

Abbildung 1.1.: Elektronische Gesundheitskarte (Muster)

kommen kann, nicht zuletzt weil sie Jahr für Jahr leistungsfähiger und vielseitiger wird. Ein solches Szenario ist die Berechnung einer Gruppensignatur auf der Smart Card, welches in dieser Diplomarbeit untersucht wird.

Aufbau der Arbeit

Kapitel 2 gibt einen kurzen Überblick über das Projekt TURBINE. In Kapitel 3 werden einige grundlegende kryptografische Verfahren vorgestellt mit dem Schwerpunkt auf der Gruppensignatur von Boneh und Shacham. Weiterhin wird das Verfahren von Bringer et al. zur biometrischen Authentisierung unter Verwendung der Gruppensignatur vorgestellt.

In Kapitel 4 werden Schritt für Schritt alle Komponenten beschrieben, die zur konkreten Berechnung der Gruppensignatur benötigt werden. Ein Schwerpunkt dabei ist die Berechnung einer geeigneten bilinearen Abbildung.

Kapitel 5 enthält Erläuterungen zur konkreten Realisierung der Gruppensignatur auf der ausgewählten Smart Card. Neben den verwendeten Algorithmen und einigen Optimierungen enthält es auch Untersuchungen zur Laufzeit der Berechnung der Gruppensignatur.

Abschließend wird eine Zusammenfassung der Ergebnisse dieser Arbeit gegeben.

2. Projekt TURBINE

Das Projekt TURBINE (TrUsted Revocable Biometric IdeNtitiEs) untersucht neue Möglichkeiten im Rahmen der elektronischen Authentisierung unter besonderer Berücksichtigung der Privatsphäre der Benutzer. Die Erkennung von Benutzern erfolgt anhand von biometrischen Merkmalen, wobei der Fokus auf der Nutzung von Fingerabdrücken liegt. Als neuartiger Ansatz wird versucht, die dabei anfallenden biometrischen Daten durch fortschrittliche kryptografische Verfahren gegen Missbrauch zu schützen.

Um dieses zu erreichen, sollen Wege gefunden werden, um aus Fingerabdruckdaten sogenannte Pseudo-Identitäten zu generieren. Durch eine Überprüfung des Fingerabdrucks soll ein Anwender jederzeit bestätigen können, dass er der rechtmäßige Eigentümer einer Pseudo-Identität ist. Umgekehrt soll die Pseudo-Identität allerdings keinerlei Informationen über die Fingerabdruckdaten verraten, wodurch ein Missbrauch der Fingerabdruckdaten ausgeschlossen werden soll.

Die Pseudo-Identität wird zusammen mit eventuell nötigen Zusatzdaten der kryptografischen Verfahren in einem sogenannten gesicherten biometrischen Template (*Protected Biometric Template*) gespeichert. Das ist ein entscheidender Unterschied zu bisherigen biometrischen Verfahren, bei denen direkt die Fingerabdruckdaten in einem sogenannten biometrischen Referenztemplate gespeichert werden, um so einen Vergleichsgegenstand für zukünftige Überprüfungen eines Fingerabdrucks bereitzustellen.

Eine weitere wesentliche Anforderung besteht darin, aus einem Fingerabdruck unterschiedliche Pseudo-Identitäten generieren zu können. Dadurch kann ein Benutzer für unterschiedliche Anwendungen unterschiedliche Pseudo-Identitäten benutzen (beziehungsweise sogar mehrere Pseudo-Identitäten für ein und dieselbe Anwendung), die aber alle mit demselben Fingerabdruck überprüft werden können. Würde ein Benutzer für unterschiedliche Anwendungen mit bisherigen biometrischen Verfahren jeweils dasselbe Referenztemplate zur Authentisierung benutzen, so könnten zwischen den Datenbeständen der Anwendungen Quervergleiche gemacht werden, um so umfangreiche Benutzerprofile erstellen zu können. Da die Datensammelwut von Behörden und Unternehmen stetig ansteigt, wie zum Beispiel die aktuellen Beispiele der Vorratsdatenspeicherung oder von ELENA[1] zeigen, sind solche umfangreichen Benutzerprofile wirklich eine ernst zu nehmende Gefahr für die Privatsphäre eines Benutzers. Durch Nutzung von unterschiedlichen Pseudo-Identitäten für unterschiedliche Anwendungen kann ein Benutzer daher solche Quervergleiche wirksam verhindern.

Eine weitere Anforderung ist die Möglichkeit zum Widerruf (Revocation) von Pseudo-Identitäten beziehungsweise den gesicherten biometrischen Templates. Wird eine Pseudo-Identität kompromittiert, so kann durch den Widerruf eine weitere Verwendung verhindert werden. Da aus der Pseudo-Identität keine Rückschlüsse auf die zugehörigen Fingerabdruckdaten möglich sind, sind diese nach wie vor unversehrt. Aus diesem Grund können auch andere Pseudo-Identitäten, die aus demselben Fingerabdruck erzeugt wur-

[1] http://www.das-elena-verfahren.de/

den, weiterhin genutzt werden.

Organisatorisches

Das TURBINE-Projekt wird im Rahmen des „Seventh Framework Programme (FP7)“ von der EU gefördert. Die Mitglieder des Projektes stammen aus Industrie und Forschung, Beteiligte sind unter anderem die Firmen Sagem Orga und Sagem Sécurité. Mögliche Anwendungsgebiete für die Ergebnisse des Projektes finden sich in den Bereichen eGovernment, eHealth, eID, eBanking und beim physikalischen Zugangsschutz.

Neben der Erforschung der neuen Technologien für das Identitätsmanagement auf Basis der gesicherten biometrischen Templates ist auch vorgesehen, dass diese im Rahmen zweier konkreter Anwendungsfälle ihre Praxistauglichkeit demonstrieren müssen.

Der erste Anwendungsfall stammt aus dem Gesundheitsbereich, hier werden die Arbeitsabläufe eines Apothekers nachgestellt. Mittels Fingerabdruck und auf einer Smart Card gespeicherter Templates soll er zum einen die Annahme von elektronischen Rezepten bestätigen, zum anderen soll er sich mit den unterschiedlichen Pseudo-Identitäten zum Beispiel in ein Apotheker-Forum einloggen oder auf speziellen Web-Portalen von Pharmazie-Unternehmen Zugang erhalten.

Der zweite Anwendungsfall zeigt die Möglichkeiten bei der Zugangskontrolle. Auf dem Flughafen von Thessaloniki (TIA) existieren für das Bodenpersonal unterschiedliche Zugangsberechtigungen für die verschiedenen Bereiche des Flughafens. Die Berechtigungen werden anhand unterschiedlicher Pseudo-Identitäten abgebildet, die auf einer Smart Card gespeichert sind. Zusammen mit dem zugehörigen Fingerabdruck kann so Zugang zu den verschiedenen Bereichen erlangt werden.

2.1. Identitätsmanagement

Beim Identitätsmanagement unterscheidet man drei verschiedene Parteien:

1. Der Benutzer, der einen Dienst nutzen möchte und bereit ist, seine Identität mit einem Fingerabdruck zu bestätigen.

2. Der Dienstanbieter (Service-Provider), der den Benutzern einen Dienst bereitstellt. Insbesondere wenn der Dienst kostenpflichtig ist, möchte der Dienstanbieter sicherstellen, dass nur Benutzer mit der entsprechenden Berechtigung Zugang erhalten. Dazu kann er auf eine externe Partei zurückgreifen, die ihm Dienste zur Verifizierung der Identität des Benutzers bereitstellt.

3. Der Identity-Provider, der dem Dienstanbieter Möglichkeiten bereitstellt, um die physikalische Identität eines Benutzers anhand einer digitalen Identität zu verifizieren.

Die wesentlichen Aktionen beim Identitätsmanagement sind das sogenannte Enrolment, die Verifikation einer (Pseudo-)Identität sowie der Widerruf. Im Folgenden wird auf abstrakter Ebene beschrieben, was dabei geschehen soll [TUR08].

Enrolment

Das Enrolment ist sozusagen die initiale Registrierung des Benutzers beim Identity-Provider. Der Benutzer muss zum Beispiel anhand eines Ausweises den Identity-Provider von seiner physikalischen Identität überzeugen. Anschließend wird der Fingerabdruck des Benutzers an einem biometrischen Sensor abgenommen und aus diesen Daten eine Pseudo-Identität abgeleitet (eventuell auch gleich mehrere Identitäten). Diese wird als gesichertes biometrisches Template auf einer Smart Card gespeichert, die dem Benutzer ausgehändigt wird. Zusätzlich wird das Template eventuell auch noch in einer Datenbank des Identity-Providers abgespeichert. Der Benutzer kann von jetzt an die Pseudo-Identität mit Hilfe des Fingerabdrucks bestätigen. Der Identity-Provider steht gegenüber dem Dienstanbieter dafür ein, dass hinter der Pseudo-Identität eine echte physikalische Person steht, die sich ordnungsgemäß ausgewiesen hat, wobei die physikalische Identität dem Dienstanbieter im Normalfall allerdings nicht bekannt gemacht wird.

Verifikation

Die Verifikation der Pseudo-Identität durch den Dienstanbieter ist der Hauptanwendungsfall. Dazu erzeugt der Benutzer an einem Sensor eine frische Abnahme seines Fingerabdrucks. Nur mit dem passenden Fingerabdruck kann dann in einem Authentisierungsverfahren zwischen Dienstanbieter und Benutzer die Pseudo-Identität verifiziert werden. Ist die Authentisierung erfolgreich, kann der Dienstanbieter dem Benutzer Zugriff auf den angebotenen Dienst gewähren.

Widerruf

Der Widerruf läuft ähnlich ab wie das Enrolment. Der Benutzer muss sich wiederum gegenüber dem Identity-Provider ausweisen, woraufhin dieser anschließend die Pseudo-Identität widerruft. Lag eine Kompromittierung der Pseudo-Identität vor, so kann auch gleich wieder ein neues Enrolment durchgeführt werden, damit der Benutzer wieder eine gültige Pseudo-Identität im System besitzt. Anschließend werden Widerruf und neue Pseudo-Identität vom Identity-Provider an den Dienstanbieter kommuniziert.

2.2. Konkrete Verfahren

Im TURBINE-Projekt sollen konkrete Verfahren zur Erzeugung von Pseudo-Identitäten aus einem Fingerabdruck und zur späteren Verifikation der Pseudo-Identität erforscht werden. Die Pseudo-Identität soll letztlich ein 0-1-Bitstring sein, der wie zufällig aussieht.

Ein generelles Problem bei der Anwendung von biometrischen Verfahren ist, dass die mehrmalige Abnahme eines biometrischen Merkmals – hier also eines Fingerabdrucks – niemals die exakt identische digitale Darstellung erzeugt. Daher arbeitet man in der Biometrie mit sogenannten Matching-Verfahren, die letztlich anhand von Ähnlichkeiten entscheiden müssen, ob zwei unterschiedliche digitale Darstellungen von ein und demselben Fingerabdruck stammen. Die Güte dieser Verfahren wird häufig anhand der Falschakzeptanzrate (False-Acceptance-Rate, FAR) und der Falschrückweisungsrate (False-Rejection-Rate, FRR) gemessen [TUR08][Kap. 10].

Die FAR ist ein Maß dafür, wie häufig ein „falscher“ Fingerabdruck für einen „richtigen“ gehalten wird, letztlich also ein eigentlich unberechtigter Benutzer trotzdem für einen

Berechtigten gehalten wird. Daher ist sie ein wichtiges Sicherheitsmerkmal und sollte möglichst klein sein. Umgekehrt ist die FRR ein Maß dafür, wie häufig ein eigentlich berechtigter Benutzer fälschlicherweise abgewiesen wird. Dies ist ein Komfortmerkmal, allerdings trotzdem nicht zu unterschätzen. Ein System, das seine Benutzer ständig irrtümlich aussperrt, wird keine Akzeptanz finden und ist damit nicht praktikabel.

Unglücklicherweise bedingen sich FAR und FRR gegenseitig. Eine Senkung der FAR erzeugt meist eine höhere FRR, umgekehrt gilt das gleiche. Für biometrische Systeme liegt die FAR häufig im Bereich von 1:10000 oder 1:100000. Je nach Anwendungsfall kann die Anforderung an die FAR höher oder niedriger ausfallen.

Auch die Kombination von biometrischen und kryptografischen Verfahren gestaltet sich schwierig. Da mehrmalige Abnahmen eines Merkmals nie identisch sind, kann man sie zum Beispiel nicht einfach als einen geheimen Schlüssel für ein kryptografisches Verfahren benutzen, da er nicht exakt reproduzierbar wäre.

Ein konkretes Verfahren zur Umsetzung der Pseudo-Identitäten, welches genau das eben genannte Problem adressiert, basiert auf den Eigenschaften eines Secure-Sketch und eines Fuzzy-Extractor und wird im TURBINE-Projekt als eines von mehreren möglichen Verfahren untersucht. Die Grundlagen des Verfahrens werden in Kapitel 3.6 etwas näher dargestellt.

2.2.1. Anonymität

Erste Analysen im Projekt haben ergeben, dass man sich in einigen Anwendungsfällen noch mehr Anonymität wünschen würde, als durch die Pseudo-Identitäten gewährleistet werden kann. So wurde zum Beispiel der Anwendungsfall eines Apothekerforums betrachtet. Selbst wenn die Pseudo-Identität nicht die wahre Identität des Apothekers verrät, so wird doch zumindest vermutet, dass sich aus der Summe der verfassten Beiträgen eines Benutzers ermitteln lässt, um wen es sich dabei handeln könnte, zum Beispiel weil jemand immer wieder gezielt nach bestimmten Medikamenten oder Krankheiten fragt. Insgesamt wünscht man sich mehr Anonymität, sodass die Zuordnung von Beiträgen zu einer bestimmten Identität nicht mehr möglich ist.

Nach wie vor soll aber gewährleistet sein, dass nur Benutzer mit der entsprechenden Zugangsberechtigung am Forum teilnehmen dürfen. Auf den ersten Blick klingt es erst einmal paradox, jemandem Zugang zu gewähren, ohne zu erfahren, wer er ist, aber für die Problemstellung gibt es tatsächlich Lösungen. Daher wird im TURBINE-Projekt ein Verfahren zur biometrischen Authentisierung mit einer Gruppensignatur untersucht, dass tatsächlich die Anonymität zwischen Benutzer und Dienstanbieter gewährleistet. Das Verfahren wird ausführlich in Kapitel 3.5 vorgestellt.

3. Grundlegende Verfahren

In diesem Kapitel werden einige mathematischen Definitionen angegeben, die im weiteren Verlauf der Arbeit noch benötigt werden. Zusätzlich werden einige Verfahren vorgestellt, mit besonderem Fokus auf der Gruppensignatur und ihrer Anwendung in Form einer biometrischen Authentisierung.

3.1. Zyklische Gruppen

In den folgenden Abschnitten und speziell in Kapitel 4 wird das Konzept einer mathematischen Gruppe benötigt [MOV96, S. 75f].

Definition 3.1.1. *Eine Gruppe $(G, *)$ besteht aus einer Menge G und einer Operation $* : G \times G \to G, (a, b) \mapsto a * b$, die folgende Axiome erfüllt:*

1. *Assoziativität: Für alle Gruppenelemente a, b, c gilt: $(a * b) * c = a * (b * c)$.*
2. *Neutrales Element: Es gibt ein neutrales Element $e \in G$, sodass für alle $a \in G$ gilt: $a * e = e * a = a$.*
3. *Inverses Element: Für jedes $a \in G$ existiert ein inverses Element $a^{-1} \in G$ mit $a * a^{-1} = a^{-1} * a = e$.*

Eine Gruppe heißt abelsch, falls zusätzlich gilt

4. *Kommutativität: Für alle $a, b \in G$ gilt $a * b = b * a$.*

Ist die Anzahl der Elemente in G endlich, so spricht man von einer endlichen Gruppe. Die Anzahl der Elemente einer endlichen Gruppe wird als Ordnung der Gruppe bezeichnet, geschrieben als $|G|$.

Definition 3.1.2. *Eine Gruppe G ist zyklisch, falls ein Element $\alpha \in G$ existiert, sodass es für jedes $b \in G$ eine Zahl $i \in \mathbb{Z}$ gibt mit $\alpha^i = b$. So ein Element α heißt Generator von G.*

Ist g ein Generator von G, so ist die von g erzeugte Untergruppe, bezeichnet mit $\langle g \rangle$, gerade G.

Definition 3.1.3. *Sei G eine Gruppe und $a \in G$. Die Ordnung von a ist definiert als die kleinste positive ganze Zahl i mit $a^i = e$, falls so ein i existiert. Andernfalls ist die Ordnung unendlich.*

Als Folgerung aus dem Theorem von Lagrange [MOV96, 2.171] ist die Ordnung eines Elementes einer Gruppe ein Teiler der Gruppenordnung. Sei G eine Gruppe mit Ordnung p, p prim. Dann hat jedes Element der Gruppe entweder Ordnung 1 oder p. Das neutrale Element ist das einzige Element mit Ordnung 1. Daher haben alle anderen Elemente

der Gruppe die Ordnung p und sind damit auch ein Generator der Gruppe. G ist also zyklisch. Insbesondere in Kapitel 4 wird dieser Zusammenhang häufig genutzt werden.

Für die Kryptografie sind zyklische Gruppen deshalb interessant, weil man auf ihnen mathematische Probleme definieren kann, für die zumindest in bestimmten Gruppen keine effizienten Algorithmen zur Lösung des Problems bekannt sind.

Sei G eine endliche, zyklische Gruppe mit Primzahlordnung p. Sei g ein Generator von G und seien $x, y, z \in \mathbb{Z}_p$.

- **Diskreter-Logarithmus-Problem (DLP):**
  Gegeben g, g^x, berechne x.
- **Diffie-Hellmann-Problem (CDHP):**
  Gegeben g, g^x, g^y, berechne g^{xy}.
- **Diffie-Hellmann-Entscheidungsproblem (DDHP):**
  Gegeben g, g^x, g^y, g^z, entscheide ob $z = xy$.

Das wichtigste dieser drei Probleme ist das DLP, denn wenn dieses gelöst werden kann, dann können auch die anderen beiden Probleme gelöst werden. Kryptografische Verfahren, deren Sicherheit auf der Nichtlösbarkeit eines der angegebenen Probleme beruht, müssen also generell sicherstellen, dass der diskrete Logarithmus in den verwendeten Gruppen nicht effizient berechenbar ist. Ausführliche Informationen zu den genannten Problemen finden sich in [MOV96, Kap. 3.6, 3.7].

3.1.1. Bilineare Gruppen

Für sogenannte bilineare Gruppen wird eine bilineare Abbildung benötigt.

Definition 3.1.4. *Seien G_1, G_2, G_3 Gruppen und e eine Abbildung $e : G_1 \times G_2 \to G_3$. Dann ist e bilinear, falls*

$$\forall u_1, u_2 \in G_1, v \in G_2 : e(u_1 \cdot u_2, v) = e(u_1, v) \cdot e(u_2, v) \text{ und}$$
$$\forall u \in G_1,\ v_1, v_2 \in G_2 : e(u, v_1 \cdot v_2) = e(u, v_1) \cdot e(u, v_2)$$

Für zyklische G_1, G_2 lautet eine alternative Formulierung der Bedingung:

$$\forall u \in G_1, v \in G_2 \text{ und } \forall a, b \in \mathbb{Z} : e(u^a, v^b) = e(u, v)^{ab}.$$

Von bilinearen Gruppen spricht man, wenn zyklische Gruppen wie folgt mit einer bilinearen Abbildung verknüpft werden [BS04, Kap. 3]:

1. Sei p prim und seien $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ drei zyklische Gruppen mit Ordnung p.
2. Seien g_1 und g_2 Generatoren von $\mathbb{G}_1$ und $\mathbb{G}_2$.
3. ψ ist ein Isomorphismus von $\mathbb{G}_2$ nach $\mathbb{G}_1$ mit $\psi(g_2) = g_1$.
4. $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ ist eine bilineare, nicht-degenerierte Abbildung. Dazu müssen folgende Bedingungen erfüllt sein:
   - Bilinearität: $\forall u \in \mathbb{G}_1, v \in \mathbb{G}_2$ und $\forall a, b \in \mathbb{Z}$: $e(u^a, v^b) = e(u, v)^{ab}$.

- Nicht-Degeneriertheit: $e(g_1, g_2) \neq 1$.

5. Sowohl die Gruppenoperation in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ als auch ψ und e müssen effizient berechenbar sein.

Die Wahl von $\mathbb{G}_1 = \mathbb{G}_2$ ist möglich, genau solch eine Konstellation wird im weiteren Verlauf der Arbeit behandelt werden. Auch für bilineare Gruppen muss mindestens der diskrete Logarithmus in den Gruppen schwer zu berechnen sein, um sinnvolle kryptografische Verfahren mit bilinearen Gruppen konstruieren zu können. Einige Beispiele sind die Gruppensignatur in Abschnitt 3.4.2, Identity-Based-Encryption [BF03] oder der tripartite Diffie-Hellmann-Schlüsselaustausch [Jou04].

Eine wichtige Erkenntnis über bilineare Gruppen sei an dieser Stelle noch angemerkt: Das Diffie-Hellmann-Entscheidungsproblem ist in bilinearen Gruppen mit einer Abbildung $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ effizient lösbar, denn durch die Bilinearität ist $e(g^x, g^y) = e(g, g)^{xy} = e(g, g)^z = e(g, g^z)$ genau dann erfüllt, wenn $z = xy$ gilt [Men09, S. 49].

3.2. Digitale Signaturen

Digitale Signaturen sind in der digitalen Welt die Entsprechung einer handschriftlichen Unterschrift unter ein Dokument ([HPS08, Kap. 7]). Die digitale Signatur ist fest an ein Dokument beziehungsweise eine Nachricht gebunden. Der Urheber der Signatur bestätigt damit die Echtheit des Dokumentes. Digitale Signaturverfahren basieren meist auf asymmetrischer Kryptografie.

Ein digitales Signaturverfahren besteht aus den drei Algorithmen KeyGen(), Sign() und Verify():

- KeyGen() erzeugt ein asymmetrisches Schlüsselpaar bestehend aus einem privaten Schlüssel sk und einem öffentlichen Schlüssel pk.

- Sign(M, sk) erstellt zu einer Nachricht M mit Hilfe des privaten Schlüssels sk eine Signatur σ.

- Verify(M, pk, σ) überprüft mit Hilfe von pk, ob die Signatur σ eine gültige Signatur zur Nachricht M ist und liefert als Ergebnis `true` oder `false`.

Ein digitales Signaturverfahren muss die Anforderungen der Korrektheit und Nichtfälschbarkeit erfüllen. Das bedeutet zum einen, dass jede gültige Signatur durch den Verifikationsalgorithmus als gültig erkannt werden muss, zum anderen darf es für einen Angreifer ohne den privaten Schlüssel nicht möglich sein, eine gültige Signatur zu erstellen. Insbesondere darf es auch nicht möglich sein, aus einer Signatur den privaten Schlüssel zu ermitteln.

Signaturen werden meist nicht über die Nachricht M selber erstellt, sondern über einen Hashwert $H(M)$ der Nachricht. Eine kryptografische Hashfunktion ist eine Abbildung $H : \{0,1\}^* \to \{0,1\}^k$ für ein festes k und muss folgende Bedingungen erfüllen:

- H muss schwer zu invertieren sein, das heißt es ist schwierig zu einem gegebenen Wert $h(x)$ ein passendes x zu berechnen.

- H muss kollisionsresistent sein, das heißt es ist schwierig zwei Werte x, y zu finden mit $h(x) = h(y)$.
- H soll effizient berechenbar sein.

Die Kollisionsresistenz ist sehr wichtig, denn andernfalls könnte ein Angreifer zwei Nachrichten M, M' erzeugen mit $H(M) = H(M')$. Wenn er jetzt M signieren lässt, so wäre die resultierende Signatur σ auch eine gültige Unterschrift für die Nachricht M', obwohl der Signierer diese Nachricht nie unterschrieben hat.

3.3. Authentisierung

Eine Authentisierung ist eine Interaktion zwischen zwei Parteien, bei der Partei A eine Eigenschaft behauptet, die von Partei B verifiziert werden soll. Partei A muss dazu einen Beweis der Eigenschaft vorlegen.

Der Hauptanwendungsfall besteht in der Überprüfung der Identität einer Person beziehungsweise eines Benutzers. Möchte ein Benutzer beispielsweise Zugang zu einem System bekommen, so muss er gegenüber dem Dienstanbieter beweisen, dass er tatsächlich derjenige ist, der er vorgibt zu sein. Beim Zugang zu einem Computersystem behauptet der Benutzer zum Beispiel durch Eingabe seines Benutzernamens, eben diejenige Person zu sein, und beweist es durch die Kenntnis des dazugehörigen Passwortes.

Ein Benutzer kann den Beweis mit Hilfe unterschiedlicher *Authentisierungs-Faktoren* erbringen, die in drei Kategorien eingeteilt werden:

- Der Benutzer *weiß* etwas, zum Beispiel kennt er eine PIN oder ein Passwort.
- Der Benutzer *besitzt* etwas, zum Beispiel eine Smart Card oder einen Schlüssel.
- Der Benutzer *ist* etwas, hier bezieht man sich auf biometrische Merkmale wie zum Beispiel einen Fingerabdruck.

Werden Faktoren aus zwei unterschiedlichen Kategorien kombiniert, so spricht man von einer Zwei-Faktor-Authentisierung (*Two Factor Authentication*). Durch die Kombination unterschiedlicher Faktoren erhöhen solche Verfahren im Allgemeinen die Sicherheit.

Das Geldabheben am Bankautomaten ist eine sehr bekannte Zwei-Faktor-Authentisierung: Der Benutzer muss zum einen die EC-Karte in den Automaten einführen (er besitzt etwas), zum anderen muss er seine persönliche PIN eingeben (er weiß etwas). Nur wenn beides korrekt zusammenpasst, kann er Geld abheben.

3.3.1. Authentisierung mit digitalen Signaturverfahren

Eines der Standardverfahren zur Authentisierung benutzt digitale Signaturen in Form eines Challenge-Response-Protokolls ([MOV96, Kap. 10.3.3]). Der prinzipielle Ablauf ist in Abbildung 3.1 dargestellt.

Der Dienstanbieter sendet eine Zufallszahl r als Herausforderung an den Benutzer, der sich authentisieren möchte. Der Benutzer erstellt daraufhin mit seinem privaten Schlüssel eine digitale Signatur über diese Zufallszahl. Mit dem öffentlichen Schlüssel des Benutzers kann der Dienstanbieter anschließend überprüfen, ob die Signatur eine gültige Signatur

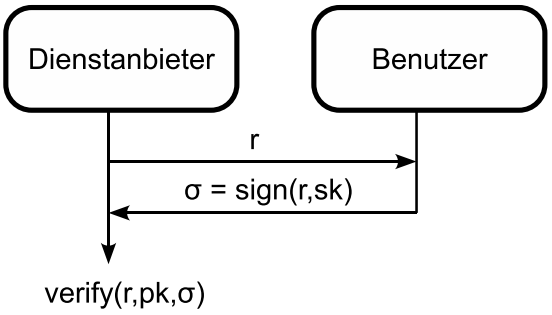

Abbildung 3.1.: Authentisierung mit digitaler Signatur

über die Zufallsnachricht ist. Ist die Signatur gültig, so hat der Benutzer damit bewiesen, dass er der Besitzer des privaten Schlüssels ist und damit die Identität bestätigt, die mit diesem privaten Schlüssel verbunden ist. Die Verbindung zwischen dem privaten Schlüssel und der (menschlichen) Identität des Benutzers wird meist durch digitale Zertifikate sichergestellt, die von vertrauenswürdigen Stellen herausgegeben werden, den sogenannten *Trust-Centern*.

3.4. Gruppensignaturen

Gruppensignaturen wurden von Chaum und van Heyst eingeführt [CH91] und bieten jedem Mitglied einer Gruppe die Möglichkeit, Nachrichten im Namen der Gruppe zu signieren. Kernelement von Gruppensignaturen ist die **Anonymität** des Signierers. Jedes Mitglied hat einen eigenen privaten Schlüssel und kann damit Signaturen erstellen. Ein Verifizierer kann mit Hilfe des einzigen öffentlichen Gruppenschlüssels zwar ermitteln, ob eine Signatur von irgendeinem Mitglied der Gruppe erstellt wurde, aber er erhält keine Information darüber, welches Mitglied konkret die Signatur erstellt hat.

Dieses steht damit ein wenig im Widerspruch zu den digitalen Signaturverfahren aus Abschnitt 3.2. Dort erstellt ein Signierer mit Hilfe seines privaten Schlüssels eine Signatur zu einer Nachricht. Jeder Verifizierer kann mit Hilfe des zugehörigen öffentlichen Schlüssels überprüfen, ob die Signatur zu der Nachricht tatsächlich von dem Besitzer des privaten Schlüssels erstellt wurde.

Bei der Gruppensignatur wird diese direkte Zuordnung gerade verhindert. Bei einer gültigen Signatur erhält der Verifizierer nur die Garantie, dass sie von irgendeinem Mitglied der Gruppe erstellt wurde. Über die konkrete Identität erfährt er nichts, somit bleibt der Signierer gegenüber dem Verifizierer anonym.

In den meisten Systemen gibt es neben dem Signierer und dem Verifizierer zusätzlich noch eine dritte Partei, den sogenannten Gruppenmanager. Dieser hat als einziger die Möglichkeit, aus einer Signatur mit Hilfe eines speziellen Schlüssels oder einer speziellen Zusatzinformation den tatsächlichen Signierer zu ermitteln und damit die Anonymität aufzuheben. Dieses Merkmal wird als **Rückverfolgbarkeit** (Traceability) bezeichnet und kann beispielsweise bei Rechtsstreitigkeiten wichtig sein.

Gruppensignaturen kommen zum Beispiel zum Einsatz, wenn in einer Firma der Zutritt zu einem bestimmten Bereich nur für Personen mit einer entsprechenden Berechtigung erlaubt ist. Gleichzeitig soll es dem Arbeitgeber aber nicht möglich sein, anhand der Zutrittsprotokolle Nutzerprofile über die Mitarbeiter zu erstellen, um so zum Beispiel zu ermitteln, wann ein Mitarbeiter kommt und geht oder wie oft er Pause macht. Mit einer Gruppensignatur kann ein Mitarbeiter die Zugehörigkeit zur Gruppe der Berechtigten beweisen, bleibt dabei aber trotzdem anonym, sodass das Erstellen von Nutzerprofilen nicht möglich ist.

3.4.1. Allgemeines Gruppensignatur-Schema mit Widerrufbarkeit

Boneh und Shacham stellen in [BS04, Kap. 2] ein allgemeines Gruppensignatur-Schema vor, das neben der Anonymität und der Rückverfolgbarkeit als besonderes Merkmal die Möglichkeit zum Widerruf von Signaturen (*Revocation*) bietet. Wird ein privater Schlüssel kompromittiert oder scheidet ein Mitglied aus der Gruppe aus, so wird ein spezielles Token zu einer sogenannten Widerrufs-Liste *(Revocation-List)* hinzugefügt, das den jeweiligen Schlüssel eindeutig identifiziert. Die Widerrufs-Liste wird jedem Verifizierer zugänglich gemacht, damit er beim Prüfen einer Signatur ermitteln kann, ob die Signatur mit einem eventuell inzwischen ungültigen Schlüssel erstellt wurde, und solch eine Signatur entsprechend abweisen kann. Da diese Überprüfung ausschließlich lokal beim Verifizierer stattfindet, wird das Verfahren mit *Verifier-Local-Revocation (VLR)* bezeichnet.

Diese Art des Widerrufs hat den großen Vorteil, dass die anderen privaten Schlüssel der Gruppe nicht angepasst werden müssen, wenn ein einzelner privater Schlüssel widerrufen wird. So kann man die privaten Schlüssel zum Beispiel fest in Chips einbrennen. Auch für Smart Cards ist das Verfahren vorteilhaft, denn auch wenn Schlüssel dort theoretisch veränderbar sind, so sind groß angelegte Aktualisierungen bestenfalls schwierig, häufig sogar absolut nicht praktikabel. Es existieren auch andere Verfahren zum Widerruf, bei denen beim Widerruf eines einzelnen Schlüssels sämtliche anderen Schlüssel aktualisiert werden müssen (vgl. [BBS04, Kap. 7]), dort wäre ein festes Einbrennen des Schlüssels in einen Chip also gar nicht möglich.

Die lokale Widerrufsüberprüfung beim Verifizierer hat allerdings auch den Nachteil, dass die Widerrufs-Liste mit jedem widerrufenen Schlüssel nach und nach immer weiter anwächst. Da bei einer Verifikation jedes Element der Liste überprüft werden muss, werden Verifikationen dadurch immer langsamer.

Das allgemeine Gruppensignatur-Schema von Boneh und Shacham besteht aus folgenden Algorithmen:

KeyGen(n, k) Die Eingabe n gibt die Anzahl der Mitglieder der zu erzeugenden Gruppe an. Der randomisierte Algorithmus erzeugt den einzigen öffentlichen Schlüssel gpk der gesamten Gruppe und den dazugehörigen privaten Schlüssel γ. Die jeweiligen privaten Schlüssel der Mitglieder werden später aus dem geheimen γ abgeleitet.

Zusätzlich bereitet er schon einmal die zwei n-elementigen Listen gsk und grt vor. gsk wird die privaten Schlüssel der Mitglieder enthalten, grt die dazugehörigen Widerrufs-Token. Die Autoren gehen von einer statischen Benutzergruppe aus, daher können im KeyGen Algorithmus direkt alle privaten Schlüssel der Mitglieder erzeugt werden. Dazu wird n-mal der Algorithmus `AddUser`() aufgerufen. Für die spätere Anwendung der

Gruppensignatur in Abschnitt 3.5 ist die Gruppe nicht statisch, es muss zumindest möglich sein, nachträglich noch Mitglieder hinzuzufügen. In dieser Konstellation wird dann erst bei Bedarf mittels `AddUser`() der entsprechende private Schlüssel $gsk[i]$ und das Widerrufs-Token $grt[i]$ erzeugt.

k ist der sogenannte Sicherheitsparameter und hat Einfluss darauf, wie groß die Ordnung der mathematischen Gruppen gewählt werden muss, in denen später die tatsächlichen Berechnungen durchgeführt werden, um die geforderte Sicherheit zu gewährleisten.

AddUser(gpk, γ, i) Der Algorithmus erzeugt bei Eingabe von gpk und γ für das Mitglied i der Gruppe einen privaten Schlüssel $gsk[i]$ und das dazugehörige Widerrufs-Token $grt[i]$.

Sign($gpk, gsk[i], M$) Mit diesem randomisierten Algorithmus kann ein Mitglied der Gruppe mit Hilfe des öffentlichen Schlüssels gpk und seinem privaten Schlüssel $gsk[i]$ eine Signatur σ zur Nachricht $M \in \{0,1\}^*$ erstellen.

Verify(gpk, RL, σ, M) Dieser Algorithmus überprüft mit Hilfe des öffentlichen Schlüssels gpk in einem ersten Schritt, ob die Signatur σ formal eine gültige Signatur der Nachricht M ist. In einem zweiten Schritt wird die Widerrufs-Liste RL analysiert. Nur wenn der Schlüssel noch nicht widerrufen ist und die Signatur gültig ist, liefert der Algorithmus als Ergebnis *gültig*, ansonsten liefert er *ungültig*.

Trace(gpk, grt, M, σ) Mit diesem Algorithmus kann der Gruppenmanager anhand einer gültigen Signatur die Identität des Signierers ermitteln. Der Gruppenmanager ist im Besitz der Liste grt, die die Widerrufs-Token sämtlicher Mitglieder der Gruppe enthält. Für jeden Benutzer startet er den Algorithmus Verify(gpk, RL, σ, M), wobei die Widerrufs-Liste nur aus dem Widerrufs-Token des jeweiligen Benutzers besteht, $RL = \{grt[i]\}$. Da die Signatur an sich gültig war, kann der Verifikationsalgorithmus nur dann ungültig liefern, wenn der Schlüssel, der beim Erstellen der Signatur benutzt wurde, durch das Widerrufs-Token ungültig wurde. Sobald der Verifikationsalgorithmus das erste Mal ungültig liefert, ist damit der Benutzer identifiziert, der die Signatur ursprünglich erstellt hat. Wird die Signatur in jedem Durchlauf korrekt verifiziert, so liefert der Trace-Algorithmus als Ergebnis *fail*.

RevokeUser($RL, grt[i]$) Mit diesem Algorithmus kann der Gruppenmanager einen privaten Schlüssel eines Gruppenmitglieds widerrufen. Dazu wird das Widerrufs-Token $grt[i]$ an die Liste RL angehängt und als neue Liste RL' zurückgegeben.

Anwendung des Schemas

Initial wird die Gruppe vom Gruppenmanager erzeugt. Mit Hilfe des **KeyGen()** Algorithmus erzeugt er den einzigen öffentlichen Schlüssel der Gruppe. Für jedes Mitglied der Gruppe wird mittels **AddUser**() ein privater Schlüssel erzeugt. Zusätzlich dazu erhält der Gruppenmanager die Liste der Widerrufs-Token.

Ab diesem Zeitpunkt kann das Schema aktiv genutzt werden, jedes Mitglied der Gruppe kann mit seinem privaten Schlüssel Signaturen zu beliebigen Nachrichten erstellen, die

von einem beliebigen Verifizierer mit Hilfe des einzigen öffentlichen Schlüssels verifiziert werden können, unter Wahrung der Anonymität des Signierers.

Im Fall der Fälle kann einzig der Gruppenmanager die Anonymität aufheben und die konkrete Identität des Signierers ermitteln. Als Anmerkung sei noch erwähnt, dass man die Rolle des Gruppenmanagers noch einmal unterteilen kann. Eine Person kann das Schema initial aufsetzen und die Schlüssel erzeugen, während eine andere Person für das Aufheben der Anonymität und das Veröffentlichen der Widerrufs-Token zuständig ist.

3.4.2. Definition des Boneh-Shacham-Gruppensignatur-Schemas

Boneh und Shacham stellen in [BS04] neben dem allgemeinen Schema auch eine konkrete Umsetzung des Gruppensignatur-Schemas vor, bei dem sowohl die Länge der Signaturen als auch das Sicherheitsniveau vergleichbar zu regulären RSA-Signaturen sind.

Das Schema basiert auf bilinearen Gruppen entsprechend Abschnitt 3.1.1. Auch wenn viele konkrete Konstruktionen für eine geeignete bilineare Abbildung auf elliptischen Kurven basieren, ist das Schema relativ allgemein definiert und erlaubt auch andere Möglichkeiten.

Neben den bilinearen Gruppen werden für das Schema auch noch spezielle Hashfunktionen H und H_0 benötigt, deren Realisierung bei der späteren Umsetzung des Gruppensignatur-Schemas eine weitere Herausforderung darstellen wird.

Im Folgenden werden die Elemente und Operationen des Gruppensignatur-Schemas detailliert spezifiziert.

System Parameter

- $\mathbb{G}_1, \mathbb{G}_2$ und $\mathbb{G}_T$ sind multiplikative zyklische Gruppen mit Primzahlordnung p.
- ψ ist ein Isomorphismus von $\mathbb{G}_2$ nach $\mathbb{G}_1$.
- g_1 ist ein Generator von $\mathbb{G}_1$ und g_2 ist ein Generator von $\mathbb{G}_2$ mit $\psi(g_2) = g_1$.
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ ist eine bilineare, nicht-degenerierte Abbildung. Dazu müssen folgende Bedingungen erfüllt sein:
  - Bilinearität: $\forall u \in \mathbb{G}_1, v \in \mathbb{G}_2$ und $\forall a, b \in \mathbb{Z}$: $e(u^a, v^b) = e(u, v)^{ab}$.
  - Nicht-Degeneriertheit: $e(g_1, g_2) \neq 1$.

  Zusätzlich muss e effizient berechenbar sein.
- H ist eine Hash-Funktion von $\{0,1\}^*$ nach $\mathbb{Z}_p$, H_0 ist eine Hash-Funktion von $\{0,1\}^*$ nach $\mathbb{G}_2 \times \mathbb{G}_2$.

Schlüssel

- öffentlicher Schlüssel der Gruppe: $gpk = (g_1, g_2, w)$ mit $w = g_2^\gamma$. Dabei ist $\gamma \in \mathbb{Z}_p$ ein geheimer Wert, wenn man so möchte der zu gpk zugehörige private Schlüssel. Er wird ausschließlich für die Erzeugung der privaten Schlüssel der Gruppenmitglieder benutzt. Nur derjenige, der die Schlüssel erzeugt, darf γ kennen.

- privater Schlüssel eines Gruppenmitglieds: ein Paar (A_i, x_i), wobei $A_i \in \mathbb{G}_1$ und $x_i \in \mathbb{Z}_p$ mit $A_i^{x_i+\gamma} = g_1$. Für solch ein Paar gilt aufgrund der Bilinearität von e:

  $$e(A_i, wg_2^{x_i}) = e(A_i, g_2^{\gamma} g_2^{x_i}) = e(A_i, g_2^{\gamma+x_i}) = e(A_i, g_2)^{\gamma+x_i} = e(A_i^{\gamma+x_i}, g_2) = e(g_1, g_2).$$

  Für ein gegebenes x_i kann A_i vom Besitzer von γ berechnet werden.

Erzeugung der Schlüssel

Der Algorithmus **KeyGen** führt folgende Schritte aus:

- Wähle zufällig gleichverteilt einen Generator $g_2 \in \mathbb{G}_2$ und setze $g_1 = \psi(g_2)$.
- Wähle ein zufälliges $\gamma \in \mathbb{Z}_p^*$ und setze $\omega = g_2^{\gamma}$.

Der öffentliche Schlüssel ist $gpk = (g_1, g_2, \omega)$. γ ist geheim und darf nur dem Erzeuger der Schlüssel bekannt sein.

Der Algorithmus **AddUser** berechnet für jedes Mitglied der Gruppe ein Tupel (A_i, x_i), wobei x_i zufällig so aus $\mathbb{Z}_p^*$ gewählt wird, dass $\gamma + x_i \neq 0$ ist. Setze $A_i = g_1^{1/(\gamma+x_i)}$.
Die privaten Schlüssel sind dann $gsk[i] = (A_i, x_i)$. Die Widerrufs-Liste besteht aus den A_i der jeweiligen Mitglieder der Gruppe, also $grt[i] = A_i$.

Berechnung der Signatur über M

Sei $gsk[i] = (A_i, x_i)$ der private Schlüssel des Signierers. Er wird im Folgenden verkürzt mit (A, x) bezeichnet. Mit Hilfe des öffentlichen Schlüssels gpk, der Nachricht $M \in \{0,1\}^*$ und einem zufälligen $r \in \mathbb{Z}_p$ berechnet der Signierer zwei Generatoren:

- $(\hat{u}, \hat{v}) = H_0(gpk, M, r) \in \mathbb{G}_2 \times \mathbb{G}_2$

Als nächsten Schritt berechnet der Signierer die Abbildungen $u = \psi(\hat{u}),\ v = \psi(\hat{v})$. Mit einem zufälligen $\alpha \in \mathbb{Z}_p$ berechnet er:

- $T_1 = u^{\alpha},\ T_2 = Av^{\alpha}$
- $\delta = x\alpha$

Mit weiteren Zufallswerten $r_{\alpha}, r_x, r_{\delta} \in \mathbb{Z}_p$, die der Verschleierung dienen, berechnet er folgende Hilfswerte:

- $R_1 = u^{r_{\alpha}}$
- $R_2 = e(T_2, g_2)^{r_x} \cdot e(v, w)^{-r_{\alpha}} \cdot e(v, g_2)^{-r_{\delta}}$
- $R_3 = T_1^{r_x} \cdot u^{-r_{\delta}}$

Mit diesen Werten berechnet er einen Herausforderungswert (*challenge*):

- $c = H(gpk, M, r, T_1, T_2, R_1, R_2, R_3) \in \mathbb{Z}_p$

Der Begriff *Herausforderung* stammt eigentlich aus dem Bereich der Identifikationsprotokolle und erscheint daher erst einmal etwas fehl am Platz. Abschnitt 3.4.5 wird die Namensgebung näher erläutern.

Mit Hilfe von c werden im nächsten Schritt die letzten Elemente der Signatur berechnet:

- $s_\alpha = r_\alpha + c\alpha,\ s_x = r_x + cx,\ s_\delta = r_\delta + c\delta$

Die Signatur σ der Nachricht M lautet dann:

- $\sigma = (r, c, T_1, T_2, s_\alpha, s_x, s_\delta)$.

Verifikation der Signatur σ

Die Verifikation erfolgt in zwei Schritten. Im ersten Schritt wird überprüft, ob die Signatur korrekt gebildet wurde. Im zweiten Schritt wird überprüft, ob der Schlüssel, mit dem die Signatur erzeugt wurde, noch nicht widerrufen wurde. Nur wenn beides zutrifft, ist die Signatur gültig.

1. Der Verifizierer berechnet analog zum Signierer $(\hat{u}, \hat{v})$ und deren Abbildung u, v in $\mathbb{G}_1$. Da r Teil der Signatur ist und die übrigen Parameter öffentlich bekannt sind, kann er dieselben Rechenschritte ausführen wie der Signierer, um $\hat{u}$ und $\hat{v}$ zu erhalten. Anschließend berechnet er die Hilfswerte:
   - $\tilde{R_1} = u^{s_\alpha} / T_1^c$
   - $\tilde{R_2} = e(T_2, g_2)^{s_x} \cdot e(v, w)^{-s_\alpha} \cdot e(v, g_2)^{-s_\delta} \cdot (e(T_2, w)/e(g_1, g_2))^c$
   - $\tilde{R_3} = T_1^{s_x} \cdot u^{-s_\delta}$

   Damit überprüft er, ob $c \stackrel{?}{=} H(gpk, M, r, T_1, T_2, \tilde{R_1}, \tilde{R_2}, \tilde{R_3})$ ist. Bei Übereinstimmung wird die Signatur akzeptiert, andernfalls verworfen.

2. Der Verifizierer überprüft für jedes Element der Widerrufs-Liste $A \in RL$, ob der Wert A in (T_1, T_2) enthalten ist. Dazu ist für jedes A folgende Berechnung nötig:
   - $e(T_2/A, \hat{u}) \stackrel{?}{=} e(T_1, \hat{v})$

   Stimmt der Vergleich für ein A überein, so wurde der für die Signatur verwendete Schlüssel inzwischen widerrufen, womit die Signatur ungültig ist.

3.4.3. Korrektheit

Damit das Gruppensignatur-Schema ein gültiges Gruppensignatur-Schema ist, muss mindestens die Anforderung der *Korrektheit* erfüllt sein. Dazu muss gezeigt werden, dass jede regulär erzeugte Signatur auch als gültig verifiziert wird, sofern der Schlüssel nicht widerrufen wurde.

Ein ehrlicher Signierer erzeugt die Signatur $\sigma = (r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$ genau nach dem oben skizzierten Schema. Daher benutzt der Signierer auch dieselben Generatoren $\hat{u}, \hat{v}$ beziehungsweise u, v wie der Verifizierer. Die Signatur wird nur dann akzeptiert, wenn der Hashwert der Funktion H exakt der Herausforderung c entspricht. Dies ist nur dann möglich, wenn die Eingabe für H sowohl beim Signierer als auch beim Verifizierer identisch ist (andernfalls hätte man eine Kollision in H gefunden). Mit Ausnahme von

R_1, R_2, R_3 stimmen die Eingabewert für H grundsätzlich überein, da sie in der Signatur enthalten sind und vom Verifizierer einfach übernommen werden. Daher verbleibt zu zeigen, dass die Werte $\tilde{R_1}, \tilde{R_2}, \tilde{R_3}$, die vom Verifizierer eigenständig berechnet werden, tatsächlich den originalen Werten R_1, R_2, R_3 entsprechen.

$$\tilde{R_1} = \frac{u^{s_\alpha}}{T_1^c} = \frac{u^{r_\alpha + c\alpha}}{(u^\alpha)^c} = u^{r_\alpha} = R_1.$$

$$\tilde{R_3} = T_1^{s_x} u^{-s_\delta} = (u^\alpha)^{r_x + cx} \cdot u^{-r_\delta - cx\alpha} = (u^\alpha)^{r_x} \cdot u^{-r_\delta} = T_1^{r_x} \cdot u^{-r_\delta} = R_3.$$

$$\begin{aligned}
\tilde{R_2} &= e(T_2, g_2)^{s_x} \cdot e(v, w)^{-s_\alpha} \cdot e(v, g_2)^{-s_\delta} \cdot \left(\frac{e(T_2, w)}{e(g_1, g_2)}\right)^c \\
&= \left(e(T_2, g_2)^{r_x} \cdot e(v, w)^{-r_\alpha} \cdot e(v, g_2)^{-r_\delta}\right) \cdot \left(e(T_2, g_2)^x \cdot e(v, w)^{-\alpha} \cdot e(v, g_2)^{-x\alpha} \cdot \frac{e(T_2, w)}{e(g_1, g_2)}\right)^c \\
&= R_2 \cdot \left(\frac{e(T_2 v^{-\alpha}, w g_2^x)}{e(g_1, g_2)}\right)^c \\
&= R_2 \cdot \left(\frac{e(A, w g_2^x)}{e(g_1, g_2)}\right)^c \\
&= R_2.
\end{aligned}$$

Bei einer gültigen Signatur ist $T_1 = \psi(\hat{u})^\alpha$, $T_2 = A_i \psi(\hat{v})^\alpha$ für ein zufälliges α. Der Widerrufs-Check überprüft, ob $(\hat{u}, \hat{v}, T_1, T_2/A)$ ein sogenanntes Co-Diffie-Hellman-Tupel bilden. Dies ist genau dann der Fall, wenn das A aus der Widerrufs-Liste das in T_2 enthaltene A_i eliminiert, also $A = A_i$ gilt. Ist aber $A = A_i$, so ist der für die Signatur verwendete Schlüssel gerade in der Widerrufs-Liste enthalten und die Signatur wird abgewiesen, so wie es gefordert ist.

3.4.4. Sicherheit

Damit das Schema nicht nur ein korrektes, sondern auch ein sicheres Gruppensignatur Schema ist, müssen noch die beiden Anforderungen *Rückverfolgbarkeit* und *Selfless-Anonymity* gezeigt werden. Die Anforderungen orientieren sich an den von Bellare et al. ([BMW03]) festgelegten Formalismen. Dort wird gezeigt, dass es für eine Gruppensignatur ausreichend ist, Rückverfolgbarkeit und Anonymität zu beweisen, da dadurch automatisch auch andere sicherheitsrelevante Anforderungen erfüllt sind. So folgt zum Beispiel aus der Rückverfolgbarkeit die Nicht-Fälschbarkeit von Signaturen, denn könnte ein Angreifer, der kein Gruppenmitglied ist, eine Signatur fälschen, so wäre diese Signatur nicht zum Angreifer zurück zu verfolgen, da er ja gar nicht im System existiert.

Die Sicherheit des Schemas basiert auf zwei mathematischen Problemen, die als schwierig angesehen werden: zum einen auf der *Strong-Diffie-Hellman (SDH)*-Vermutung, zum anderen auf der *Decision-Linear*-Vermutung.

q-Strong Diffie-Hellman-Vermutung

Etwas grob gesprochen besagt die q-Strong Diffie-Hellman-Vermutung folgendes: Seien $\mathbb{G}_1, \mathbb{G}_2$ zyklische Gruppen mit Primzahlordnung p, wobei $\mathbb{G}_1 = \mathbb{G}_2$ erlaubt ist. Sei g_1 ein Generator von $\mathbb{G}_1$, g_2 ein Generator von $\mathbb{G}_2$. Dann ist folgendes Problem nicht effizient berechenbar: Bei Eingabe eines $(q+2)$-Tupels $(g_1, g_2, g_2^\gamma, \ldots, g_2^{(\gamma^q)})$, berechne als Ausgabe ein Paar $(g_1^{1/(\gamma+x)}, x)$ für ein $x \in \mathbb{Z}_p^*$ eigener Wahl.

Boneh und Boyen zeigen in [BB08], dass die q-SDH-Vermutung in generischen Gruppen gilt. Generische Gruppen sind ein idealisiertes Berechenbarkeitsmodell (siehe [Sho97] und [Mau05]), bei dem der Angreifer gewissen Einschränkungen unterliegt. So werden Gruppenelemente als eine eindeutige, aber zufällige Bitfolge dargestellt, sodass ein Angreifer aus der Darstellung keine zusätzlichen Informationen ableiten kann, er hat einzig die Möglichkeit Elemente auf Gleichheit zu testen. Die Gruppenoperation wird in Form eines Orakels bereitgestellt, sodass ein Angreifer auch hier als einziges das Ergebnis der Operation erhält, keine zusätzlichen Informationen. Mit einem derart eingeschränkten Angreifer können für bestimmte Probleme untere Schranken für die Komplexität des Problems bewiesen werden. Es sei angemerkt, dass diese Einschränkung des Angreifers in der Realität weder für endliche Körper noch für elliptische Kurven zutrifft. Nichtsdestotrotz stärkt der Beweis das Vertrauen, dass das Problem im Sinne der Komplexitätstheorie ein schweres Problem sein könnte.

Decision-Linear-Vermutung

Sei $g_1 \in \mathbb{G}_1$ wie zuvor, sowie $u,\ v,\ h$ beliebige Generatoren von $\mathbb{G}_1$, dann ist das folgende Problem ebenfalls nicht effizient berechenbar: Bei Eingabe $u, v, h, u^a, v^b, h^c \in \mathbb{G}_1$ gebe *true* aus, falls $a + b = c$, ansonsten *false*.

Es kann gezeigt werden, dass ein Algorithmus, der das Decision-Linear-Problem in $\mathbb{G}_1$ löst, auch einen Algorithmus liefert, der das Diffie-Hellmann-Entscheidungsproblem in $\mathbb{G}_1$ löst. Es wird vermutet, dass der Umkehrschluß nicht gilt. Daher wird das Decision-Linear-Problem in der Gruppe $\mathbb{G}_1$ auch im Kontext von bilinearen Gruppen als schwierig angesehen, bei denen das DDH-Problem durch die Existenz einer bilinearen Abbildung $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ leicht lösbar ist. In [BBS04, Kap. 8] wird gezeigt, dass die Decision-Linear-Vermutung in generischen Gruppen gilt.

Rückverfolgbarkeit und Selfless-Anonymity

Ein Gruppensignatur-Schema gilt als *rückverfolgbar* (traceable), wenn es einem Angreifer beziehungsweise einer Angreifergruppe nicht möglich ist, eine (gefälschte) Signatur zu erstellen, für die der Trace-Algorithmus jemand anderes als den Angreifer beziehungsweise ein Mitglied der Angreifergruppe als Urheber ermittelt. Dem Angreifer ist es erlaubt, gültige Signaturen und sogar private Schlüssel von Mitgliedern der Gruppe zu erfragen. In letzterem Fall wird das jeweilige Mitglied allerdings zur Angreifergruppe hinzugezählt.

Umgekehrt formuliert ist das Schema genau dann rückverfolgbar, wenn der Trace-Algorithmus immer den tatsächlichen Urheber der Signatur ermittelt. Es ist einem Angreifer also nicht möglich, jemand anderem eine Signatur unterzuschieben, selbst dann nicht, wenn sich mehrere Angreifer zusammen tun.

Selfless-Anonymity besagt, dass es einem Angreifer nicht möglich ist zu entscheiden, ob eine Signatur σ über eine Nachricht M von Benutzer A oder Benutzer B stammt, selbst wenn er zuvor beliebige andere Signaturen gesehen hat und sogar private Schlüssel von anderen Benutzern kennt (mit Ausnahme der Schlüssel von Benutzer A und B). Aus der Signatur erfährt ein Angreifer also nichts über den Urheber der Signatur, er bleibt anonym. Ausgenommen davon ist nur der Ersteller einer Signatur, der auch zu einem späteren Zeitpunkt noch überprüfen kann, ob eine beliebige Signatur von ihm selbst erstellt wurde, daher der Zusatz *Selfless*.
[BBS04] enthält Beweise zur Rückverfolgbarkeit und Selfless-Anonymity.

Theorem 3.4.1. *Falls q-SDH in* $(\mathbb{G}_1, \mathbb{G}_2)$ *schwer ist, dann garantiert das Gruppensignatur-Schema für maximal n Benutzer Rückverfolgbarkeit, mit* $n = q - 1$.

Theorem 3.4.2. *Das Gruppensignatur-Schema mit den Gruppen* $(\mathbb{G}_1, \mathbb{G}_2)$ *gewährleistet Selfless-Anonymity, falls die Decision-Linear-Vermutung in* $\mathbb{G}_2$ *gilt.*

Die mathematisch exakten Ergebnisse, bei denen auch die Fähigkeiten des Angreifers einbezogen werden, sind als Theorem 6.2 und Lemma 6.1 in [BBS04] zu finden.

3.4.5. Anmerkungen zum Gruppensignatur-Schema

Das Gruppensignatur-Schema wurde aus einem neuartigen Protokoll abgeleitet, das den Besitz eines SDH-Tupels beweist. Das Protokoll beweist den Besitz eines Geheimnisses und ist sogar fast ein Zero-Knowledge-Protokoll. Da allerdings mit Hilfe der Widerrufs-Liste ermittelt werden kann, ob die Signatur von einem inzwischen widerrufenen Benutzer kommt, gewinnt man zusätzliches Wissen. Davon abgesehen erhält man aus dem Protokoll aber keinerlei Information über das Geheimnis, das in diesem Fall der geheime Schlüssel beziehungsweise das darin gespeicherte SDH-Tupel ist.

Bei einem Zero-Knowledge-Protokoll versucht der Beweisende (*Prover*) den Verifizierer (*Verifier*) davon zu überzeugen, dass er Kenntnis von einem Geheimnis hat. Nach Abschluss des Protokolls ist der Verifizierer davon überzeugt, dass der Beweisende das Geheimnis kennt, hat aber selber nichts über das Geheimnis gelernt. Der interaktive Ablauf ist meist eine Abfolge von *Festlegung* des Beweisenden, *Herausforderung* durch den Verifizierer, *Antwort* des Beweisenden und *Überprüfung* durch den Verifizierer. Die Herausforderung sollte so aussehen, dass der Beweisende sie nur mit Kenntnis des Geheimnisses bestehen kann.

Zero-Knowledge-Protokolle sehen häufig so aus, dass der Beweisende bestimmte Dinge in einer schwierigen Reihenfolge berechnen muss, was nur mit Kenntnis des Geheimnisses möglich ist. Wählt man eine andere Reihenfolge, so sind die Zusammenhänge trivial, weswegen ein Angreifer allein aus dem Mithören eines solchen Ablaufs keine Erkenntnisse über das Geheimnis erlangt, da er sich die Werte in der einfachen Reihenfolge auch selber hätte erzeugen können.

Das interaktive Identifikationsprotokoll, mit dem der Beweisende den Verifizierer davon überzeugt, dass er im Besitz eines SDH-Tupels ist, ohne das Tupel selber zu verraten, führt dieselben Schritte aus wie in $\texttt{sign}(gpk, gsk[i], M)$ und $\texttt{verify}(gpk, M, \sigma)$, mit Ausnahme der Berechnung der Hashfunktionen H und H_0. Denn ein Verfahren, um aus einem interaktiven Identifikationsprotokoll ein Signaturverfahren zu machen, besteht darin, anstatt der interaktiven Eingaben des Verifizierers Hashfunktionen zu benutzen. Denn Hashwerte sind fast wie Zufallswerte und stellen daher für den Beweisenden eine echte Herausforderung dar.

Anstatt $(\hat{u}, \hat{v}) = H_0(gpk, M, r)$ zu berechnen, erhält der Beweisende im Identifikationsprotokoll vom Verifizierer zwei zufällig gewählte Generatoren $(\hat{u}, \hat{v})$. Genauso berechnet der Beweisende nicht selber $c = H(gpk, M, r, T_1, T_2, R_1, R_2, R_3)$, sondern der Verifizierer wählt $c \in \mathbb{Z}_p$ zufällig und sendet es als Herausforderung an den Beweisenden.

Analog zum Algorithmus `verify` akzeptiert der Verifizierer beim Identifikationsprotokoll den Wissensbeweis, falls die drei Gleichungen bezüglich $\tilde{R}_1, \tilde{R}_2, \tilde{R}_3$ erfüllt sind. Es ist wichtig, dass der Beweisende seine Festlegung von $\alpha, x, \delta, r_\alpha, r_x, r_\delta$ trifft, bevor er die

Herausforderung c vom Verifizierer erhält, andernfalls könnte er das Protokoll auch ohne Kenntnis eines SDH-Tupels bestehen.

Die Zero-Knowledge Eigenschaft des Identifikationsprotkolls ist ein wichtiger Bestandteil des Beweises der Selfless-Anonymity.

3.5. Biometrische Authentisierung mit einer Gruppensignatur

Im Rahmen einer Authentisierung möchte ein Benutzer sich gegenüber einem Dienstanbieter in irgendeiner Form ausweisen. Angenommen, ein Dienstanbieter erlaubt den Zugriff auf gewisse Inhalte nur für volljährige Personen. Damit ein Benutzer Zugriff erlangen kann, muss er sich zuerst gegenüber dem Dienstanbieter identifizieren, damit dieser anhand der Identität des Benutzers entscheiden kann, ob der Benutzer über 18 Jahre alt ist und damit Zugriff erhalten darf oder nicht.

Authentisierung mit einer digitalen Signatur

Entsprechend Abschnitt 3.3.1 könnte ein Benutzer sich mit Hilfe einer digitalen Signatur authentisieren. Nachdem der Dienstanbieter als Herausforderung eine Zufallszahl geschickt hat, erstellt der Benutzer eine digitale Signatur über die Zufallszahl und schickt sie zurück an den Dienstanbieter. Anhand der Gültigkeit der Signatur kann der Dienstanbieter ermitteln, ob der Benutzer derjenige ist, der er vorgibt zu sein, und dann entscheiden, ob er dem Benutzer Zugriff erteilt oder nicht.

In Bezug auf das Eingangsbeispiel hat der Benutzer mit diesem Verfahren allerdings mehr Daten preis gegeben, als eigentlich notwendig war. Der Dienstanbieter kennt nun die genaue Identität des Benutzers (oder zumindest alle Daten, die er über ihn gespeichert hat), und kann damit jetzt zum Beispiel Nutzungsprofile erstellen, während er eigentlich nur wissen musste, ob der Benutzer über 18 ist oder nicht.

Authentisierung mit einer Gruppensignatur

Hier bieten Gruppensignaturen durch die garantierte Anonymität Vorteile. Angenommen, es existiert eine Gruppe der über 18-jährigen, die entsprechend der Vorgabe durch das Gruppensignatur-Schema angelegt wurde. Nach dem gleichen Vorgehen wie zuvor kann ein Benutzer auch mit einer Gruppensignatur seine Identität zumindest insoweit beweisen, dass er ein gültiges Mitglied der Gruppe der über 18-jährigen ist.

Der Benutzer signiert die Zufallsnachricht mit seinem privaten Schlüssel, woraufhin der Dienstanbieter mit dem öffentlichen Schlüssel der Gruppe die Zugehörigkeit des Benutzers zu dieser Gruppe verifizieren kann, ohne allerdings etwas über die konkrete Identität des Benutzers zu erfahren.

Biometrische Authentisierung

Bringer et al. stellen in [BCPZ08] ein Verfahren vor, welches eine biometrische Authentisierung mit Hilfe einer Gruppensignatur realisiert. Das Verfahren benutzt zum einen ein biometrisches Merkmal, zum anderen benötigt es ein dazugehöriges Security-Token, welches in diesem Fall eine Smart Card ist, es ist daher eine Zwei-Faktor-Authentisierung. In

dem entstandenen Protokoll kann man sich nur dann korrekt gegenüber einem Dienstanbieter authentisieren, wenn das biometrische Merkmal zu den Daten in der Smart Card passt. Durch den Einsatz der Gruppensignatur geschieht die Authentisierung auf anonyme Weise. Die Privatsphäre des Anfragenden bleibt gewahrt, der Dienstanbieter erfährt weder etwas über die konkrete Identität des Anfragenden, noch etwas über dessen biometrische Daten. Allerdings kann der Dienstanbieter allein anhand der Gültigkeit der Signatur entscheiden, ob der Zugriff gewährt werden darf oder nicht.

Grob skizziert sieht das Verfahren zur biometrischen Authentisierung wie folgt aus:

- Im ersten Schritt wird aus einem biometrischen Merkmal ein geheimer Schlüssel abgeleitet, der auf einer Smart Card sicher gespeichert wird. Dieser Schlüssel ist fest an die biometrische Identität geknüpft.

- Im zweiten Schritt wird dieser Schlüssel in das in Abschnitt 3.4 beschriebene Gruppensignatur-Schema integriert, um damit anschließend anonyme Signaturen erstellen zu können und damit die Authentisierung durchzuführen.

In den folgenden Abschnitten wird die konkrete Umsetzung der biometrischen Authentisierung mit Hilfe der Gruppensignatur genauer erläutert.

3.5.1. Ableitung eines geheimen Schlüssels

Bringer et al. benutzen ein sehr einfaches Verfahren, um aus einem biometrischen Merkmal einen geheimen Schlüssel zu generieren. Aus ihrer Sicht muss das biometrische Merkmal als öffentlich bekannt angesehen werden. Ein Mensch hinterlässt zum Beispiel ständig seine Fingerabdrücke (oder zumindest Teilabdrücke), sobald er etwas anfasst. Für einen Angreifer sollte es daher nicht schwierig sein, in den Besitz eines Fingerabdrucks zu kommen. Was allerdings nicht öffentlich bekannt ist, ist das Ergebnis einer einmaligen Abnahme eines Fingerabdrucks an einem Sensor. Da auch mehrmalige Abnahmen ein und desselben Merkmals immer unterschiedliche Ergebnisse produzieren, machen sich die Autoren diesen Umstand zu nutze.

Die Abweichung zwischen den mehrmaligen Abnahmen wird häufig als ein Fehler angesehen, der mit aufwendigen Fehlerkorrektur-Verfahren versucht wird zu eliminieren. Hier allerdings wird genau diese Abweichung als eine dem biometrischen Merkmal innewohnende Zufallskomponente benutzt, die von einem Angreifer nicht vorhergesagt werden kann. Daher schlagen die Autoren vor, den Hashwert des bei der initialen Abnahme gewonnenen Referenztemplates als geheimen Schlüssel zu nutzen. Um auch später den Schlüssel nutzen zu können, muss das Referenztemplate auf einer Chipkarte gespeichert werden. Bei einer erneuten Abnahme vergleicht der biometrische Sensor die frische Abnahme gegen das Referenztemplate mit Hilfe klassischer Matchingverfahren. Bei Übereinstimmung kann erneut der Hashwert über das gespeicherte Referenztemplate berechnet werden, um wiederum denselben geheimen Schlüssel zu berechnen. Wohlgemerkt wird der Hashwert über das Referenztemplate berechnet, nicht über die neue Abnahme des Merkmals. Sollte der Schlüssel einmal kompromittiert werden, so wird einfach ein neues initiales Referenztemplate erzeugt, woraus wiederum ein neuer geheimer Schlüssel abgeleitet werden kann.

Allerdings stellt sich automatisch die Frage, ob die Abweichung zwischen mehrmaligen Abnahmen eines biometrisches Merkmals tatsächlich genug Zufall enthält, sodass ein

Angreifer diese selbst dann nicht effizient ermitteln kann, wenn er schon die Ergebnisse einiger früherer Abnahmen gesehen hat.

Dazu untersuchten die Autoren die *Iris-Challenge-Evaluation (ICE)* Datenbank. Diese Datenbank enthält eine Vielzahl von Iris-Scans unterschiedlicher Augen, insbesondere aber auch mehrere Scans von ein und demselben Auge. Die Datenbank dient dazu, Matching-Algorithmen auf ihre Güte zu überprüfen. Für jedes Bild wurde ein 256 Byte Informationsvektor I berechnet, dazu ein 256 Byte Maskierungsvektor M, der angibt, ob das entsprechende Bit im Informationsvektor eine gültige Information enthält oder zum Beispiel aufgrund von Reflexionen bei der Aufnahme oder Augenlidbewegungen ungültig ist. Das Matching bestand in einer Auswertung der Hammingdistanz zweier Informationsvektoren I_1, I_2, wobei nur diejenigen Bits berücksichtigt werden, die sowohl in M_1 als auch in M_2 als gültig markiert sind.

Die Autoren untersuchten die Datenbank dahingehend, wie unterschiedlich zwei Iris-Scans desselben Auges sind und welchen Aufwand ein Angreifer betreiben muss, um von einem Scan b_i zur Referenzabnahme b zu gelangen. Die geringste Hammingdistanz zweier b_i, b betrug 44 Bits, bei einer Länge von 2048 Bit. Der Unterschied bezieht sich dabei nur auf diejenigen Bits der Informationsvektoren, die nicht durch die Maskierungsvektoren ausgeblendet wurden. Das heißt, ein Angreifer müsste b_i an mindestens 44 Stellen ändern, um b zu erhalten. Angenommen die Abweichungen entsprächen einer Gleichverteilung, so hätte er allein dafür $\binom{2048}{44} \approx 2^{302}$ Möglichkeiten, da er nicht weiß, an welchen Stellen die Unterschiede auftreten. Zieht man auch noch die Maskierungsvektoren in Betracht, die sich ebenfalls unterscheiden, so muss der Angreifer mindestens für diejenigen Bits eine Belegung erraten, die durch M_{b_i}, aber nicht durch M_b ausgeblendet werden, sodass die Anzahl der Möglichkeiten noch größer wird. Für die gesamte Datenbank ermittelten die Autoren ein Minimum von etwa 2^{500} Möglichkeiten. In der Praxis entsprechen die Abweichungen zwar keiner Gleichverteilung, aber die Autoren sind überzeugt, dass selbst dann die Anzahl der Möglichkeiten sehr groß bleibt.

In Fällen, wo die Abweichungen eventuell doch zu gering sind, besteht noch die Möglichkeit, weitere Zufallsbits in den Informationsvektor einzubringen, und zwar gerade an den Stellen, die durch den Maskierungsvektor ausgeblendet werden. So haben die Bits auf das biometrische Matching keine Auswirkung, erhöhen aber die Abweichung zweier kompletter Vektoren. Die Autoren sind ferner überzeugt, dass die Beobachtungen bei den Iris-Scans auch bei anderen biometrischen Merkmalen in ähnlicher Art und Weise zutreffen.

3.5.2. Ablauf einer Authentisierung

Das Schema für eine sichere biometrische Authentisierung besteht im Kern aus vier Parteien:

1. Der menschliche Benutzer $\mathcal{H}$, der sich mit seinen biometrischen Daten bei einem Dienstanbieter authentisieren möchte. Die Smart Card des Benutzers mit den geheimen Daten b und A dient erst einmal nur als sicherer Speicher.

2. Der Sensor-Client $\mathcal{S}$, der mit einem biometrischen Sensor das biometrische Merkmal des Benutzers abnimmt und mit dem Dienstanbieter kommuniziert.

3. Der Dienstanbieter $\mathcal{P}$, der die Authentisierungsanfrage des Benutzers erhält und entscheiden muss, ob der Benutzer Zugriff erhält.

4. der Kartenherausgeber $\mathcal{I}$, der zwei Master-Schlüssel besitzt: Zum einen den geheimen Schlüssel γ, mit dessen Hilfe die einzelnen privaten Schlüssel für die Benutzer im Gruppensignatur-Schema generiert werden. Im Kontext des Gruppensignatur Schemas übernimmt $\mathcal{I}$ die Rolle des Gruppenmanagers. Zum anderen besitzt $\mathcal{I}$ einen privaten Schlüssel λ, mit dessen Hilfe der Kartenherausgeber nachträglich Informationen über das bei einer Authentisierung abgenommene biometrische Merkmal erhalten kann. Dieser Schlüssel kommt nur in Ausnahmesituationen zum Einsatz, zum Beispiel bei Rechtsstreitigkeiten. Bei der normalen Authentisierung zwischen Benutzer $\mathcal{H}$ und Dienstanbieter $\mathcal{P}$ hat er keine Relevanz. Λ bezeichnet den zu λ gehörigen öffentlichen Schlüssel.

Das Verfahren versucht eine untrennbare Verbindung zwischen dem menschlichen Benutzer $\mathcal{H}$, seinem biometrischen Merkmal, seinem daraus abgeleiteten Schlüssel x und dem damit erzeugten Schlüssel des Gruppensignatur-Schemas $gsk_i = (A, x)$ herzustellen. Nur wenn diese Verbindung gewährleistet ist, kann sich der Dienstanbieter sicher sein, dass hinter einer gültigen Gruppensignatur ein Benutzer steht, dessen Identität vom Kartenherausgeber überprüft und akzeptiert wurde.

Enrolment beim Kartenherausgeber

Der initiale Schritt des Verfahrens besteht aus dem Enrolment. Hierbei muss sich der Benutzer $\mathcal{H}$ einmalig gegenüber dem Kartenherausgeber $\mathcal{I}$ ausweisen und damit seine (menschliche) Identität bestätigen. Anschließend wird an einem Sensor das biometrische Merkmal abgenommen und als initiale Abnahme b abgespeichert. Aus diesem b wird der biometrische Schlüssel x abgeleitet. Aus x und γ berechnet der Kartenherausgeber den privaten Schlüssel (A, x) für das Gruppensignatur-Schema. Sowohl b als auch A werden auf der Smart Card gespeichert und dem Benutzer ausgehändigt. Zusätzlich wird b noch in einer Datenbank des Kartenherausgebers gespeichert. Ab diesem Zeitpunkt kann sich der Benutzer gegenüber dem Dienstanbieter authentisieren.

Authentisierung gegenüber dem Dienstanbieter

Im Folgenden wird erst einmal eine leicht vereinfachte Variante zur Authentisierung gegenüber dem Dienstanbieter vorgestellt (siehe auch Abbildung 3.2), um so den Blick auf die wesentlichen Aktivitäten zu konzentrieren:

1. Der Dienstanbieter $\mathcal{P}$ sendet eine Zufallsnachricht M an den Sensor $\mathcal{S}$.

2. $\mathcal{S}$ erfasst eine neue Abnahme b' des biometrischen Merkmals von $\mathcal{H}$. Zusätzlich liest er von der Smart Card die Werte b und A aus.

3. $\mathcal{S}$ führt ein biometrisches Matching zwischen b' und b durch. Stimmen sie hinreichend überein, so berechnet $\mathcal{S}$ den Wert $x = H(b)$, also den aus b abgeleiteten biometrischen Schlüssel.

4. $\mathcal{S}$ berechnet mit Hilfe des privaten Schlüssels (A, x) des Gruppensignatur-Schemas die Signatur σ über die Nachricht M.

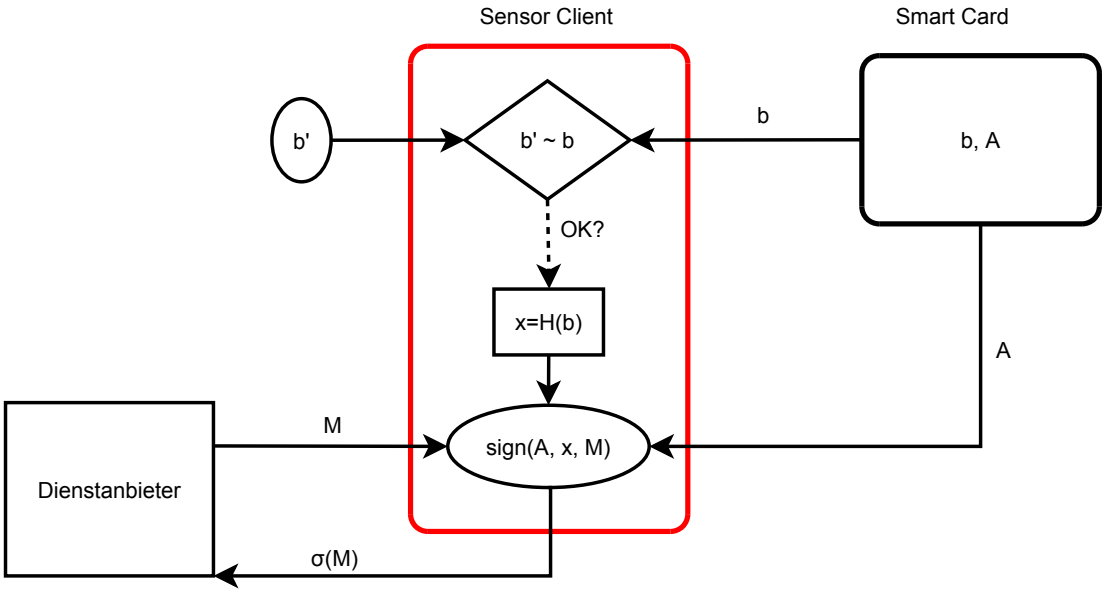

Abbildung 3.2.: Ablauf der biometrischen Authentisierung

5. $\mathcal{S}$ sendet die Gruppensignatur σ an den Dienstanbieter $\mathcal{P}$.
6. $\mathcal{P}$ überprüft die Gruppensignatur σ. Ist sie gültig, so ist die Authentisierung des Benutzers $\mathcal{H}$ erfolgreich.

Anforderungen an den Sensor

Dem Sensor $\mathcal{S}$ kommt eine zentrale Rolle zu. Es wird angenommen, dass nur *vertrauenswürdige* Sensoren (trusted Sensors) zum Einsatz kommen, das heißt, die Sensoren halten sich strikt an den vorgeschriebenen Ablauf und dürfen insbesondere keine der geheimen Daten des Benutzers speichern oder weitergeben.

Zusätzlich zu den angegebenen Berechnungen muss der Sensor auch sicherstellen, dass das biometrische Merkmal von einer lebenden Person abgegeben wird, die sogenannte Lebenderkennung (liveness detection). Ohne die Lebenderkennung müsste ein Angreifer nur eine Kopie des biometrischen Merkmals erstellen (zum Beispiel ein nachgemachter Fingerabdruck) und in den Besitz der Smart Card gelangen, schon könnte er die Identität des Benutzers annehmen. Die Verfahren zur Lebenderkennung sind nicht Thema dieser Arbeit, bei Interesse sei als Einstieg auf [JM03] verwiesen.

Als Beweis für die Vertrauenswürdigkeit des Sensors wird das Protokoll dahingehend abgeändert, dass der Sensor die Gruppensignatur σ zusätzlich mit einem digitalen Signaturverfahren signiert und damit gegenüber dem Dienstanbieter bestätigt, dass die Gruppensignatur von genau diesem Sensor erstellt wurde.

Sonderbehandlung bei Rechtsstreitigkeiten

Kommt es zu Rechtsstreitigkeiten oder sonstigen Unstimmigkeiten, so ist es wichtig, dass auch die frische Abnahme des biometrischen Merkmals b' als Beweismittel herangezogen werden kann. Das b' ist letztlich die einzige echte Information, die man von

der tatsächlichen Identität des Benutzer zum Zeitpunkt der Authentisierung hat. Alle anderen Daten beziehen sich letztlich nur auf die auf der Smart Card gespeicherten Daten, und die stammen vom Zeitpunkt des Enrolments. Das b' darf allerdings nicht dem Dienstanbieter zugänglich gemacht werden, denn der soll ja nichts über die Identität des Benutzers erfahren, sondern nur dem Kartenherausgeber, der für die Verknüpfung zwischen echter und biometrischer Identität verantwortlich ist.

Aus diesem Grund wird das Verfahren dahingehend erweitert, dass der Sensor den Wert b' mit dem öffentlichen Schlüssel Λ des Kartenherausgebers verschlüsselt und ihn mit an den Dienstanbieter $\mathcal{P}$ überträgt. So kann der Dienstanbieter den Wert b' zwar nicht entschlüsseln, aber bei Rechtsstreitigkeiten an den Kartenherausgeber $\mathcal{I}$ weiterleiten. Nur $\mathcal{I}$ kann den Wert entschlüsseln und anhand von b' in seiner Datenbank ermitteln, ob dort ein zugehöriges b existiert und damit analysieren, ob die Authentisierung berechtigt durchgeführt wurde oder nicht. Falls solch eine rechtliche Absicherung nicht gefordert ist, so kann auf die zentrale Speicherung von b in der Datenbank verzichtet werden.

Als Ergebnis entsteht folgende Variante des Verfahrens, die exakt so in [BCPZ08] vorgestellt wurde:

1. – 4. werden unverändert von der einfachen Variante übernommen.

5. $\mathcal{S}$ verschlüsselt b' mit einem sicheren Verschlüsselungsverfahren mit Hilfe des öffentlichen Schlüssels Λ des Kartenherausgebers. Mit einem digitalen Signaturverfahren erstellt $\mathcal{S}$ eine digitale Signatur Σ über den verschlüsselten Wert von b' und die Gruppensignatur σ. Anschließend sendet $\mathcal{S}$ die Gruppensignatur σ, die Verschlüsselung von b' und die Signatur Σ an den Dienstanbieter $\mathcal{P}$.

6. $\mathcal{P}$ überprüft zuerst die Signatur Σ. Damit stellt er sicher, dass sowohl σ als auch die Verschlüsselung von b' von einem vertrauenswürdigen Sensor erstellt wurden. Anschließend überprüft $\mathcal{P}$ die Gruppensignatur σ. Ist sie gültig, so ist die Authentisierung des Benutzers $\mathcal{H}$ erfolgreich. Der verschlüsselte Wert von b' wird im Normalfall nicht benötigt, nur im Ausnahmefall wie zum Beispiel bei Rechtsstreitigkeiten wird er an $\mathcal{I}$ weitergeleitet.

Widerruf

Ein besonderes Merkmal der Gruppensignatur aus Abschnitt 3.4.1 ist der Einsatz einer Widerrufs-Liste, mit der der Verifizierer überprüfen kann, ob eine Signatur von einem inzwischen ungültigen Schlüssel stammt. Der Verifizierer ist in diesem Fall der Dienstanbieter $\mathcal{P}$, demzufolge muss er im Besitz der Widerrufs-Liste sein. Verliert ein Benutzer seine Smart Card, so muss er dies beim Kartenherausgeber melden, woraufhin dieser den entsprechenden Teil des geheimen Schlüssels A vom Benutzer zur Widerrufs-Liste hinzufügt und an $\mathcal{P}$ weiterleitet, damit dieser wieder eine aktuelle Widerrufs-Liste hat. Gleichzeitig durchläuft der Benutzer noch einmal die Phase des Enrolment, damit er anschließend wieder eine (neue) gültige Identität im System besitzt.

Rolle der Smart Card

Da der Sensor vollständigen Zugriff auf die sicherheitsrelevanten Daten (b, A) auf der Smart Card benötigt, ist es absolut unerlässlich, dass der Sensor ein vertrauenswürdi-

ger Sensor ist und nach Erzeugung der Signatur die sicherheitsrelevanten Daten b, b', A wieder löscht. Es darf keinerlei Information über b, b', A nach außen dringen. Für Smart Cards existieren standardisierte Verfahren zur gegenseitigen Authentisierung (Mutal Authentication) zwischen Smart Card und Terminal, die die Vertrauenswürdigkeit zum Beispiel anhand der gemeinsamen Kenntnis eines geheimen Schlüssels überprüfen [RE99, Kap. 4.10]. Normalerweise wird dabei auch gleichzeitig ein Sitzungsschlüssel (Session Key) ausgehandelt, mit dem anschließend die gesamte Kommunikation zwischen Smart Card und Sensor verschlüsselt wird, um sich so gegen ein Abhören der Kommunikation zu schützen.

3.5.3. Varianten des Schemas

Für das bisher vorgestellte Verfahren waren die Anforderungen an die Smart Card sehr gering, sie diente nur als sicherer Speicher. Damit lässt sich das Schema schon mit sehr günstigen Karten umsetzen. Trotz der Versuche, die Vertrauenswürdigkeit des Sensors durch kryptografische Verfahren sicherzustellen, bleiben doch gewisse Restrisiken. Selbst wenn nach außen hin alles korrekt aussieht, ist doch letztlich nicht garantiert, dass der Sensor die sicherheitsrelevanten Daten sofort nach Benutzung wieder löscht. Ein Angreifer könnte den Sensor dahingehend manipuliert haben, dass die Daten gespeichert oder direkt an ihn übertragen werden, während alle anderen Berechnungen wie vorgeschrieben durchgeführt werden. Könnte man es schaffen, den privaten Schlüssel des Gruppensignatur-Schemas – nach dem initialen Einbringen beim Enrolment – dauerhaft ausschließlich innerhalb der sicheren Umgebung der Smart Card zu belassen, so würde das die Sicherheit des Verfahrens automatisch erhöhen.

Im Rahmen des TURBINE-Projektes wurden daher verschiedene Varianten des Schemas zur Diskussion gestellt (vgl. [TUR09, Kap. 4.8]), bei denen bestimmte Berechnungen vom Sensor auf die Smart Card verlagert werden.

Variante 1: Berechnung der Gruppensignatur auf der Smart Card

Bei dieser Variante wird Schritt 4 – die Berechnung der Gruppensignatur – vom Sensor in die Smart Card verlagert (siehe Abbildung 3.3). Das biometrische Matching zwischen b und b' geschieht nach wie vor im Sensor, daher benötigt $\mathcal{S}$ immer noch Zugriff auf b. Dadurch kann $\mathcal{S}$ zwar nach wie vor x aus b ableiten, aber der Wert A, der zweite Teil des privaten Schlüssels des Gruppensignatur-Schemas, verlässt die Karte nicht mehr. Selbst wenn der Sensor manipuliert würde, könnte ein Angreifer mit den abgehörten Daten keine gültige Gruppensignatur erzeugen, denn ohne A ist das nicht möglich (Zur Erinnerung, A lässt sich nur mit dem geheimen Master-Schlüssel γ des Kartenherausgebers aus x erzeugen).

Die Berechnung der Gruppensignatur auf der Smart Card erhöht also die Sicherheit des gesamten Verfahrens. Andererseits stellt sie erhöhte Anforderungen an die Smart Card, denn für die Erzeugung einer Gruppensignatur müssen diverse anspruchsvolle Berechnungen durchgeführt werden.

Diese Variante der biometrischen Authentisierung ist die Basis für den praktischen Teil dieser Diplomarbeit, einer Machbarkeitsstudie zur Berechnung der Gruppensignatur auf einer vorgegebenen Smart Card.

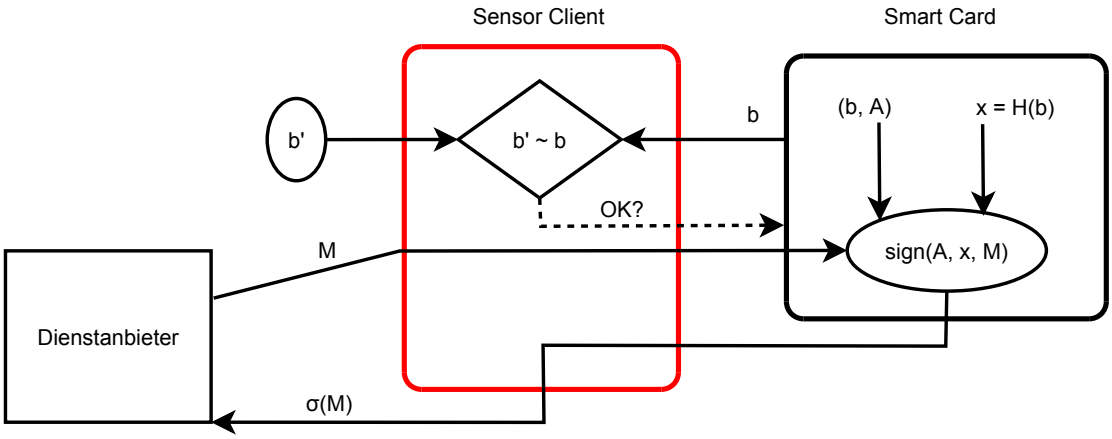

Abbildung 3.3.: Berechnung der Gruppensignatur auf der Smart Card

Variante 2: Biometrisches Matching und Gruppensignatur auf der Smart Card

Bei Variante 2 wird neben der Gruppensignatur auch das biometrische Matching auf der Smart Card durchgeführt, also der Vergleich zwischen b' und b. Dieser Teil wird als *Match on Card* bezeichnet. Als Aufgabe für den Sensor verbleibt nur noch, die neue Abnahme des biometrischen Merkmals b' durchzuführen und zusammen mit der Nachricht M an die Smart Card zu übertragen. Die Smart Card vergleicht b' und b, sind sie genügend ähnlich, so berechnet sie die Gruppensignatur σ. Diese wird an den Sensor übertragen und von dort zum Dienstanbieter. Keine der sicherheitskritischen Daten (b, A, x) verlassen mehr die Smart Card, im Vergleich zu Variante 1 ist das ein weiterer Sicherheitsgewinn.

Variante 3: Smart Card mit integriertem Sensor

Es existieren Smart Cards mit integriertem biometrischen Sensor. Für solche Karten ist eine Variante vorstellbar, bei der sowohl die Abnahme des biometrischen Merkmals, das biometrische Matching als auch die Berechnung der Gruppensignatur komplett von der Smart Card durchgeführt werden. Der bisherige Sensor Client $\mathcal{S}$ wäre in diesem Fall nur noch ein Vermittler zwischen dem Dienstanbieter und der Smart Card.

3.6. Secure-Sketch und Fuzzy-Extractor

Dodis et al. stellen in [DORS08] Methoden vor, um aus biometrischen Daten einen privaten Schlüssel zu erzeugen, der anschließend zum Beispiel für ein Authentisierungsverfahren entsprechend Abschnitt 3.3.1 oder für ein asymmetrisches Verschlüsselungsverfahren genutzt werden kann. Weder der private Schlüssel noch biometrische Referenzdaten werden in irgendeiner Form abgespeichert, der Schlüssel kann einzig durch eine erneute Abnahme des biometrischen Merkmals zurückgewonnen werden.

Damit dieses möglich ist, müssen zwei Probleme gelöst werden: Biometrische Daten sind im Allgemeinen weder exakt reproduzierbar noch entsprechen sie einer Gleichverteilung. Offensichtlich benötigt man eine exakte Reproduzierbarkeit, um immer wieder denselben Schlüssel zu erhalten. Ohne dies könnte man verschlüsselte Daten nicht wieder entschlüsseln beziehungsweise gültige Signaturen erstellen.

Auch die Gleichverteilung ist wichtig, denn die Sicherheit der kryptografischen Verfahren beruht darauf, dass jeder Schlüssel mit gleicher Wahrscheinlichkeit vorkommen kann. Ist dies nicht mehr gegeben, so ist die Sicherheit des gesamten Verfahrens gefährdet, da ein Angreifer Schlüssel auf einmal mit höherer Wahrscheinlichkeit „erraten“ könnte, wenn gewisse Schlüssel häufiger vorkommen als andere.

3.6.1. Secure-Sketch

Um die Reproduzierbarkeit zu gewährleisten, führen die Autoren den *Secure-Sketch* ein. Gegeben sei eine Menge $\mathcal{M}$ derart, dass sich ein biometrisches Merkmal als ein $w \in \mathcal{M}$ darstellen lässt. Zusätzlich sei eine Abstandsfunktion $d : \mathcal{M} \times \mathcal{M} \to \mathbb{Z}$ definiert. Interessant sind in erster Linie beliebige Bitfolgen ($\mathcal{M} = \{0,1\}^*$) und als Abstandsfunktion der Hamming-Abstand zweier Elemente, der die Anzahl der unterschiedlichen Bits angibt.

Ein Secure-Sketch erlaubt es, aus einem $w' \in \mathcal{M}$ immer wieder $w \in \mathcal{M}$ zu reproduzieren, solange sich w' nur „geringfügig“ von w unterscheidet.

Ein $(m, \tilde{m}, t)$-Secure-Sketch besteht aus einer Sketching-Funktion SS und einer Rückgewinnungsfunktion Rec. SS erzeugt aus einem $w \in \mathcal{M}$ einen Sketch $s \in \{0,1\}^*$. Mit diesem Sketch s und einem $w' \in \mathcal{M}$ erzeugt die Rückgewinnungsfunktion Rec wieder das ursprüngliche w, falls $d(w, w') \leq t$ ist, sich w und w' also nur geringfügig unterscheiden. Dieses muss für alle $w' \in \mathcal{M}$ gelten, für die $d(w, w') \leq t$ ist, nicht nur für einige wenige.

Der Sketch s ist nach der Erzeugung öffentlich bekannt, daher muss ein Secure-Sketch gewährleisten, dass von einem zufälligen w nur wenig von der Zufälligkeit verloren geht, selbst wenn man einen zugehörigen Sketch s kennt. Man darf also höchstens einige wenige Bits von w direkt aus s rekonstruieren können.

Dieses lässt sich auch formaler ausdrücken: $H(A)$ bezeichnet die Entropie für eine Zufallsvariable A, dies stellt ein Maß für die enthaltene Zufälligkeit dar. $H_\infty(A)$ bezeichnet die *min-Entropie*, eine Art worst-case Entropie über alle möglichen Zufallsvariablen A und $\tilde{H}_\infty(A|B)$ bezeichnet die *bedingte min-Entropie*, ein Maß für die Zufälligkeit von A unter Kenntnis einer weiteren Zufallsvariable B (für exakte Definitionen siehe [DORS08, S. 104f]). Für einen Secure-Sketch muss dann folgendes gelten:
Für alle Zufallsvariablen W über $\mathcal{M}$ mit $H_\infty(W) \geq m$ ist $\tilde{H}_\infty(W|SS(W)) \geq m'$.

3.6.2. Fuzzy-Extractor

Ein Secure-Sketch stellt zwar die Reproduzierbarkeit von biometrischen Daten sicher, aber er adressiert nicht das Problem der fehlenden Gleichverteilung der biometrischen Daten. Daher führen die Autoren den *Fuzzy-Extractor* ein. Ein Fuzzy-Extractor besteht aus den zwei Funktionen Gen und Rep („generate“ und „reproduce“). Gen erzeugt aus einem $w \in \mathcal{M}$ die Ausgabe $R \in \{0,1\}^l$ und die Hilfsdaten $P \in \{0,1\}^*$. Für zufällige w soll R wie zufällig gleichverteilt sein.

Rep erhält als Eingabe $w' \in \mathcal{M}$ und $P \in \{0,1\}^*$ und erzeugt als Ausgabe dasselbe $R \in \{0,1\}^l$ wie $Gen(w)$, falls $d(w, w') \leq t$ ist.

Ein Fuzzy-Extractor lässt sich grundsätzlich aus einem Secure-Sketch und einer universellen Hashfunktion Ext konstruieren. Die Konstruktion ist in Abbildung 3.4 dargestellt und sieht wie folgt aus:
Gen erzeugt aus einem $w \in \mathcal{M}$ den Sketch $s = SS(w)$. Mit dem Sketch und einer Zu-

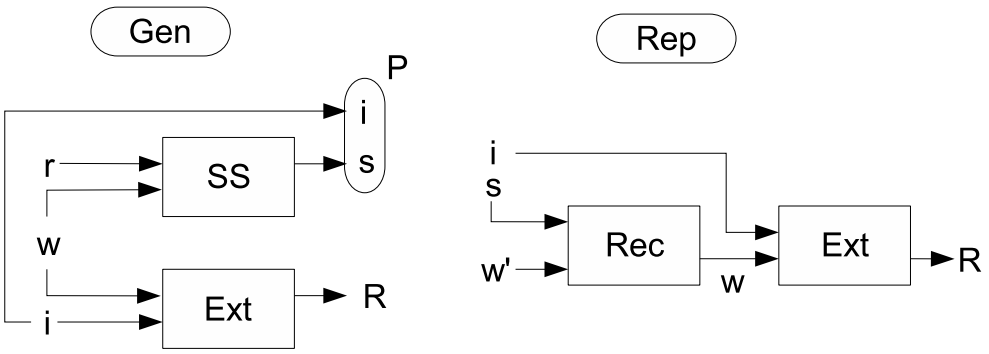

Abbildung 3.4.: Fuzzy-Extractor

fallszahl i wird der öffentliche Hilfswert $P = (s, i)$ des Fuzzy-Extractors gebildet. Aus w und i wird mit der Hashfunktion Ext die Ausgabe $R = Ext(w, i)$ erzeugt.

Die Funktion $Rep(w', P) = Rep(w', (s, i))$ ruft als erstes die Rückgewinnungsfunktion des Secure-Sketch auf, um aus w' und s das ursprüngliche $w = Rec(w', s)$ zu gewinnen. Nun wird ebenfalls die Hashfunktion aufgerufen, um $R = Ext(w, i)$ zu berechnen. Da die Hashfunktion dieselbe Eingabe erhält wie zuvor bei Gen, wird auch dasselbe R erzeugt.

Der Secure-Sketch sorgt also für die Reproduzierbarkeit, während die Hashfunktion sicherstellt, dass zum einen die erzeugten Zahlen annähernd gleichverteilt sind (dieses ist ein Merkmal von universellen Hashfunktionen) und zum anderen nicht allzu viel der in w enthaltenen Zufälligkeit verloren geht.

Bezug zu TURBINE

Verknüpft man den Fuzzy-Extractor mit einem Authentisierungsverfahren basierend auf einer digitalen Signatur entsprechend Abschnitt 3.3.1, so bilden sie zusammen ein Verfahren für eine biometrische Authentisierung. Die öffentlichen Werte $P = (s, i)$ fungieren als Pseudo-Identität, während das vom Fuzzy-Extractor erzeugte R als privater Schlüssel für das Signaturverfahren dient. Nur mit dem korrekten Fingerabdruck kann wieder das originale R erzeugt werden, und nur mit dem korrekten R lassen sich digitale Signaturen erzeugen, die mit dem zugehörigen öffentlichen Schlüssel vom Dienstanbieter verifiziert werden können (vgl. Kap. 2).

3.6.3. Konstruktion eines Secure-Sketches

Eine Möglichkeit für eine konkrete Umsetzung von Secure-Sketches besteht im Einsatz von fehlerkorrigierenden Codes. Diese stammen aus dem Bereich der Codierungstheorie und sind insbesondere bei der Datenübertragung sehr gut erforscht. Mit ihnen können Fehler, die während einer Übertragung von Datenpaketen auftreten können, direkt beim Empfänger erkannt und auch korrigiert werden, ohne das Paket noch einmal neu übertragen zu müssen.

Um Fehler korrigieren zu können, wird der originalen Nachricht zusätzliche Information hinzugefügt, die sogenannte *Redundanz*. Bei einem (n, k, d)-Code über dem Alphabet

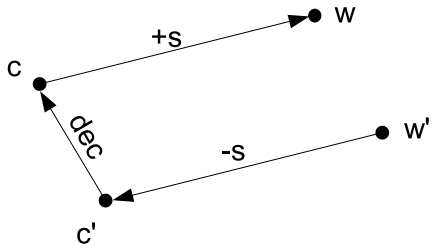

Abbildung 3.5.: Code-Offset Konstruktion

$\mathcal{F}$ der Größe F haben die Codewörter Länge n, die originale Nachricht hat Länge k, und der minimale Hamming-Abstand zwischen zwei Codewörtern ist d. Mit diesem Code ist es möglich bis zu $\lfloor \frac{d-1}{2} \rfloor$ viele Fehler zu korrigieren. Das heißt, selbst wenn sich das empfangene Codewort an $\lfloor \frac{d-1}{2} \rfloor$ Stellen von dem gesendeten unterscheidet, ist es trotzdem noch möglich daraus die originale Nachricht zu ermitteln.

Als Beispiel betrachte man einen sehr einfachen Code, der jedes Bit der Originalnachricht dreimal hintereinander sendet, aus '0' würde '000'. Der Empfänger würde jetzt diejenigen 3-Bit-Folgen wieder zu '0' dekodieren, bei denen mindestens zwei der drei Bits '0' sind. Durch diesen Code kann also pro drei empfangener Bits genau ein Fehler korrigiert werden. Dieses ist natürlich kein guter Code, da die Redundanz sehr groß ist, die Fehlerkorrekturfähigkeit aber nur gering, aber er zeigt den prinzipiellen Ablauf.

Inwiefern hilft der fehlerkorrigierende Code jetzt bei der Konstruktion eines Secure-Sketches? Sei C ein $(n, k, 2t+1)$-Code, er korrigiert also maximal t Fehler. C soll benutzt werden, um Fehler in w zu korrigieren, selbst wenn w kein Element des Codes ist. Dazu wird w auf ein beliebiges Codewort verschoben und diese Verschiebung als sketch s gespeichert.

Code-Offset Konstruktion Zu einer Eingabe w wähle zufällig gleichverteilt ein Codewort $c \in C$, setze $SS(w) = w - c$, die Verschiebung um von w zu c zu gelangen.

$Rec(w', s)$ subtrahiert als erstes die Verschiebung s von w', um damit ein Codewort $c' = w' - s$ zu erhalten. Anschließend wird c' zu c dekodiert. Dies funktioniert, wenn $d(w, w') \leq t$ ist, denn dann ist auch $d(c, c') \leq t$ und enthält somit höchstens t Fehler, die von C korrigiert werden können.

Schlussendlich erhält man w, indem die Verschiebung s wieder zu c addiert wird, $w = c + s$. Die Operationen erfolgen über dem Alphabet $\mathcal{F}$ beziehungsweise $\mathbb{Z}_F$, bei Bitfolgen entspricht dies gerade $\mathbb{Z}_2$.

4. Konzept zur Umsetzung der Gruppensignatur

Um das Gruppensignatur-Schema umsetzen zu können, müssen die Systemparameter des Schemas mit konkreten Werten belegt werden. Kernelement der Gruppensignatur ist eine bilineare Abbildung. Nach aktuellem Forschungsstand sind überhaupt nur wenige effizient berechenbare bilineare Abbildungen bekannt, mit deren Hilfe man bilineare Gruppen konstruieren kann. Eine der am besten erforschten ist das sogenannte *Weil-Pairing* $e(P,Q)$, das auf einer Untergruppe einer elliptischen Kurve E und einer Untergruppe des Erweiterungskörpers $\mathbb{F}^*_{q^2}$ definiert ist. Um das Weil-Pairing berechnen zu können, benötigt man Arithmetik in $\mathbb{F}_q$ und $\mathbb{F}_{q^2}$ sowie Arithmetik über den elliptischen Kurven $E(\mathbb{F}_q)$ und $E(\mathbb{F}_{q^2})$.

Ein weiterer wichtiger Bestandteil des Gruppensignatur-Schemas ist die Hashfunktion H_0. Wie sich später zeigen wird, stellt auch ihre Konstruktion eine gewisse Herausforderung dar.

In den folgenden Abschnitten werden Schritt für Schritt sämtliche Elemente beschrieben, die zur Berechnung des Weil-Pairings $e(P,Q)$ beziehungsweise des modifizierten Weil-Pairings $\hat{e}(P,Q)$ sowie der Hashfunktion H_0 benötigt werden. Mit ihnen hat man alle Bausteine zusammen, die für die konkrete Umsetzung des Gruppensignatur-Schemas benötigt werden.

4.1. Endliche Körper und ihre Arithmetik

In den folgenden Abschnitten werden die endlichen Körper $\mathbb{F}_q$ und $\mathbb{F}_{q^2}$ beschrieben. Es soll zumindest soviel Wissen vermittelt werden, um die spätere Konstruktion für die Gruppensignatur nachvollziehen zu können. Weitergehende Informationen zu endlichen Körpern finden sich zum Beispiel in [MOV96], [HMV04] und [HPS08].

4.1.1. Basiskörper $\mathbb{F}_q$

Sei q eine Primzahl. Die Berechnungen in einem endlichen Körper $\mathbb{F}_q$ sind ein Fundament der modernen Kryptografie und werden auch als modulare Arithmetik bezeichnet. Der endliche Körper $\mathbb{F}_q$ ist isomorph zum Körper $\mathbb{Z}_q$ und besteht daher aus den Elementen $\{0,\ldots,q-1\}$. Als Operationen werden die Addition und Multiplikation modulo q benutzt. Subtraktion und Division ergeben sich als Addition beziehungsweise Multiplikation mit dem inversen Element.

Die multiplikative Gruppe $\mathbb{F}^*_q$ besteht aus der Menge $\mathbb{F}_q \setminus \{0\}$ und ist eine zyklische Gruppe mit der Ordnung $q-1$. In $\mathbb{F}^*_q$ existiert zu jedem Element ein inverses Element, das sich zum Beispiel mit dem erweiterten Euklidischen Algorithmus berechnen lässt. Die 0 ist das neutrale Element der Addition, die 1 das neutrale Element der Multiplikation.

Die Ordnung eines Elementes $g \in \mathbb{F}_q^*$ ist entsprechend Abschnitt 3.1 definiert als das kleinste $k > 0$, für das $g^k = 1$ ist. Die von g erzeugte Untergruppe $G = \langle g \rangle \subseteq \mathbb{F}_q^*$ hat dieselbe Ordnung wie g, und g ist ein Generator von G.

Wie schon in Abschnitt 3.1 angedeutet wurde, benötigen einige kryptografische Verfahren genau solche zyklischen Untergruppen einer bestimmten Ordnung. Sei g ein Generator von $\mathbb{F}_q^*$ und $m \in \mathbb{F}_q^*$, so ist der diskrete Logarithmus $\log_g(m) = x$ die Lösung der Kongruenz $g^x \equiv m \mod q$. Zur Erinnerung, das Diskreter-Logarithmus-Problem besteht darin, zu gegebenem Generator $g \in \mathbb{F}_q^*$ und einem Element $m \in \mathbb{F}_q^*$ die Lösung x zu finden mit x = $\log_g(m)$.

Für dieses Problem existieren in $\mathbb{F}_q^*$ keine schnellen Algorithmen. Es existieren einige generische Algorithmen zum Lösen des diskreten Logarithmus, die eine Laufzeit von $O(\sqrt{q})$ haben, also exponentielle Laufzeit. Der *Index-Calculus-Algorithmus* ist der schnellste bekannte Algorithmus zur Berechnung des diskreten Logarithmus und hat subexponentielle Laufzeit. Wählt man allerdings für q eine Bitlänge von 1024 oder mehr, so ist auch mit diesem Algorithmus der diskrete Logarithmus nicht mehr effizient berechenbar. Daher wird das Diskreter-Logarithmus-Problem als Basis für einige kryptografische Verfahren benutzt. Weitere Details zu endlichen Körpern und dem Diskreter-Logarithmus-Problem finden sich in [MOV96].

4.1.2. Erweiterungskörper $\mathbb{F}_{q^2}$

Sei K_2 ein Körper und $K_1 \subset K_2$. Falls K_1 ebenfalls ein Körper für die auf K_2 definierten Operationen ist, so ist K_1 ein Unterkörper von K_2 und umgekehrt ist K_2 ein Erweiterungskörper von K_1, denn K_2 enthält K_1 und zusätzlich noch weitere Elemente.

Allgemein konstruiert man einen Körper $\mathbb{F}_{q^d}$, indem man Polynome mit Koeffizienten aus $\mathbb{F}_q$ betrachtet, geschrieben $\mathbb{F}_q[X]$. Zusätzlich benötigt man ein über $\mathbb{F}_q$ irreduzibles Polynom f vom Grad d. Dann ist die Menge $\mathbb{F}_q[X]$ modulo des irredubziblen Polynoms f zusammen mit der Polynomaddition und -multiplikation modulo f ein Körper, bezeichnet mit $\mathbb{F}_q[X]/f$. Da die Polynome maximal Grad $d-1$ haben, hat der Körper q^d Elemente (siehe [HPS08, 2.10.4]).

Für $d = 2$ wird daher die folgende Konstruktion gewählt: Das Polynom $f = X^2 + 1$ hat Grad 2 und ist irreduzibel über $\mathbb{F}_q$, wenn $q = 3 \mod 4$ gewählt wird. Dies lässt sich wie folgt beweisen:
Angenommen f wäre nicht irreduzibel, so ließe sich f darstellen als $f = (x+a)(x+b) = x^2 + (a+b)x + ab$. Durch Koeffizientenvergleich ergibt sich $a + b \equiv 0 \mod q$ und $ab \equiv 1 \mod q$, zusammen $a^2 \equiv -1 \mod q$. Gesucht ist also ein Element a, welches quadriert den Rest $-1 \mod q$ hat. Dies entspricht genau der Frage, ob -1 ein quadratischer Rest modulo q ist. Diese Frage kann mit Hilfe des Legendre-Symbols entschieden werden [MOV96, 2.145]: Für $q = 3 \mod 4$ ist das Legendre-Symbol $\left(\frac{-1}{q}\right) \equiv (-1)^{(q-1)/2} \equiv -1 \mod q$, da $(q-1)/2$ ungerade ist. Nach Definition des Legendre-Symbols ist -1 somit kein quadratischer Rest modulo q, das heißt es existiert kein $a \in \mathbb{F}_q$ mit $a^2 \equiv -1 \mod q$, folglich war die Annahme falsch und $f = X^2 + 1$ ist somit irreduzibel über $\mathbb{F}_q$ für $q = 3 \mod 4$.

Daher wählt man $q = 3 \mod 4$, damit ist $\mathbb{F}_{q^2}$ isomorph zum Körper $\mathbb{F}_q[X]/(X^2+1)$ (siehe [HPS08, Seite 103]).

Im Folgenden wird für $\mathbb{F}_{q^2}$ nicht die Polynomschreibweise $a + bX \mod f$ gewählt,

stattdessen setzt man i als „Quadratwurzel“ von -1. Elemente von $\mathbb{F}_{q^2}$ lassen sich daher schreiben als $(a, b) \in \mathbb{F}_{q^2} : a + ib$, mit $a, b \in \mathbb{F}_q$ und $i^2 = -1$. So wird auch ersichtlich, dass $\mathbb{F}_q \subset \mathbb{F}_{q^2}$ ist, denn die Menge aller Elemente, für die $b = 0$ ist, entspricht gerade $\mathbb{F}_q$.

Die multiplikative Gruppe $\mathbb{F}_{q^2}^*$ hat q^2-1 Elemente. Sei p prim und ein Teiler von q^2-1, so enthält $\mathbb{F}_{q^2}^*$ eine Untergruppe der Ordnung p. Bezeichne diese Untergruppe mit $\mathbb{G}_T$. Für jedes Element $g \in \mathbb{G}_T$ ist $g^p = 1$, $\mathbb{G}_T$ enthält also gerade die p-ten Einheitswurzeln. Genau in diese Untergruppe $\mathbb{G}_T$ hinein wird unsere bilineare Abbildungsfunktion $\hat{e}$ später abbilden.

4.1.3. Arithmetik im Erweiterungskörper $\mathbb{F}_{q^2}$

Die grundlegenden arithmetischen Operationen in $\mathbb{F}_{q^2}$ können auf arithmetische Operationen im Körper $\mathbb{F}_q$ zurückgeführt werden [DhSD06, Kap. 3]. Die Addition in $\mathbb{F}_{q^2}$ benötigt nach der Formel

$$u + v = (a, b) + (c, d) = (a + ib) + (c + id) = (a + c) + i(b + d)$$

zwei Additionen in $\mathbb{F}_q$. Analog dazu benötigt die Subtraktion in $\mathbb{F}_{q^2}$ zwei Subtraktionen in $\mathbb{F}_q$:

$$u - v = (a, b) - (c, d) = (a + ib) - (c + id) = (a - c) + i(b - d)$$

Die Multiplikation sieht wie folgt aus:

$$u \cdot v = (a, b) \cdot (c, d) = (a + ib) \cdot (c + id) = ac + iad + ibc + i^2bd = (ac - bd) + i(ad + bc)$$

Nach dem naiven Ansatz benötigt eine Multiplikation in $\mathbb{F}_{q^2}$ vier Multiplikationen in $\mathbb{F}_q$. Eine Multiplikation lässt sich allerdings noch einsparen:

$$(a + b) \cdot (c - d) + ad - bc = (ac - ad + bc - bd) + ad - bc = (ac - bd)$$

Da ad und bc schon für die 2. Komponente berechnet werden müssen, kann die 1. Komponente mit nur einer anstatt zwei weiteren Multiplikation berechnet werden.

Das Quadrat einer Zahl lässt sich natürlich mit der normalen Multiplikationsoperation berechnen, aber es geht noch etwas besser:

$$\begin{aligned} u^2 = (a, b)^2 = (a + ib)^2 = (a^2 + i \cdot 2ab + i^2b^2) = (a^2 + i \cdot 2ab - b^2) = (a^2 - b^2) + i(2ab) \\ = ((a + b)(a - b)) + i(2ab) \end{aligned}$$

Damit lässt sich das Quadrat einer Zahl in $\mathbb{F}_{q^2}$ im Wesentlichen mit nur zwei Multiplikationen in $\mathbb{F}_q$ berechnen.

Das Inverse eines Elements in $\mathbb{F}_{q^2}$ lässt sich mit nur einer Inversion in $\mathbb{F}_q$ sowie zwei Quadrierungen und zwei Multiplikationen berechnen:

$$u^{-1} = \frac{1}{(a, b)} = \frac{1}{(a + ib)} = \frac{a - ib}{(a + ib)(a - ib)} = \frac{a}{a^2 + b^2} - i\frac{b}{a^2 + b^2}$$

Damit lassen sich also alle grundlegenden Operationen in $\mathbb{F}_{q^2}$ mittels weniger Operationen in $\mathbb{F}_q$ effizient berechnen.

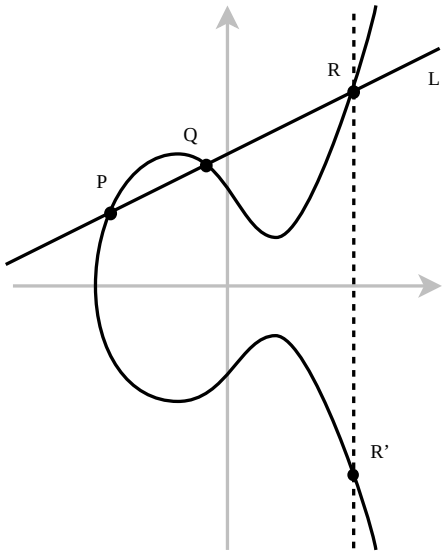

Abbildung 4.1.: Addition zweier Punkte P, Q auf E

4.2. Elliptische Kurven

Eine elliptische Kurve über dem Körper K ist definiert durch

$$E : y^2 = x^3 + ax + b, \text{ mit } a, b \in K \text{ und } 4a^3 + 27b^2 \neq 0.$$

Alle Punkte $P = (x, y)$, die die Kurvengleichung erfüllen, bilden zusammen mit dem *unendlich fernen Punkt* $\mathcal{O}$ die Elemente der elliptischen Kurve. Kombiniert mit einer speziellen Additionsfunktion zum Addieren zweier Punkte erhält man eine abelsche Gruppe. Die Bedingung $4a^3 + 27b^2 \neq 0$ ist nötig, damit die Additionsfunktion für jeden Punkt wohldefiniert ist.

Die Addition zweier Punkte lässt sich am einfachsten geometrisch veranschaulichen (siehe auch [HPS08, Kap. 5.1]).

- **Punktaddition:** Seien P, Q zwei Punkte auf der Kurve und sei L die Gerade durch P und Q. L schneidet die Kurve zusätzlich zu P und Q noch in einem dritten Punkt $R = (x_3, y_3)$. Als Ergebnis der Punktaddition von P und Q wird jetzt allerdings nicht der Punkt R definiert, sondern dessen Spiegelung an der x-Achse $R' = (x_3, -y_3)$.

- **Punktverdopplung:** Addiert man P zu sich selbst, so wird die Tangente im Punkt P berechnet. Diese hat wiederum einen weiteren Schnittpunkt R mit der Kurve, und dessen Spiegelung an der x-Achse R' wird als Ergebnis der Addition $P + P$ definiert.

- Addiert man P zu $-P$, so ist die resultierende Gerade durch P und $-P$ eine Vertikale ($-P$ ist die Spiegelung von P an der x-Achse). Auch bei der Punktverdopplung kann es passieren, dass die Tangente eine Vertikale ist. Diese Vertikale hat keinen weiteren Schnittpunkt mit der Kurve. Für diesen Fall wird der unendliche ferne Punkt $\mathcal{O}$ benötigt. Dieser ist definiert als Schnittpunkt aller Vertikalen

im Unendlichen und fungiert als neutrales Element der Punktaddition. Somit ist $P + (-P) = \mathcal{O}$, zusätzlich wird $P + \mathcal{O} = \mathcal{O} + P = P$ definiert, da $-P$ der dritte Punkt auf der Vertikalen durch P und $\mathcal{O}$ ist, die Spiegelung an der x-Achse ergibt dann wieder P.

Um mit der elliptischen Kurve tatsächlich rechnen zu können, benötigt man explizite Formeln zur Berechnung der Punktaddition. Diese lassen sich direkt aus der geometrischen Darstellung ableiten. Im Wesentlichen muss man die Steigung der Gerade beziehungsweise der Tangente berechnen, erhält damit die Geraden- beziehungsweise Tangentengleichung und kann diese mit der Kurvengleichung gleichsetzen. Daraus lassen sich die Koordinaten des weiteren Schnittpunktes ableiten. Die Formeln werden weiter unten angegeben, eine Herleitung findet sich in [HPS08, S. 285].

Während sich die geometrische Motivation auf elliptische Kurven über dem Körper $\mathbb{Q}$ bezieht, sind für die Kryptografie elliptische Kurven über endlichen Körpern von Interesse. Auch über endlichen Körpern bilden die Lösungen der Kurvengleichung zusammen mit dem unendlichen fernen Punkt $\mathcal{O}$ und der Punktaddition eine abelsche Gruppe. Für die Punktaddition können im Prinzip dieselben Formeln wie über dem Körper $\mathbb{Q}$ benutzt werden, angepasst natürlich auf die Operationen im endlichen Körper [HPS08, Kap. 5.2].

Über endlichen Körpern ist die Anzahl der Punkte einer elliptischen Kurve ebenfalls endlich. Zusätzlich haben die Koordinaten eines Punktes eine feste Länge. Dies beides sind wichtige Voraussetzungen, um hier exakte und effiziente Berechnungen durchführen zu können.

Sei P ein Punkt auf der Kurve, so wird mit $2P$ die Addition des Punktes P mit sich selbst bezeichnet, analog ist $kP = P + \ldots + P$ die k-fache Addition des Punktes P, bezeichnet als skalare Multiplikation oder Punktmultiplikation. Damit lässt sich auch die Ordnung eines Punktes P definieren, dieses ist das kleinste $k > 0$ für das gilt: $kP = \mathcal{O}$.

Ferner ist es möglich, zyklische Gruppen einer bestimmten Größe zu erzeugen. Die gesamte elliptische Kurve ist im allgemeinen keine zyklische Gruppe, allerdings erzeugt jeder Punkt der Kurve eine zyklische Untergruppe. Das Finden einer zyklischen Untergruppe mit einer bestimmten Ordnung reduziert sich somit auf das Problem, einen einzelnen Punkt zu finden, der die gewünschte Ordnung hat. Wie so etwas funktioniert, wird später in Abschnitt 4.4 noch deutlich werden, da für das Weil-Pairing eine spezielle Untergruppe benötigt wird.

Da auf elliptischen Kurven zyklische Gruppen existieren, kann auch hier ein diskreter Logarithmus definiert werden. Das Diskreter-Logarithmus-Problem auf elliptischen Kurven (ECDLP) besteht darin, zu gegebenen zwei Punkten P, Q mit $Q = kP$ die Zahl k zu berechnen. Für dieses Problem sind keine schnellen Algorithmen bekannt, insbesondere gibt es für den schnellsten bekannten Algorithmus zur Berechnung des diskreten Logarithmus über endlichen Körpern – dem Index-Calculus-Algorithmus – keine Entsprechung auf elliptischen Kurven. Dadurch reichen schon sehr kurze Bitlängen von etwa 160 Bit aus, um das gleiche Sicherheitsniveau zu erreichen wie bei Verfahren, die auf dem diskreten Logarithmus über endlichen Körpern basieren und aufgrund des Index-Calculus-Algorithmus Bitlängen von 1024 Bit und mehr erfordern. Daher bietet sich der Einsatz von auf elliptischen Kurven basierenden Verfahren insbesondere auf Smart Cards an, bei denen der Speicher und die Rechenleistung bekanntlich sehr begrenzt sind.

Unabhängig von den genannten Vorteilen werden elliptische Kurven für die Umsetzung der Gruppensignatur zwingend benötigt, da das Weil-Pairing gerade über ellipti-

schen Kurven definiert ist. Der folgende Abschnitt zeigt, wie Operationen auf elliptischen Kurven über einem endlichen Körper effizient berechnet werden können.

4.2.1. Arithmetik auf elliptischen Kurven über dem Körper $\mathbb{F}_q$

Die elementaren Operationen auf einer elliptischen Kurve sind die Addition von zwei Punkten und die Punktverdopplung. Ziel ist es, diese Operationen möglichst effizient berechnen zu können.

Eine elliptische Kurve lässt sich in unterschiedlichen Koordinatensystemen darstellen. Je nach verwendetem Koordinatensystem variiert der Aufwand zur Berechnung der Punktaddition und Punktverdopplung. [CMO98] gibt eine Übersicht über einige Koordinatensysteme, liefert die genauen Berechnungsvorschriften für die Punktaddition und Punktverdopplung im jeweiligen System und eine Aufwandsanalyse bezüglich der benötigten Operationen im zugrunde liegenden Körper. [HMV04, Kap. 3.2] enthält zusätzlich noch Algorithmen zur Punktaddition und Punktverdopplung. Eine ausführliche Untersuchung der Koordinatensysteme findet sich auch in [Hen04]. Im Folgenden werden die affinen, projektiven und Jacobischen Koordinaten betrachtet.

Bezüglich des Berechnungsaufwandes sind die Anzahl an Inversionen (I), Multiplikationen (M) und Quadrierungen (S) von Interesse. Additionen, Subtraktionen und Multiplikationen mit kleinen Konstanten (die normalerweise über mehrfache Additionen realisiert sind) werden in der Aufwandsbetrachung meist ignoriert, da ihr Aufwand im Verhältnis zur Multiplikation oder Inversion sehr gering ist.

Affine Koordinaten

Das affine Koordinatensystem ist das bekannteste und einfachste System. Ein Punkt besteht aus zwei Koordinaten (x, y). Die Kurvengleichung einer Kurve E über dem Körper $\mathbb{F}_q$ (bezeichnet mit $E(\mathbb{F}_q)$) lautet:

$$E : y^2 = x^3 + ax + b \text{ mit } a, b \in \mathbb{F}_q,\ 4a^3 + 27b^2 \neq 0.$$

Seien $P = (x_1, y_1), Q = (x_2, y_2)$ und $R = P+Q = (x_3, y_3)$ Punkte auf $E(\mathbb{F}_q)$. Die Formel für die **Punktaddition** ($P \neq \pm Q$) lautet:

$$\begin{aligned} \lambda &= \frac{y_2 - y_1}{x_2 - x_1} \\ x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned} \tag{4.1}$$

λ entspricht gerade der Steigung der Geraden durch die Punkte P und Q.

Ist $P = Q$, so entspricht die Addition der Punktverdopplung, ist $P = -Q$, so ist das Ergebnis der neutrale Punkt $\mathcal{O}$.
Die Formel für die **Punktverdopplung** ($P = Q, R = 2P$) lautet:

$$\begin{aligned} \lambda &= \frac{3x_1^2 + a}{2y_1} \\ x_3 &= \lambda^2 - 2x_1 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned} \tag{4.2}$$

Ist $y_1 = 0$, so ist das Ergebnis der neutrale Punkt $\mathcal{O}$(die Tangente wäre in diesem Fall eine Vertikale).

Der Berechnungsaufwand wird im Folgenden durch $t()$ symbolisiert, $\mathcal{A}$ bezeichnet die affinen Koordinaten. Der Aufwand für eine Punktaddition beträgt $t(\mathcal{A}+\mathcal{A}) = I+2M+S$ und für eine Punktverdopplung $t(2\mathcal{A}) = I + 2M + 2S$.

Projektive Koordinaten

Die projektive Geometrie erweitert die affine Geometrie um unendlich ferne Punkte. So schneiden sich in der projektiven Ebene zwei Geraden grundsätzlich in einem (möglicherweise unendlich fernen) Punkt. Ein Punkt wird nicht mehr mit zwei Koordinaten (X, Y) dargestellt, sondern er besteht aus drei Koordinaten (X, Y, Z).

Von besonderem Interesse ist folgende Eigenschaft: zwei Punkte (X, Y, Z),$(\lambda X, \lambda Y, \lambda Z)$ mit beliebigem $\lambda \neq 0$ sind zueinander äquivalent. Alle Punkte, die diese Äquivalenzrelation erfüllen, bilden eine Äquivalenzklasse [HMV04, Kap. 3.2.1]. Insbesondere ist für $Z \neq 0$ $(X/Z, Y/Z, 1)$ der einzige Repräsentant dieser Äquivalenzklasse mit einer Z-Koordinate von 1, somit hat die Äquivalenzklasse eine eindeutige Entsprechung im affinen System, nämlich gerade den Punkt $(X/Z, Y/Z)$.

Während der unendlich ferne Punkt im affinen System eine Art Sonderkonstrukt ist, sind unendlich ferne Punkte im projektiven System etwas „normales“, es sind einfach die Punkte mit einer Z-Koordinate von 0.

In Bezug auf die elliptische Kurvenarithmetik kann man durch die Darstellung in projektiven Koordinaten durch die Äquivalenzrelation die kostspieligen Inversionen einsparen. Treten bei Berechnungen Divisionen auf, so werden diese Divisionen nicht ausgeführt, stattdessen bildet man den Hauptnenner und verschiebt diesen sozusagen in die Z-Koordinate, wählt damit also einen anderen Repräsentanten für denselben projektiven Punkt, der ohne Bruchdarstellung auskommt. Dies spart Inversionen auf Kosten einiger zusätzlicher Multiplikationen. Einzig für die Rücktransformation in die affinen Koordinaten wird noch eine Inversion benötigt.

Beim projektiven Koordinatensystem setzt man $x = X/Z,\ y = Y/Z$ und erhält damit die Gleichung [1]:

$$E_P : Y^2Z = X^3 + aXZ^2 + bZ^3 \text{ mit } a, b \in \mathbb{F}_q,\ 4a^3 + 27b^2 \neq 0.$$

Für die Konvertierung vom affinen System in das projektive System setzt man $X = x,\ Y = y,\ Z = 1$. Für die Rücktransformation vom projektiven System ins affine System setzt man $x = X/Z,\ y = Y/Z$. Ein Repräsentant für den unendlich fernen Punkt ist $\mathcal{O} = (0, 1, 0)$ (beachte: $\mathcal{O}$ ist eine Lösung von E).

Seien $P = (X_1, Y_1, Z_1), Q = (X_2, Y_2, Z_2)$ und $R = P + Q = (X_3, Y_3, Z_3)$ Punkte auf $E_P(\mathbb{F}_q)$. Die folgenden Formeln ergeben sich aus den Formeln im affinen System, indem man dort entsprechend $x = X/Z$ und $y = Y/Z$ einsetzt. Die dabei entstehenden Brüche werden normiert und verändern entsprechend die Z-Koordinate.

[1] $E_P : \frac{Y^2}{Z^2} = \frac{X^3}{Z^3} + a\frac{X}{Z} + b \Leftrightarrow Y^2Z = X^3 + aXZ^2 + bZ^3$

Die Formel für die **Punktaddition** ($P \neq \pm Q$) lautet:

$$\begin{aligned} X_3 &= vA \\ Y_3 &= u(v^2 X_1 Z_2 - A) - v^3 Y_1 Z_2 \\ Z_3 &= v^3 Z_1 Z_2 \\ &\text{mit} \\ u &= Y_2 Z_1 - Y_1 Z_2 \\ v &= X_2 Z_1 - X_1 Z_2 \\ A &= u^2 Z_1 Z_2 - v^3 - 2v^2 X_1 Z_2 \end{aligned} \tag{4.3}$$

u und v entsprechen dem Zähler und Nenner der Steigung λ im affinen System.
Die Formel für die **Punktverdopplung** ($P = Q, R = 2P$) lautet:

$$\begin{aligned} X_3 &= 2hs \\ Y_3 &= w(4B - h) - 8Y_1^2 s^2 \\ Z_3 &= 8s^3 \\ &\text{mit} \\ w &= aZ_1^2 + 3X_1^2,\ s = Y_1 Z_1,\ B = X_1 Y_1 s,\ h = w^2 - 8B \end{aligned} \tag{4.4}$$

Der Aufwand für eine Punktaddition beträgt $t(\mathcal{P} + \mathcal{P}) = 12M + 2S$ und für eine Punktverdopplung $t(2\mathcal{P}) = 7M + 5S$. $\mathcal{P}$ bezeichnet dabei die projektiven Koordinaten.

Jacobische Koordinaten

Das Jacobische Koordinatensystem ist eine Variante des projektiven Koordinatensystems, bei dem der Übergang vom affinen System anders gewichtet ist.

Beim Jacobischen Koordinatensystem setzt man $x = X/Z^2,\ y = Y/Z^3$ und erhält damit die Gleichung [2]:

$$E_J : Y^2 = X^3 + aXZ^4 + bZ^6 \text{ mit } a, b \in \mathbb{F}_q,\ 4a^3 + 27b^2 \neq 0.$$

Für eine Konvertierung zwischen affinem System und Jacobischem System setzt man $X = x,\ Y = y,\ Z = 1$, von Jacobisch nach affin setzt man $x = X/Z^2,\ y = Y/Z^3$. Ein Repräsentant für den unendlich fernen Punkt ist $\mathcal{O} = (1, 1, 0)$.

Seien $P = (X_1, Y_1, Z_1), Q = (X_2, Y_2, Z_2)$ und $R = P + Q = (X_3, Y_3, Z_3)$ Punkte auf $E_J(\mathbb{F}_q)$. Die Formel für die **Punktaddition** ($P \neq \pm Q$) lautet:

$$\begin{aligned} X_3 &= -H^3 - 2U_1 H^2 + r^2 \\ Y_3 &= -S_1 H^3 + r(U_1 H^2 - X_3) \\ Z_3 &= Z_1 Z_2 H \\ &\text{mit} \\ U_1 &= X_1 Z_2^2,\ U_2 = X_2 Z_1^2,\ S_1 = Y_1 Z_2^3,\ S_2 = Y_2 Z_1^3, \\ H &= U_2 - U_1,\ r = S_2 - S_1 \end{aligned} \tag{4.5}$$

[2] $E_J : \frac{Y^2}{Z^6} = \frac{X^3}{Z^6} + a\frac{X}{Z^2} + b \Leftrightarrow Y^2 = X^3 + aXZ^4 + bZ^6$

Die Formel für die **Punktverdopplung** ($P = Q, R = 2P$) lautet:

$$\begin{aligned} X_3 &= T \\ Y_3 &= -8Y_1^4 + M(S - T) \\ Z_3 &= 2Y_1 Z_1 \\ \text{mit} & \\ S &= 4X_1 Y_1^2,\ M = 3X_1^2 + aZ_1^4,\ T = -2S + M^2 \end{aligned} \tag{4.6}$$

Der Aufwand für eine Punktaddition beträgt $t(\mathcal{J} + \mathcal{J}) = 12M + 4S$ und für eine Punktverdopplung $t(2\mathcal{J}) = 4M + 6S$. $\mathcal{J}$ bezeichnet dabei die Jacobischen Koordinaten.

Punktmultiplikation

Sei $Q \in E(\mathbb{F}_q)$ das Ergebnis einer k-fachen Addition eines Punktes $P \in E(\mathbb{F}_q)$, geschrieben $Q = k \cdot P$. Basierend auf der Punktaddition und Punktverdopplung lässt sich diese Punktmultiplikation mit Hilfe des Double-And-Add-Algorithmus (siehe Algorithmus 1) effizient berechnen. Dieser ist ähnlich aufgebaut wie der bekannte Square-And-Multiply-Algorithmus zur Exponentiation in endlichen Körpern, nur werden hier die Multiplikation und das Quadrieren durch die Punktaddition und die Punktverdopplung ersetzt. Der Algorithmus läuft in einer Schleife über die binäre Darstellung des Skalars k. Für jedes Bit erfolgt eine Punktverdopplung, für jedes Bit mit Wert 1 wird zusätzlich noch eine Punktaddition durchgeführt. So ist zum Beispiel die Binärdarstellung von $5 = (101)_2$, daher lässt sich $5P$ berechnen als $2(2(2(\mathcal{O}) + P)) + P = 2(2(P)) + P = 2(2P) + P = 4P + P = 5P$.

Ist m die Bitlänge des Skalars, so führt der Algorithmus m Punktverdopplungen und im Mittel etwa $\frac{m}{2}$ Punktadditionen aus, da bei einem zufälligen Skalar im Mittel etwa die Hälfte der Bits Eins ist.

Algorithmus 1: Double-And-Add-Algorithmus

Input : positive ganze Zahl k mit Bitlänge m, $P \in E(\mathbb{F}_q)$
Output : kP

$Q = \mathcal{O}$
for $i = m - 1$ **downto** 0 **do**
  $Q = 2 \cdot Q$
  if $k_i = 1$ **then** $Q = Q + P$
return Q

Optimierung Punktaddition: gemischte Addition

Für das projektive oder Jacobische Koordinatensystem können bei der Punktaddition einige Rechenschritte eingespart werden, wenn der zweite Punkt im affinen System angegeben wird. Im projektiven oder Jacobischen System ist die Z-Koordinate dieses Punktes 1, dadurch können alle Multiplikationen oder Quadrierungen in den Formeln 4.3 und 4.5 mit Z_2 eingespart werden. Dadurch ergeben sich Berechnungskosten von $t(\mathcal{P} = \mathcal{P} + \mathcal{A}) = 9M + 2S$ und $t(\mathcal{J} = \mathcal{J} + \mathcal{A}) = 8M + 3S$.

Betrachtet man den Double-And-Add-Algorithmus (Algorithmus 1), so sieht man, dass P nur bei der Berechnung $Q = Q + P$ benutzt wird. Genau an dieser Stelle kann die gemischte Addition angewandt werden, indem Q im projektiven oder Jacobischen und P im affinen Koordinatensystem gegeben ist.

Optimierung Punktverdopplung: spezielle Wahl der elliptischen Kurve

Für die bilineare Abbildung wird später die elliptische Kurve mit der Gleichung $E : y^2 = x^3 + 1$ über dem endlichen Körper $\mathbb{F}_q$ benutzt werden. Die Wahl von $a = 0$ hat positive Auswirkungen auf die Berechnung der Punktverdopplung im projektiven und Jacobischen Koordinatensystem. Die Berechnung von $w = aZ_1^2 + 3X_1^2$ beziehungsweise $M = 3X_1^2 + aZ_1^4$ vereinfacht sich zu $w = 3X_1^2$ und $M = 3X_1^2$. Die Kosten der Punktverdopplung reduzieren sich dadurch zu $t^*(2\mathcal{P}) = 6M + 4S$ und $t^*(2\mathcal{J}) = 3M + 4S$. t^* bezeichnet dabei die Kosten bei Wahl der speziellen Kurve mit $a = 0$.

Im affinen System spart die Wahl von $a = 0$ lediglich eine Addition ein, welche allerdings im Rahmen der Aufwandsbetrachtung von vorneherein nicht berücksichtigt wird.

Optimierung Punktmultiplikation: NAF-Darstellung des Skalars

Für die Punktmultiplikation $Q = k \cdot P$ läuft für den Double-And-Add-Algorithmus die Schleife über die Binärdarstellung von k, $k = \sum_{i=0}^{m-1} b_i 2^i, b_i \in \{0, 1\}$, wobei m die Länge der Binärdarstellung ist. Wie zuvor erwähnt wird für jedes Bit eine Punktverdopplung durchgeführt und für jedes $b_i = 1$ zusätzlich noch eine Punktaddition. Da für ein zufälliges k im Schnitt etwa $\frac{m}{2}$ Bits Eins sind, erfolgen also etwa $\frac{m}{2}$ Punktadditionen.

Für k können aber auch andere Darstellungsformen benutzt werden. Das Inverse eines Punktes $P = (x, y) \in E(\mathbb{F}_q)$ ist $-P = (x, -y)$, kann also mit minimalem Aufwand berechnet werden (anders als in $\mathbb{F}_q^*$, wo die Inversion eine sehr teure Operation ist). Aus diesem Grund bietet sich für k die Darstellung in der *Non Adjacent Form (NAF)* an (siehe [HMV04, S. 98]). Dieses ist eine spezielle vorzeichenbehaftete Binärdarstellung.

Definition 4.2.1. *Die NAF-Darstellung einer positiven ganzen Zahl k ist $k = \sum_{i=0}^{m} k_i 2^i$, $k_i \in \{-1, 0, 1\}$ und für alle i ist $k_i \cdot k_{i+1} = 0$, das heißt keine zwei aufeinander folgenden k_i sind ungleich Null.*

Die NAF-Darstellung enthält also relativ viele Nullen. Für jede positive ganze Zahl k gibt es eine eindeutige NAF-Darstellung $NAF(k)$. So ist zum Beispiel die NAF-Darstellung von $255 = 1 \cdot 2^8 - 1 \cdot 2^0 = (10000000{-}1)_2$. Die Binärdarstellung hingegen besteht aus acht Einsen: $255 = (11111111)_2$. An diesem Beispiel wird auch die prinzipielle Idee bei der NAF-Darstellung deutlich, die auf der Transformation $1^a \mapsto 10^{(a-1)}{-}1$ basiert, es werden als a aufeinanderfolgende Einsen in eine Sequenz aus einer Eins, $a-1$ Nullen gefolgt von einer -1 verwandelt.

Algorithmus 2 zeigt, wie die NAF-Darstellung einer Zahl k berechnet werden kann. Der Algorithmus teilt k fortlaufend durch 2. Beginnend vom LSB (Least Significant Bit) erzeugt er die NAF. Ist k gerade, so wird k_i auf 0 gesetzt. Ist k ungerade, so wird k_i gerade so gesetzt, dass $(k - k_i)/2$ eine gerade Zahl ist. Dies stellt sicher, dass k_{i+1} in der nächsten Iteration auf 0 gesetzt wird. Ist $k = 3 \mod 4$, so wird $k_i = -1$ gesetzt, ist $k = 1 \mod 4$, so wird $k_i = 1$ gesetzt. Sobald also ein k_i ungleich 0 gesetzt wird, ist auf jeden Fall sichergestellt, dass das benachbarte k_{i+1} auf 0 gesetzt wird, somit können keine zwei benachbarten k_i, k_{i+1} jeweils ungleich 0 sein.

Die NAF-Darstellung hat maximal einen Koeffizienten mehr als die Binärdarstellung und es wurde gezeigt, dass im Mittel nur etwa $\frac{1}{3}$ der Koeffizienten ungleich Null sind [MO90]. Benutzt man im Double-And-Add-Algorithmus anstelle der Binärdarstellung von k die NAF-Darstellung von k, so reduziert sich die erwartete Laufzeit auf ebenfalls m Punktverdopplungen aber nur noch etwa $\frac{m}{3}$ Punktadditionen. Als Ergebnis erhält man Algorithmus 3, der zum Beispiel in [BHLM01] vorgestellt wurde.

Algorithmus 2: Berechnung der NAF Darstellung

Input : positive ganze Zahl k
Output : NAF(k)

$i = 0$
while $k \geq 1$ **do**
  if k *ungerade* **then**
    $k_i = 2 - (k \mod 4)$
    $k = k - k_i$
  else $k_i = 0$
  $k = k/2$
  $i = i + 1$
return $(k_{i-1}, k_{i-2}, \ldots, k_1, k_0)$

Algorithmus 3: Punktmultiplikation mit NAF Darstellung

Input : NAF(k) $= \sum_{i=0}^{l-1} k_i 2^i, P \in E(\mathbb{F}_q)$
Output : kP

$Q = \mathcal{O}$
for $i = l - 1$ **downto** 0 **do**
  $Q = 2 \cdot Q$
  if $k_i = 1$ **then** $Q = Q + P$
  if $k_i = -1$ **then** $Q = Q - P$
return Q

Vergleich der Koordinatensysteme

Wie zuvor erwähnt, sind für einen Vergleich des Berechnungsaufwandes der Operationen in den unterschiedlichen Koordinatensystemen die Anzahl an Inversionen (I), Multiplikationen (M) und Quadrierungen (S) von Interesse. Der Aufwand für eine Quadrierung wird in der Literatur meist mit $0,8M$ angegeben. Der Aufwand für eine Inversion ist ein Vielfaches vom Aufwand einer Multiplikation, allerdings hängt der Wert sehr stark vom verwendeten Algorithmus und der verwendeten Hardware ab. In [CMO98] wird ein Wert zwischen $9M$ und $30M$ als realistisch angesehen. Auf jeden Fall ist sie die teuerste der Basisoperationen und daher wird versucht, diese Operation möglichst zu vermeiden. Tabelle 4.1 fasst noch einmal den jeweiligen Aufwand für die Operationen in den entsprechenden Koordinatensysteme zusammen.

$t(\mathcal{A} = \mathcal{A} + \mathcal{A})$	$I + 2M + S$
$t(\mathcal{P} = \mathcal{P} + \mathcal{P}))$	$12M + 2S$
$t(\mathcal{P} = \mathcal{P} + \mathcal{A}))$	$9M + 2S$
$t(\mathcal{J} = \mathcal{J} + \mathcal{J})$	$12M + 4S$
$t(\mathcal{J} = \mathcal{J} + \mathcal{A})$	$8M + 3S$
$t(2\mathcal{A})$	$I + 2M + 2S$
$t(2\mathcal{P})$	$7M + 5S$
$t(2\mathcal{J})$	$4M + 6S$
$t^*(2\mathcal{P})$	$6M + 4S$
$t^*(2\mathcal{J})$	$3M + 4S$

Tabelle 4.1.: Aufwand für die Punktaddition und Punktverdopplung in $E(\mathbb{F}_q)$

Im affinen Koordinatensystem wird sowohl für die Punktaddition als auch für die Punktverdopplung jeweils eine Inversion benötigt. Das projektive und Jacobische System benötigen für die eigentliche Punktaddition und Punktverdopplung nur Multiplikationen, Quadrierungen und Additionen. Eine Inversion wird allerdings fällig, wenn zurück in das affine System transformiert wird.

Müssen Punkte übertragen beziehungsweise gespeichert werden, so wird dort fast ausschließlich die affine Darstellung benutzt, da ein Punkt nur aus zwei und nicht aus drei Koordinaten besteht und damit eine kürzere Darstellung hat. Daher macht es für eine einzelne Punktaddition oder Punktverdopplung keinen Sinn, das projektive oder Jacobische System zu benutzen, wenn man anschließend wieder eine Rücktransformation ins affine System durchführen muss und dort die kostspielige Inversion durchführt, die man gerade vermeiden wollte.

Anders sieht es aus, wenn eine ganze Folge von Punktadditionen oder Punktverdopplungen berechnet werden, so wie es bei einer Punktmultiplikation der Fall ist. Hier lohnt sich eine Berechnung im projektiven oder Jacobischen System, da eine Vielzahl von Inversionen eingespart wird und stattdessen nur weniger teure Operationen benutzt werden. Erst ganz am Ende der Berechnung erfolgt einmalig die Rücktransformation ins affine System und damit nur eine einzige kostspielige Inversion.

Im Vergleich zwischen projektivem und Jacobischem System fällt auf, dass im Jacobischen System die Punktverdopplung schneller ist, allerdings auf Kosten einer etwas langsameren Punktaddition. In Bezug auf den Double-And-Add-Algorithmus ist diese Gewichtung durchaus vorteilhaft, denn dort werden m Punktverdopplungen durchgeführt, aber im Mittel nur $\frac{m}{2}$ Punktadditionen. Somit wirkt sich der Geschwindigkeitsgewinn durch die schnellere Punktverdopplung im Jacobischen System stärker aus als der Verlust durch die etwas langsamere Punktaddition (im Vergleich zum projektiven System). Berücksichtigt man jetzt noch die Optimierungen bezüglich gemischter Addition und der speziellen Wahl der Kurve, so geht das Jacobische System sowohl bei der Punktaddition als auch bei der Punktverdopplung als Sieger gegenüber dem projektiven System hervor.

	Operationen in $\mathbb{F}_{q^2}$	Operationen in $\mathbb{F}_q$
$t(\mathcal{A} = \mathcal{A} + \mathcal{A})$	$I' + 2M' + S'$	$I + 10M + 2S$
$t(\mathcal{P} = \mathcal{P} + \mathcal{A}))$	$9M' + 2S'$	$31M$
$t(\mathcal{J} = \mathcal{J} + \mathcal{A})$	$8M' + 3S'$	$30M$
$t^*(2\mathcal{A})$	$I' + 2M' + 2S'$	$I + 12M + 2S$
$t^*(2\mathcal{P})$	$6M' + 4S'$	$26M$
$t^*(2\mathcal{J})$	$3M' + 4S'$	$17M$

Tabelle 4.2.: Aufwand für die Punktaddition und Punktverdopplung in $E(\mathbb{F}_{q^2})$

4.2.2. Arithmetik auf elliptischen Kurven über dem Körper $\mathbb{F}_{q^2}$

Die Arithmetik auf elliptischen Kurven über dem Körper $\mathbb{F}_{q^2}$ funktioniert im Prinzip genauso wie über dem Körper $\mathbb{F}_q$. Es werden die gleichen Formeln wie bei Kurven aus $E(\mathbb{F}_q)$ benutzt (vgl. Abschnitt 4.2.1), mit dem einzigen Unterschied, dass zur Berechnung der Punktaddition/Punktverdopplung als Körperoperationen die Addition, Subtraktion, Multiplikation und Inversion über $\mathbb{F}_{q^2}$ benutzt werden und Punkte $P = (X, Y)$ jetzt als $P = (a + ib, c + id)$ dargestellt werden, da die einzelnen Koordinaten Elemente aus $\mathbb{F}_{q^2}$ sind.

Auch der Double-And-Add Algorithmus funktioniert genauso, er benutzt entsprechend die auf $E(\mathbb{F}_{q^2})$ angepasste Punktaddition/Punktverdopplung. Die elliptische Kurve lässt sich auf die gleiche Art und Weise über verschiedenen Koordinatensystemen darstellen wie das für $E(\mathbb{F}_q)$ möglich ist. Seien I', M', S' die Aufwände für eine Inversion, Multiplikation beziehungsweise Quadrierung in $\mathbb{F}_{q^2}$. Die Optimierungen bezüglich des Berechnungsaufwandes aus Abschnitt 4.2.1 gelten genauso für $E(\mathbb{F}_{q^2})$ und sind in Tabelle 4.2 aufgelistet. Da sich die Operationen in $\mathbb{F}_{q^2}$ mittels Operationen im Basiskörper $\mathbb{F}_q$ realisieren lassen, ist insbesondere der Aufwand an Operationen in $\mathbb{F}_q$ von Interesse, da man so einen direkten Vergleich zu den Aufwänden in Tabelle 4.1 für die Operationen in $E(\mathbb{F}_q)$ herstellen kann. Mit $I' = I + 2M + 2S$, $M' = 3M$ und $S' = 2M$ (vgl. Abschnitt 4.1.3) ergeben sich die Werte in der rechten Spalte der Tabelle. Es bleibt festzuhalten, dass das Jacobische System gegenüber dem projektiven System Vorteile hat, da die Punktverdopplung deutlich schneller ist, während die Addition etwa gleichauf liegt.

Das projektive und Jacobische System in $E(\mathbb{F}_{q^2})$ verlieren gegenüber dem Aufwand für die Operationen in $E(\mathbb{F}_q)$ beinahe Faktor 3. Für das affine System sieht das etwas besser aus, denn sowohl in $E(\mathbb{F}_{q^2})$ als auch in $E(\mathbb{F}_q)$ wird für die Punktaddition und Punktverdopplung jeweils nur eine Inversion in $\mathbb{F}_q$ benötigt. Für $E(\mathbb{F}_{q^2})$ kommen noch einige Multiplikationen hinzu, so dass die Operationen in $E(\mathbb{F}_{q^2})$ vielleicht nur um etwa den Faktor 2 langsamer sind als die Operationen in $E(\mathbb{F}_q)$ (wobei diese Aussage natürlich stark vom Verhältnis zwischen I und M abhängt).

4.3. Erzeugung der Primzahlen p und q

Für das Gruppensignatur-Schema beziehungsweise für das Weil-Pairing werden zwei Primzahlen p und q benötigt, die folgende Eigenschaften erfüllen: p hat eine Größe von 160 Bit und q von etwa 512 Bit. Ferner müssen folgende Bedingungen gelten:

- $p \mid (q+1)$.
- $p^2 \nmid (q+1)$.
  Es wird später eine elliptische Kurve benutzt, die eine Ordnung von $q+1$ hat. $p \mid (q+1)$ und $p^2 \nmid (q+1)$ stellen zusammen sicher, dass dort exakt eine Untergruppe der Ordnung p existiert (vgl. Abschnitt 4.4).
- $p \mid q^2 - 1$.
  Diese Bedingung stellt sicher, dass $\mathbb{F}_{q^2}^*$ eine Untergruppe der Ordnung p enthält, die Gruppe $\mathbb{G}_T$ der p-ten Einheitswurzeln (vgl. Abschnitt 4.1.2). Zusätzlich wird diese Bedingung wichtig für den sogenannten Einbettungsgrad (vgl. Theorem (4.5.2)).
- $q \equiv 3 \ mod\ 4$.
  Diese Bedingung gewährleistet, dass $\mathbb{F}_{q^2}$ isomorph zum Körper $\mathbb{F}_q[X]/(X^2+1)$ ist (vgl. Abschnitt 4.1.2) und die Arithmetik in $\mathbb{F}_{q^2}$ entsprechend Abschnitt 4.1.3 umgesetzt werden kann.
- $q \equiv 2 \ mod\ 3$.
  Diese Bedingung ermöglicht es, in $\mathbb{F}_q$ effizient dritte Wurzeln zu berechnen. Sie ist ferner eine Voraussetzung dafür, dass die Kurve $E : Y^2 = X^3 + 1$ eine sogenannte supersinguläre Kurve ist (siehe Abschnitt 4.4).

Mit folgendem Vorgehen können p und q entsprechend der Anforderungen generiert werden:

1. Erzeuge mit einem Primzahlgenerator eine 160-Bit Primzahl p.
2. Generiere Primzahl q wie folgt:
   a) Generiere eine beliebige 348-Bit Zufallszahl r (nicht notwendigerweise prim)
   b) Setze $q = 12 \cdot p \cdot r - 1$. Überprüfe mit einem Primzahltest, ob q prim ist. Falls ja, dann ist ein passendes q gefunden, andernfalls gehe zurück zu a).

Spätestens nach einigen hundert Durchläufen der Schleife 2a), 2b) sollte ein passendes q gefunden worden sein, da nach dem Primzahlsatz ([MOV96, 4.1.2]) die Primzahlen wie zufällig verteilt und in genügender Anzahl vorhanden sind, so dass ein zufällig gewähltes x mit etwa einer Wahrscheinlichkeit von $\frac{1}{\ln x}$ eine Primzahl ist. Hier wird eine Primzahl gesucht, für die $q \equiv -1 \mod 12$ gelten soll. Nach [MOV96, 4.1.2, Fact 4.2] sind auch solche Primzahlen genügend häufig vorhanden, so dass man sie bei zufälliger Wahl von r mit etwa Wahrscheinlichkeit $\frac{1}{\varphi(12)\cdot\ln x} = \frac{1}{4\cdot\ln x}$ findet.

Durch die Konstruktion hat q tatsächlich etwa eine Länge von 512 Bit. Zu zeigen bleibt noch, dass p und q sämtliche geforderten Bedingungen erfüllen.
Da 3 und 4 Teiler von 12 sind, ist

- $q \equiv 12 \cdot p \cdot r - 1 \equiv 0 - 1 \equiv 3 \mod 4$.
- $q \equiv 12 \cdot p \cdot r - 1 \equiv 0 - 1 \equiv 2 \mod 3$.

Die Bedingung p teilt $q+1$ ist ebenfalls erfüllt, denn $q + 1 = 12 \cdot p \cdot r - 1 + 1 = 12 \cdot p \cdot r = p \cdot (12 \cdot r)$. Die Bedingung $p^2 \nmid (q+1)$ ist nach obiger Konstruktion mit hoher Wahrscheinlichkeit erfüllt. Andernfalls müsste gelten:

$$p^2 \mid (q+1) \ \Rightarrow\ p^2 \mid (12 \cdot p \cdot r) \ \Rightarrow\ p \mid (12 \cdot r) \ \Rightarrow\ p \mid r$$

Da p eine 160-Bit lange Primzahl ist und r zufällig aus dem Intervall $R = [2^{347}, 2^{348} - 1]$ gewählt wird, sind die ungünstigen Wahlen gerade die Vielfachen von p, davon liegen etwa R/p viele in R. Somit liegt die Wahrscheinlichkeit, ein ungünstiges r zu wählen, etwa bei $\frac{R/p}{R} = \frac{1}{p}$, dies entspricht etwa $\frac{1}{2^{160}}$ und ist damit vernachlässigbar klein.

p teilt auch $q^2 - 1$, denn $q^2 - 1 = (q + 1)(q - 1)$ und p teilt bereits $(q + 1)$. Für Theorem (4.5.2) wird später noch die zusätzliche Bedingung $p \nmid q - 1$ benötigt, die aber trivialer Weise erfüllt ist, da p schon $q+1$ teilt. Würde p auch noch $q-1$ teilen, so würde p als Konsequenz $(q + 1) - (q - 1) = 2$ teilen, p ist aber eine große Primzahl.

4.4. Finden eines Punktes der Ordnung p in $E(\mathbb{F}_q)$

Die Ordnung der Kurve $E : Y^2 = X^3 + 1$ ist $|E(\mathbb{F}_q)| = q + 1$, sofern $q = 2 \mod 3$ gewählt ist. Für diese spezielle Konstruktion ist jeder Wert $a \in \mathbb{F}_q^*$ ein kubischer Rest, es existiert also ein $b \in \mathbb{F}_q^*$ mit $b^3 \equiv a \mod q$. Sei $q = 3n + 2$, somit ist $q - 1 = 3n + 1$. Nach dem Satz von Fermat gilt [MOV96, 2.127]: $a^q \equiv a \mod q$ und $a^{q-1} \equiv 1 \mod q$. a lässt sich somit schreiben als

$$a \equiv a \cdot 1 \equiv a^q \cdot a^{q-1} \equiv a^{3n+2} \cdot a^{3n+1} \equiv a^{6n+3} \equiv (a^{2n+1})^3 \mod q.$$

Daher gibt es für jeden Wert $(Y^2-1) \in \mathbb{F}_q^*$ ein eindeutiges X mit $X^3 = Y^2-1$. Dies sind schon einmal $q-1$ Lösungen von $E(\mathbb{F}_q)$. Für den Sonderfall $Y^2 - 1 = 0$ ergibt sich, dass $(-1, 0)$ eine weitere Lösung von $E(\mathbb{F}_q)$ ist. Zuletzt ist auch der unendlich ferne Punkt ein Element der elliptischen Kurve, somit ist die Ordnung exakt $q + 1$.

Für die Gruppensignatur benötigt man eine Gruppe $\mathbb{G}_1$ der Ordnung p. Gesucht ist daher ein Punkt P auf der Kurve mit Ordnung p, wobei p und q entsprechend Abschnitt 4.3 gewählt sind.

Dieser Punkt P erzeugt eine Untergruppe der Ordnung p. Setze $\mathbb{G}_1 := \langle P \rangle \subseteq E(\mathbb{F}_q)$, die von P erzeugte Untergruppe. Der Satz von Lagrange [MOV96, 2.171] besagt, dass es für jeden positiven Teiler d einer Gruppenordnung eine Untergruppe der Ordnung d gibt. Für zyklische Gruppen gilt sogar, dass es für jeden Teiler d exakt eine zyklische Untergruppe der Ordnung d gibt (vgl. [MOV96, 2.172]). Nach [BF03, S. 610] bilden die Punkte von $E(\mathbb{F}_q)$ eine zyklische Gruppe mit Ordnung $q+1$. Die Bedingungen $p \mid (q+1)$ und $p^2 \nmid (q+1)$ stellen daher sicher, dass es exakt eine einzige Untergruppe der Ordnung p gibt. Da jeder Punkt mit Ordnung p eine Untergruppe der Ordnung p erzeugt, es aber nur eine einzige Gruppe dieser Ordnung gibt, liegen alle Punkte mit Ordnung p in derselben Gruppe, nämlich $\mathbb{G}_1$.

Um so einen Punkt P zu finden, geht man in zwei Schritten vor. Zuerst wird ein beliebiger Punkt $R \in E(\mathbb{F}_q)$ ermittelt. Normalerweise wählt man dazu ein zufälliges $X \in \mathbb{F}_q^*$, setzt es in die Kurvengleichung ein und löst anschließend die quadratische Gleichung $Y^2 = X^3 + 1$, um das zugehörige Y zu finden.

Bei der vorliegenden Kurve gibt es allerdings einen eleganteren Lösungsweg. Man wählt ein zufälliges Y und löst die Gleichung $X^3 = Y^2 - 1$. Wie zuvor schon gezeigt wurde, existiert ein eindeutiges X. Sei $a = Y^2 - 1 \in \mathbb{F}_q^*$, dann ist

$$(a^{(2q-1)/3})^3 \equiv a^{(2q-1)} \equiv a^q \cdot a^{(q-1)} \equiv a^1 \cdot a^{(q-1)} \cdot a^{(q-1)} \equiv a \mod q.$$

Mit $q = 2 \mod 3$ ist $(2q-1)/3 \in \mathbb{N}$, somit existiert $X = a^{(2q-1)/3}$ und ist die eindeutige dritte Wurzel von a. Die Eindeutigkeit ist wie folgt ersichtlich: Seien $x, y \in \mathbb{F}_q$ mit

$x^3 \equiv y^3 \equiv b \mod q$. Dann ist

$$b^{(2q-1)} \equiv (x^3)^{(2q-1)} \equiv x \mod q$$

$$b^{(2q-1)} \equiv (y^3)^{(2q-1)} \equiv y \mod q$$

Folglich ist $x \equiv y \mod q$.

Zusammenfassend ist es also mit nur wenigen Rechenoperationen möglich, einen Punkt $R = (X, Y)$ auf der Kurve zu bestimmen.

Um mit Hilfe von R einen Punkt P mit Ordnung p zu bestimmen, setzt man $P := h \cdot R$ mit $h := \frac{q+1}{p}$. Da p ein Teiler von $q+1$ ist, existiert h. Falls $P = \mathcal{O}$, so war R ungünstig gewählt und man startet noch einmal mit einem neuen R. Da h ein Teiler von $q+1$ ist, existiert eine Untergruppe mit Ordnung h, die gerade die Elemente enthält, die als Ordnung entweder h oder einen Teiler von h haben. Diese Elemente wären eine schlechte Wahl, allerdings erhält man so ein Element nur mit Wahrscheinlichkeit $\frac{h}{q+1} \approx \frac{1}{2^{160}}$.

Ist hingegen $P \neq \mathcal{O}$, so hat man einen Punkt P mit Ordnung p gefunden, denn $p \cdot P = (p \cdot h) \cdot R = (q+1) \cdot R = \mathcal{O}$.

4.5. Bilineare Abbildung

Bilineare Abbildungen werden durch Paarungen realisiert. Die bekanntesten Paarungen sind das Weil-Pairing und das Tate-Pairing. Inzwischen wurden weitere Paarungen gefunden, zum Beispiel das Ate-Pairing und das Eta-Pairing. Paarungen haben so interessante Eigenschaften, dass sie inzwischen einen eigenen Zweig in der Kryptografie bilden, die sogenannte paarungsbasierte Kryptografie.

Für die Gruppensignatur ist eine bilineare Abbildung $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ gefordert. Für die Umsetzung wird ausschließlich der Fall $\mathbb{G}_1 = \mathbb{G}_2$ betrachtet, also $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$, eine sogenannte symmetrische Paarung. Das Diskreter-Logarithmus-Problem muss in der Gruppe $\mathbb{G}_T$ schwer sein, andernfalls gibt es Möglichkeiten auch das DLP in $\mathbb{G}_1$ zu lösen, somit bliebe dann kein schwieriges Problem mehr übrig, auf dem man ein sicheres kryptografisches Verfahren aufsetzen könnte (siehe [Jou02, S. 21]). Als Folge davon kommen viele einfache bilineare Abbildungen nicht in Frage.

Für die konkrete Umsetzung der Gruppensignatur wurde das Weil-Pairing ausgewählt. Die folgenden Ausführungen orientieren sich an [HPS08, S. 315ff] und [BF03, Appendix A]. Für das Weil-Pairing werden speziell ausgewählte Gruppen benötigt.

Definition 4.5.1. *Sei E eine elliptische Kurve und m eine ganze Zahl mit $m \geq 1$. Die Menge der Punkte, deren Ordnung entweder m oder ein Teiler von m ist, wird bezeichnet mit*

$$E[m] = \{P \in E : mP = \mathcal{O}\}.$$

Diese Punkte werden auch Torsionspunkte beziehungsweise m-Torsionspunkte genannt.

$E[m]$ ist eine Untergruppe von E. Sollen die Punkte in einem bestimmten Körper K liegen, so schreibt man $E(K)[m]$.

Nach Behauptung 5.33 in [HPS08] gilt: Sei E eine elliptische Kurve über $\mathbb{F}_q$ und $q \nmid m$. Dann gibt es ein k, so dass

$$E(\mathbb{F}_{q^{jk}})[m] \cong \mathbb{Z}_m \times \mathbb{Z}_m, \forall j \geq 1. \tag{4.7}$$

Das heißt also, wenn man den Körper groß genug wählt, so enthält $E(\mathbb{F}_{q^k})[m]$ m^2 viele m-Torsionspunkte. Ferner existieren zwei Torsionspunkte T_1, T_2, mit denen sich jedes Element von $E(\mathbb{F}_{q^k})[m]$ erzeugen lässt, das heißt für jeden Punkt $R \in E(\mathbb{F}_{q^k})[m]$ gibt es eindeutige $a, b \in \mathbb{Z}_m$, so dass $R = aT_1 + bT_2$. Im Folgenden bezeichnet $E[m]$ gerade $E(\mathbb{F}_{q^k})[m]$.
Das folgende Theorem zeigt, wie sich k bestimmen lässt.

Theorem 4.5.2 ([BK98, Theorem 1]). *Sei E eine elliptische Kurve über $\mathbb{F}_q$. Sei m eine Primzahl, die die Anzahl der Punkte der elliptischen Kurve $N = |E(\mathbb{F}_q)|$ teilt, aber m teilt nicht $q - 1$. Dann enthält $E(\mathbb{F}_{q^k})$ m^2 Punkte der Ordnung m, genau dann, wenn $m \mid q^k - 1$.*

Definition 4.5.3 ([Men09, Definition 4.1]). *Sei E eine elliptische Kurve über $\mathbb{F}_q$ und sei $P \in E(\mathbb{F}_q)$ ein Punkt mit Ordnung m, m prim. Sei $\mathrm{ggt}(m, q) = 1$. Dann ist der Einbettungsgrad (embedding degree) von $G = \langle P \rangle$ das kleinste k, für das $m \mid q^k - 1$ gilt.*

Für die Berechnung des Weil-Pairings sucht man solche Kurven, auf denen Gruppen mit kleinem Einbettungsgrad existieren, denn andernfalls wird $\mathbb{F}_{q^k}$ so groß, dass sich dort nicht mehr effizient rechnen lässt.

4.5.1. Weil-Pairing

Das Weil-Pairing e_m ist eine Abbildung, das Paare von Punkten aus $E[m]$ auf Elemente der Gruppe der m-ten Einheitswurzeln in $\mathbb{F}_{q^k}^*$ abbildet. Die genaue Definition von e_m erfolgt später, da dafür noch zusätzliches Hintergrundwissen benötigt wird. e_m wird folgende Eigenschaften besitzen [HPS08, Theorem 5.38]:

- $e_m(P, Q)^m = 1 \quad \forall P, Q \in E[m]$. $e_m(P, Q)$ ist also eine m-te Einheitswurzel.
- Das Weil-Pairing ist eine bilineare Abbildung:

$$\begin{aligned} e_m(P_1 + P_2, Q) &= e_m(P_1, Q) \cdot e_m(P_2, Q) \quad \forall P_1, P_2, Q \in E[m] \\ e_m(P, Q_1 + Q_2) &= e_m(P, Q_1) \cdot e_m(P, Q_2) \quad \forall P, Q_1, Q_2 \in E[m] \end{aligned}$$

  Zu beachten ist, dass für die linke Seite eine additive Schreibweise genutzt wird, während auf der rechten Seite eine multiplikative Schreibweise genutzt wird, da in $E[m]$ die Gruppenoperation die Punktaddition ist, während in $\mathbb{F}_{q^k}^*$ die Gruppenoperation die Multiplikation ist.

- $e_m(P, P) = 1 \quad \forall P \in E[m]$ und
  $e_m(P, Q) = e_m(Q, P)^{-1} \quad \forall P, Q \in E[m]$.
- Nicht-Degeneriertheit:
  Falls $e_m(P, Q) = 1 \quad \forall Q \in E[m]$, dann gilt $P = \mathcal{O}$.

4.5.2. Rationale Funktionen und Divisoren auf elliptischen Kurven

Für die Definition des Weil-Pairings werden rationale Funktionen über einer elliptischen Kurve benötigt und dabei insbesondere der Zusammenhang zu den Nullstellen und Polstellen der Funktion. Zuerst wird der einfachere Fall einer rationalen Funktion mit einer

Variablen betrachtet:

$$f(X) = \frac{a_0 + a_1X + a_2X^2 + \cdots + a_zX^z}{b_0 + b_1X + b_2X^2 + \cdots + b_nX^n}$$

Erlaubt man komplexe Zahlen, so lassen sich Zähler und Nenner als Produkt der Linearfaktoren des jeweiligen Zähler- und Nennerpolynoms darstellen:

$$f(X) = \frac{a(X-\alpha_1)^{e_1}(X-\alpha_2)^{e_2}\cdots(X-\alpha_r)^{e_r}}{b(X-\beta_1)^{d_1}(X-\beta_2)^{d_2}\cdots(X-\beta_s)^{d_s}}$$

Die Zahlen $\alpha_1, \ldots, \alpha_r$ sind gerade die Nullstellen von $f(X)$, während die Zahlen $\beta_1, \ldots, \beta_s$ genau die Polstellen von $f(X)$ sind. Die Exponenten $e_1, \ldots, e_r, d_1, \ldots, d_s$ sind die Vielfachheiten der jeweiligen Nullstelle oder Polstelle. Um einen Ausdruck für die Nullstellen und Polstellen mit ihren Vielfachheiten zu haben, führt man den *Divisor* von $f(X)$ ein. Betrachtet man X für $X \to \infty$, so hat die Funktion im Unendlichen eine Polstelle, falls $z > n$, oder eine Nullstelle, falls $n > z$ ist. Daher kann man auch den Punkt im Unendlichen mit in den Divisor aufnehmen. Der Divisor von $f(X)$ ist definiert als die formale Summe

$$div(f(X)) = e_1[\alpha_1] + e_2[\alpha_2] + \cdots + e_r[\alpha_r] - d_1[\beta_1] - d_2[\beta_2] - \cdots - d_s[\beta_s] - (z-n)[P_\infty].$$

Sei $E : Y^2 = X^3 + AX + B$ eine elliptische Kurve und f(X,Y) eine rationale Funktion in zwei Variablen, so gibt es Punkte $P = (X, Y)$ auf E, so dass $f(X, Y)$ im Zähler oder Nenner Null wird, also hat f Nullstellen oder Polstellen auf der Kurve E. Auch hier kann man entsprechend Vielfachheiten zuordnen, so dass sich der Divisor $div(f)$ darstellen lässt als: $div(f) = \sum_{P \in E} n_P[P]$, mit $n_P \in \mathbb{Z}$. Nur endlich viele der n_P sind ungleich 0, so dass $div(f)$ eine endliche Summe ist. Allgemein definiert man einen Divisor auf E wie folgt:

Definition 4.5.4. *Sei E eine elliptische Kurve. So ist ein Divisor $\mathcal{A}$ definiert als die formale Summe:*

$$\mathcal{A} = \sum_{P \in E} n_P[P], \text{ mit } n_P \in \mathbb{Z} \text{ und } n_P = 0 \text{ für alle bis auf endliche viele Punkte } P.$$

Der Grad eines Divisors $\mathcal{A}$ ist $deg(\mathcal{A}) = \sum_{P \in E} n_P$, die Summe eines Divisors $\mathcal{A}$ ist

$$Sum(\mathcal{A}) = Sum\left(\sum_{P \in E} n_P[P]\right) = \sum_{P \in E} n_P P$$

und der *Träger (Support)* eines Divisors $\mathcal{A}$ ist die Menge der Punkte P mit $n_P \neq 0$.

Bei dem Ausdruck $Sum(\mathcal{A})$ meint $n_P P$ die Punktmultiplikation des Punktes P mit dem Skalar n_P auf der elliptischen Kurve E und die Summe wird mit Hilfe der Punktaddition realisiert.

Von besonderem Interesse ist der Zusammenhang zwischen Funktionen und Divisoren und in welchem Maß ein Divisor eine Funktion spezifiziert:

Theorem 4.5.5 ([HPS08, Theorem 5.36]). *Sei E eine elliptische Kurve.*

a. *Seien f und f' rationale Funktionen auf E. Falls $div(f) = div(f')$, so unterscheiden sich f und f' nur um einen konstanten Faktor.*

b. *$\mathcal{A} = \sum_{P \in E} n_P[P]$ ist genau dann ein Divisor einer rationalen Funktion f auf E, wenn $deg(\mathcal{A}) = \sum_{P \in E} n_P = 0$ und $Sum(\mathcal{A}) = \sum_{P \in E} n_P P = \mathcal{O}$. Solch einen Divisor nennt man Hauptdivisor.*

Schließlich wird noch folgende Aussage über die Äquivalenz von Divisoren benötigt [BF03, S. 610]:
Zwei Divisoren $\mathcal{A}, \mathcal{B}$ sind äquivalent, falls ihre Differenz $\mathcal{A} - \mathcal{B}$ ein Hauptdivisor ist. Damit ist ein einzelner Divisor ein Repräsentant für eine ganze Klasse von Divisoren.

Folgende Notation wird im Weiteren genutzt: Sei $\mathcal{A} = \sum_{P \in E} n_P[P]$, so bezeichne $f(\mathcal{A}) = \prod_{P \in E} f(P)^{n_p}$. Dieser Ausdruck ist wohldefiniert, wenn der Träger von $div(f)$ und der Träger von $\mathcal{A}$ disjunkt sind [Jou04][S. 265].

Sei beispielsweise $\mathcal{A} = [P_1] - [P_2]$, so ist $f(\mathcal{A}) = \frac{f(P_1)}{f(P_2)}$. Für die Berechnung des Wertes $f(\mathcal{A})$ ist es unerheblich, ob die Funktion f oder eine Funktion $c \cdot f$ mit $c \in \mathbb{F}_{q^k}^*$ benutzt wird, da der Grad des Divisors $\mathcal{A}$ Null ist und damit die Konstante innerhalb des Produktes $\prod_{P \in E} cf(P)^{n_p}$ genauso häufig im Zähler wie im Nenner vorkommt und sich demzufolge herauskürzt. Zusammen mit Theorem (4.5.5) kann daher zur Berechnung von $f(\mathcal{A})$ jede Funktion f' benutzt werden, für die $div(f') = div(f)$ gilt.

4.5.3. Definition von e_m

Folgende Beobachtung ist für die Definition des Weil-Pairings wichtig: Sei $P \in E[m]$ ein Punkt der Ordnung m. Für die konkrete Umsetzung wird m später eine Primzahl sein. Nach Definition ist $mP = \mathcal{O}$. Dann ist $\mathcal{A} = m[P] - m[\mathcal{O}]$ ein Hauptdivisor, denn $deg(\mathcal{A}) = m - m = 0$ und $Sum(\mathcal{A}) = mP - m\mathcal{O} = \mathcal{O}$. Nach Theorem (4.5.5) existiert daher eine rationale Funktion $f_P(X, Y)$ auf E mit dem Divisor $div(f_P) = m[P] - m[\mathcal{O}]$. Dieser Zusammenhang wird in der folgenden Definition genutzt.

Definition 4.5.6. *Seien $P, Q \in E[m]$, also Punkte der Ordnung m. Sei $\mathcal{A}_P$ ein Divisor, der äquivalent zu $[P] - [\mathcal{O}]$ ist, und $\mathcal{A}_Q$ ein Divisor, der äquivalent zu $[Q] - [\mathcal{O}]$ ist, und seien der Träger von $\mathcal{A}_P$ und $\mathcal{A}_Q$ disjunkt. Seien f_P, f_Q rationale Funktionen auf E mit $div(f_P) = m\mathcal{A}_P$ und $div(f_Q) = m\mathcal{A}_Q$. Dann ist das Weil-Pairing von P und Q definiert als*

$$e_m(P, Q) = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)}.$$

Die Divisoren $\mathcal{A}_P$ und $div(f_P)$ beziehungsweise $\mathcal{A}_Q$ und $div(f_Q)$ besitzen den gleichen Träger, daher sind nach Voraussetzung auch die Träger von $div(f_P)$ und $\mathcal{A}_Q$ beziehungsweise $div(f_Q)$ und $\mathcal{A}_P$ disjunkt. Durch diesen Zusammenhang ist gewährleistet, dass der Ausdruck $\frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)}$ wohldefiniert ist, also weder 0 noch ∞.

Die konkrete Wahl der Divisoren $\mathcal{A}_P, \mathcal{A}_Q$ ist für $e_m(P, Q)$ unerheblich, solange sie äquivalent zu $[P] - [\mathcal{O}]$ beziehungsweise $[Q] - [\mathcal{O}]$ sind. Ein Beweis dieser Aussage findet sich in [BF03, S. 611]. Dort wird gezeigt, dass für einen zu $\mathcal{A}_P$ äquivalenten Divisor $\hat{\mathcal{A}}_P$

und die Funktion $\hat{f}_P$ mit Divisor $div(\hat{f}_P) = m\hat{\mathcal{A}}_P$ folgendes gilt:

$$e_m(P,Q) = \frac{\hat{f}_P(\mathcal{A}_Q)}{f_Q(\hat{\mathcal{A}}_P)} = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)}.$$

Analoges gilt für einen zu $\mathcal{A}_Q$ äquivalenten Divisor $\hat{\mathcal{A}}_Q$ und einer Funktion $\hat{f}_Q$ mit Divisor $div(\hat{f}_Q) = m\hat{\mathcal{A}}_Q$. Daher kann man tatsächlich von *dem* $e_m(P,Q)$ sprechen.

Welche Divisoren werden jetzt konkret für $\mathcal{A}_P$ und $\mathcal{A}_Q$ gewählt? Seien $P, Q \in E[m]$ und seien R, S zwei beliebige Punkte in $E[m]$, für die folgendes gilt:

- $P+R, R$ sind weder Pol noch Nullstelle von f_Q, sind also kein Element des Trägers von $div(f_Q)$.
- $Q+S, S$ sind weder Pol noch Nullstelle von f_P, sind also kein Element des Trägers von $div(f_P)$.

Die Divisoren $\mathcal{A}_P = [P+R]-[R]$ und $\mathcal{A}_Q = [Q+S]-[S]$ sind äquivalent zu den Divisoren $[P]-[\mathcal{O}]$ beziehungsweise $[Q]-[\mathcal{O}]$. Zur Erinnerung, zwei Divisoren sind äquivalent, wenn ihre Differenz ein Hauptdivisor ist. Sei $\mathcal{A} = \mathcal{A}_P - ([P]-[\mathcal{O}]) = [P+R]-[R]-[P]+[\mathcal{O}]$. $\mathcal{A}$ ist ein Hauptdivisor, denn $deg(\mathcal{A}) = 0$ und $Sum(\mathcal{A}) = (P+R) - R - P + \mathcal{O} = P - P + \mathcal{O} = \mathcal{O}$. Somit ist $\mathcal{A}_P$ tatsächlich äquivalent zu $[P]-[\mathcal{O}]$, analoges gilt für $\mathcal{A}_Q$. Daher können $\mathcal{A}_P$ und $\mathcal{A}_Q$ für die Berechnung des Weil-Pairings benutzt werden:

$$e_m(P,Q) = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)} = \frac{f_P(Q+S)}{f_P(S)} \Big/ \frac{f_Q(P+R)}{f_Q(R)} = \frac{f_P(Q+S)\cdot f_Q(R)}{f_P(S)\cdot f_Q(P+R)} \tag{4.8}$$

Durch die spezielle Wahl von R und S wird sichergestellt, dass bei der Auswertung von f_P oder f_Q keiner der Ausdrücke 0 wird, somit ist der Ausdruck $\frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)}$ wohldefiniert.

Als letztes verbleibt jetzt noch zu zeigen, wie man die Funktionen f_P und f_Q zu den Divisoren $m\mathcal{A}_P = m[P+R] - m[R]$ und $m\mathcal{A}_Q = m[Q+S] - m[S]$ konstruiert. Dies geschieht mit Hilfe des Miller-Algorithmus, der im nächsten Abschnitt vorgestellt wird.

4.5.4. Miller-Algorithmus zur Berechnung des Weil-Pairings

Victor Miller beschrieb in einer unveröffentlichen Arbeit [Mil86] erstmals, wie das Weil-Pairing berechnet werden kann (später noch einmal in [Mil04]). Der dort beschriebene Algorithmus ist inzwischen als Miller-Algorithmus bekannt. Das Besondere am Miller-Algorithmus ist, dass er die Funktionen f_P und f_Q nicht explizit als ganzrationale Funktionen in den Variablen x und y konstruiert, denn diese Darstellung könnte exponentiell in der Eingabegröße sein und damit nicht effizient [Mil86, S. 1]. Stattdessen genügt es, f_P und f_Q nur an bestimmten Stellen auswerten zu können, nämlich gerade in den Punkten, die im Träger des Divisors $\mathcal{A}_Q$ beziehungsweise $\mathcal{A}_P$ enthalten sind (für andere Punkte können f_P, f_Q undefiniert sein) [Mil04, S. 238].

Im Folgenden wird gezeigt, wie man den Funktionswert $f_P(\mathcal{A}_Q)$ berechnet, die Berechnung von $f_Q(\mathcal{A}_P)$ erfolgt analog, zusammen erhält man so den Wert $e_m(P,Q)$.

Für eine positive ganze Zahl b definiere den Divisor

$$\mathcal{A}_b = b[P+R] - b[R] - [bP] + [\mathcal{O}]$$

$\mathcal{A}_b$ ist ein Hauptdivisor, daher existiert eine Funktion f_b, so dass $div(f_b) = \mathcal{A}_b$. Für den Fall $b = m$ und mit $mP = \mathcal{O}$ ergibt sich der Divisor $div(f_m) = m[P+R]-m[R]-[mP]+[\mathcal{O}] = m[P+R]-m[R] = div(f_P)$. Die Funktionen f_m und f_P unterscheiden sich nach Theorem (4.5.5) also nur um einen konstanten Faktor, daher ist $f_m(\mathcal{A}_Q) = f_P(\mathcal{A}_Q)$, denn der konstante Faktor tritt im Zähler und Nenner gleich häufig auf und kürzt sich daher heraus.

Im Folgenden wird ein Algorithmus vorgestellt, der bei Eingabe der Funktionswerte $f_b(\mathcal{A}_Q)$, $f_c(\mathcal{A}_Q)$ und der Punkte $bP, cP, (b+c)P$ für $b, c > 0$ als Ausgabe den Funktionswert $f_{b+c}(\mathcal{A}_Q)$ berechnet.

Es werden zwei Hilfsfunktionen benötigt, dazu seien $U = (x_1, y_1), V = (x_2, y_2)$ zwei Punkte auf E:

1. Die Gerade durch die Punkte U und V lässt sich durch folgende Gleichung beschreiben (vgl. Kapitel 4.2.1 und [HPS08, S. 280]):

$$Y - y_1 - \lambda(X - x_1) = 0 \text{ mit } \lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

   Für den Fall $U = V$ ist $\lambda = \frac{3x_1^2+a}{2y_1}$, die Steigung der Tangente im Punkt U.
   Definiere die Funktion

$$L_{U,V}(X, Y) = Y - y_1 - \lambda(X - x_1),\ \lambda \text{ wie zuvor.}$$

2. Die Vertikale durch den Punkt U wird durch die Gleichung $X - x_1 = 0$ beschrieben. Definiere die Funktion

$$V_U(X, Y) = X - x_1.$$

Dann ergibt sich für die Punkte aP, bP und $(b + c)P$:

1. $L_{bP,cP}(X, Y)$ hat als Nullstellen die Punkte bP, cP und $-(b + c)P$, denn das sind genau die Schnittpunkte der Geraden mit der Kurve E. Im Endlichen hat die Funktion keine Polstellen. Da der Divisor der Funktion nach Theorem (4.5.5) von Seite 51 Grad 0 haben muss, ergibt sich eine dreifache Polstelle im Punkt $\mathcal{O}$. Der Divisor von $L_{bP,cP}$ ist also:

$$div(L_{bP,cP}) = [bP] + [cP] + [-(b + c)P] - 3[\mathcal{O}].$$

2. $V_{(b+c)P}(x, y)$ hat als Nullstellen die Punkte $(b+c)P$ und $-(b+c)P$, denn das sind genau die Schnittpunkte der Vertikalen mit der Kurve E. Auch $V_{(b+c)P}$ hat keine Polstellen im Endlichen, daher ist der Divisor von $V_{(b+c)P}$:

$$div(V_{(b+c)P}) = [(b + c)P] + [-(b + c)P] - 2[\mathcal{O}].$$

Strikt nach der Definition von $\mathcal{A}_b$ ergeben sich folgende Divisoren:

$$\begin{aligned} \mathcal{A}_b &= b[P + R] - b[R] - [bP] + [\mathcal{O}] \\ \mathcal{A}_c &= c[P + R] - c[R] - [cP] + [\mathcal{O}] \\ \mathcal{A}_{b+c} &= (b + c)[P + R] - (b + c)[R] - [(b + c)P] + [\mathcal{O}] \end{aligned}$$

Durch Rechnen mit den Divisoren ergibt sich, dass

$$\mathcal{A}_{b+c} = \mathcal{A}_b + \mathcal{A}_c + div(L_{bP,cP}) - div(V_{(b+c)P}).$$

Zu den Divisoren existieren die Funktionen $f_{b+c}, f_b, f_c, L_{bP,cP}, V_{(b+c)P}$, daher gilt:

$$f_{b+c}(\mathcal{A}_Q) = f_b(\mathcal{A}_Q) \cdot f_c(\mathcal{A}_Q) \cdot \frac{L_{bP,cP}(\mathcal{A}_Q)}{V_{(b+c)P}(\mathcal{A}_Q)}$$

Falls also $f_b(\mathcal{A}_Q)$ und $f_c(\mathcal{A}_Q)$ gegeben sind, so genügt es die Werte $L_{bP,cP}(\mathcal{A}_Q)$ und $V_{(b+c)P}(\mathcal{A}_Q)$ zu berechnen, um nach obiger Formel $f_{b+c}(\mathcal{A}_Q)$ zu erhalten. Das zuvor skizzierte Vorgehen setzt der Miller-Algorithmus konkret um. Algorithmus 4 zeigt den Miller-Algorithmus im Pseudocode, er wurde aus [Lyn07, S. 36] entnommen, findet sich alternativ auch in [BMX06, S. 138]. Analog zu einem Double-And-Add-Algorithmus berechnet er in einer Schleife über die Binärdarstellung von m den Wert mP mit Hilfe von Punktverdopplungen und Punktadditionen. In diesem Fall interessiert allerdings nicht das Ergebnis mP, denn $mP = \mathcal{O}$ ist schon vorher bekannt. Stattdessen interessieren die Geradengleichungen, die bei der Punktaddition beziehungsweise Punktverdopplung implizit berechnet werden, denn diese liefern innerhalb der jeweiligen Schleifeniteration gerade die Funktionen $L_{bP,cP}$ und $V_{(b+c)P}$ zur Berechnung von f_{b+c}, sodass am Ende der Schleife tatsächlich der Wert f_m berechnet wird.

Sei $Z = kP$. Sei T_Z eine abkürzende Schreibweise für die Tangente im Punkt Z, also $T_Z = L_{Z,Z}$. Für eine Punktverdopplung $2Z$ setze $b = k$ und $c = k$, für eine Punktaddition $Z + P$ setze $b = k$ und $c = 1$. Mit diesen Werten ergibt sich im Schleifendurchlauf k innerhalb des Algorithmus

$$f_{2k} = f_k^2 \cdot \frac{T_Z}{V_{2Z}} \text{ und } \quad f_{k+1} = f_k \cdot f_1 \cdot \frac{L_{Z,P}}{V_{Z+P}}.$$

Definiere $f_0 = 1$. Unklar ist bisher noch, wie sich f_1 berechnen lässt. Nach Definition ist

$$div(f_1) = \mathcal{A}_1 = [P + R] - [R] - [P] + [\mathcal{O}].$$

Durch Rechnen mit den Divisoren ergibt sich:

$$\begin{aligned} div(V_{P+R}) - div(L_{P,R}) &= [P + R] + [-(P + R)] - 2[\mathcal{O}] - ([P]+[R]+[-(P + R)]-3[\mathcal{O}]) \\ &= [P + R] - [P] - [R] + [\mathcal{O}] \\ &= \mathcal{A}_1 \end{aligned}$$

Folglich ist $f_1 = \frac{V_{P+R}}{L_{P,R}}$.

Angemerkt sei an dieser Stelle noch, dass im letzten Durchlauf der Schleife des Miller-Algorithmus in Zeile 8 ein degenerierter Fall entsteht. Durch die letzte Punktverdopplung erhält Z den Wert $Z = (p-1)P = -P$. Durch die letzte Addition wird Z zu $Z = -P + P = \mathcal{O}$. Dadurch wird bei der Auswertung $L_{Z,P}(Q)/V_{Z+P}(Q)$ die Gerade $L_{Z,P} = L_{-P,P}$ tatsächlich zu einer Vertikalen, während $V_{Z+P} = V_{\mathcal{O}}$ für genau diesen Sonderfall als 1 definiert wird [Lyn07, S. 40].

Zur Erinnerung, $f_m(\mathcal{A}_Q)$ lässt sich mittels $\frac{f_m(Q+S)}{f_m(S)}$ berechnen. Algorithmus 4 zeigt die Berechnung von $f_m(Q)$, er müsste also einmal für $f_m(Q + S)$ und einmal für $f_m(S)$

Algorithmus 4: Miller-Algorithmus zur Berechnung des Weil-Pairing

Input : $P, R, Q \in E[m], m = m_t, m_{t-1}, \ldots, m_0$
Output : $f = f_m(Q) = f_P(Q) \in \mathbb{F}_{q^k}^*$

$f_1 = V_{P+R}(Q)/L_{P,R}(Q)$
$f = f_1$
$Z = P$
for $i = t - 1$ **downto** 0 **do**
    $f = f^2 \cdot T_Z(Q)/V_{2Z}(Q)$
    $Z = 2Z$
    if $m_i = 1$ **then**
        $f = f \cdot f_1 \cdot L_{Z,P}(Q)/V_{Z+P}(Q)$
        $Z = Z + P$
return f

aufgerufen werden. Effizienter ist es allerdings, den Algorithmus so zu erweitern, dass die Auswertung für die Punkte $Q + S$ und S gleichzeitig innerhalb eines einzigen Aufrufs durchgeführt wird, da für $Q + S$ und S innerhalb des Algorithmus exakt dieselben Funktionen L, T, V ausgewertet werden müssen.

Für die Auswertung der Funktionen L, T werden nur eine Multiplikation und drei Subtraktionen in $\mathbb{F}_{q^k}$ benötigt, die Steigung λ erhält man automatisch bei der Berechnung der Punktverdopplung beziehungsweise der Punktaddition. Für die Auswertung der Funktionen V wird nur eine Subtraktion in $\mathbb{F}_{q^k}$ benötigt. Die Schleife des Algorithmus hat $\log(m)$ Iterationen. Daher lässt sich durch mehrmalige Anwendung des Miller-Algorithmus das Weil-Pairing

$$e_m(P, Q) = \frac{f_P(Q + S) \cdot f_Q(R)}{f_P(S) \cdot f_Q(P + R)}$$

effizient berechnen, sofern die Operationen in $\mathbb{F}_{q^k}$ effizient berechenbar sind. Eine genauere Laufzeitanalyse erfolgt später noch in Kapitel 5.6.2 an einer optimierten Variante des Miller-Algorithmus.

4.5.5. Modifiziertes Weil-Pairing

Sei E eine elliptische Kurve über $\mathbb{F}_q$, G eine Untergruppe in $E(\mathbb{F}_q)$ der Ordnung m (damit ist G auch eine Teilmenge von $E[m]$), m prim (G ist zyklisch) und k der Einbettungsgrad von G. Sei $\mathbb{G}_1 = G, \mathbb{G}_2 = \mathbb{G}_1$ und $\mathbb{G}_T$ die Gruppe der m-ten Einheitswurzeln in $\mathbb{F}_{q^k}^*$. Mit diesen Voraussetzungen erfüllt das Weil-Pairing e_m schon beinahe die Anforderungen an eine bilineare Abbildung, wie sie in Kapitel 3.4.2 für die Gruppensignatur gefordert ist. Leider ist die Abbildung bei dieser Festlegung der Gruppen degeneriert, denn seien $P, Q \in \mathbb{G}_1,\ P, Q \neq \mathcal{O}$, so existiert ein $a \in \mathbb{F}_q$ mit $Q = aP$, denn jeder Punkt außer $\mathcal{O}$ ist ein Generator von $\mathbb{G}_1$ (siehe Kapitel 3.1). Daher ist $e_m(P, Q) = e_m(P, aP) = e_m(P, P)^a = 1^a = 1$. An dieser Stelle hilft jetzt eine spezielle Wahl der Kurve E.

Spezielle Konstruktion

Seien die Primzahlen p, q so gewählt wie in Kapitel 4.3 spezifiziert. Setze $m = p$. Entsprechend Kapitel 4.4 hat die supersinguläre Kurve $E : Y^2 = X^3 + 1$ über $\mathbb{F}_q$ eine Ordnung von $q + 1$ und es existiert die Untergruppe $\mathbb{G}_1$ mit Ordnung p, da $p \mid q + 1$.

Ferner teilt p auch $q^2 - 1$, damit hat die Gruppe $\mathbb{G}_1$ nach Theorem (4.5.2) den Einbettungsgrad $k = 2$ (siehe dazu auch [KM05, Kap. 7] und [FST06, Kap. 3.2]) und $E(\mathbb{F}_{q^2})$ enthält p^2 viele p-Torsionspunkte. Damit ist die gesuchte Gruppe $E[p]$ also gerade $E(\mathbb{F}_{q^2})[p]$.

Entsprechend Kapitel 4.1.2 existiert in $\mathbb{F}_{q^2}^*$ die Gruppe $\mathbb{G}_T$ mit Ordnung p, die Gruppe der p-ten Einheitswurzeln.

Somit wird das Weil-Pairing zu einer Abbildung $e_p : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$. Nach wie vor ist die Abbildung degeneriert. Seien $P, Q \in \mathbb{G}_1$. Damit die Abbildung $e_p(P, Q)$ nicht degeneriert ist, darf Q kein Vielfaches von P sein, folglich darf Q nicht aus $\mathbb{G}_1$ sein. Alle Punkte der Ordnung p in $E(\mathbb{F}_q)$ liegen allerdings in $\mathbb{G}_1$ (vgl. Kapitel 4.4). Daher muss Q also notgedrungen aus der großen Gruppe $E(\mathbb{F}_{q^2})$ beziehungsweise genauer aus $E(\mathbb{F}_{q^2})[p] \backslash \mathbb{G}_1$ gewählt werden. Wie zuvor erwähnt existieren in $E(\mathbb{F}_{q^2})$ p^2 Punkte der Ordnung p. Daher existieren dort $p + 1$ unterschiedliche Untergruppen der Ordnung p, denn jede Untergruppe enthält $p - 1$ Punkte der Ordnung p und zusätzlich $\mathcal{O}$. Eine dieser Untergruppen ist $\mathbb{G}_1$. Bezeichne $\mathbb{G}_2$ eine der anderen Untergruppen mit Ordnung p. Damit wäre das Weil-Pairing als Abbildung von $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ tatsächlich nicht degeneriert.

Allerdings handelt man sich damit neuere Herausforderungen ein. Für die Gruppensignatur benötigt man einen Isomorphismus von $\mathbb{G}_2$ nach $\mathbb{G}_1$. Zusätzlich muss man nach $\mathbb{G}_2$ hashen können, was wiederum auch nur für spezielle Konstellationen möglich ist, falls $\mathbb{G}_2$ nicht in $E(\mathbb{F}_q)$ liegt, auf die hier nicht näher eingegangen wird (vgl. [SBC+09]).

Glücklicherweise hilft uns hier unsere supersingulären Kurve E weiter, denn auf ihr existiert eine sogenannte *Distortion-Map*. Diese ist definiert als

$$\phi : E(\mathbb{F}_q) \to E(\mathbb{F}_{q^2}),\ \phi(x, y) \mapsto (\alpha x, y),\ \text{mit } \alpha^3 = 1, \alpha \neq 1$$

ϕ bildet einen Punkt aus $E(\mathbb{F}_q)$ auf einen Punkt aus $E(\mathbb{F}_{q^2})$ ab. α ist eine nichttriviale dritte Einheitswurzel in $\mathbb{F}_{q^2}$. Folgende Eigenschaften gelten, ein Beweis findet sich in Anhang A:

- $\alpha = \frac{-1+i\delta}{2}$ mit $\delta = 3^{\frac{q+1}{4}} \in \mathbb{F}_q$ ist eine dritte Einheitswurzel in $\mathbb{F}_{q^2}$.
- Sei $P \in E(\mathbb{F}_q)$. Dann ist $\phi(P) \in E(\mathbb{F}_{q^2})$.
- $\phi(P + Q) = \phi(P) + \phi(Q) \quad \forall P, Q \in E(\mathbb{F}_q)$.

Die letzte Eigenschaft stellt sicher, dass die Ordnung eines Punktes bei Anwendung von $\phi(x, y)$ erhalten bleibt. Gleichzeitig ist α so gewählt, dass $\alpha \cdot x \notin \mathbb{F}_q$, somit ist $\phi(x, y) \notin E(\mathbb{F}_q)$.

Mit Hilfe der Distortion-Map wird ein modifiziertes Weil-Pairing $\hat{e}$ definiert:

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T, \quad \hat{e}(Q_1, Q_2) \mapsto e_p(Q_1, \phi(Q_2)) \tag{4.9}$$

Da $\phi(Q_2)$ nicht in $E(\mathbb{F}_q)$ liegt, wird durch die Distortion-Map gewährleistet, dass $\phi(Q_2)$ kein Vielfaches von Q_2 ist, und damit auch kein Vielfaches von Q_1. Dadurch ist das Weil-Pairing e_p nicht degeneriert, dementsprechend ist auch $\hat{e}$ eine nicht degenerierte, bilineare

Abbildung und kann für die Gruppensignatur genutzt werden. Die Nicht-Degeneriertheit soll noch einmal explizit gezeigt werden:
Sei $P \in \mathbb{G}_1$, $Q = \phi(P) \in E(\mathbb{F}_{q^2})[p] \backslash \mathbb{G}_1$, $P, Q \neq \mathcal{O}$. Nach Gleichung (4.7) auf Seite 48 existieren $T_1, T_2 \in E(\mathbb{F}_{q^2})[p]$ und eindeutige $a, b, c, d \in \mathbb{Z}_p$, so dass $P = aT_1 + bT_2$, $Q = cT_1 + dT_2$. Ohne Beschränkung der Allgemeinheit sei $a \neq 0$. Dann ist $P = aT_1 + bT_2 \Leftrightarrow T_1 = \frac{1}{a}P - \frac{b}{a}T_2$. Damit existieren auch eindeutige $\hat{c}, \hat{d}$, sodass $Q = cT_1 + dT_2 = \hat{c}P + \hat{d}T_2$. Da Q kein Vielfaches von P ist, gilt $\hat{d} \neq 0$.

Damit $\hat{e}$ nicht degeneriert ist, muss gelten $\hat{e}(g_1, g_2) \neq 1$ (siehe S. 16). Sei $g_1 = P, g_2 = P$. Angenommen $\hat{e}(P, P) = 1$, so ist:

$$
\begin{aligned}
1 = \hat{e}(P,P) &= e_p(P,Q) \\
&= e_p(P, \hat{c}P + \hat{d}T_2) && \text{Bilinearität des Weil-Pairings} \\
&= e_p(P, \hat{c}P) \cdot e_p(P, \hat{d}T_2) && \text{Bilinearität des Weil-Pairings} \\
&= e_p(P,P)^{\hat{c}} \cdot e_p(P,T_2)^{\hat{d}} && e_p(P,P) = 1 \\
&= 1^{\hat{c}} \cdot e_p(P,T_2)^{\hat{d}} \\
&= e_p(P,T_2)^{\hat{d}}
\end{aligned}
$$

Da $\hat{d} \neq 0$ müsste $e_p(P, T_2) = 1$ sein. Als Konsequenz wäre dann aber $e_p(P, R) = 1$ für alle $R \in E(\mathbb{F}_{q^2})[p]$, das Weil-Pairing ist auf $E(\mathbb{F}_{q^2})[p]$ aber gerade nicht degeneriert (siehe Kapitel 4.5.1). Somit erzeugt die Annahme einen Widerspruch, daher ist $\hat{e}(P, P) \neq 1$.

Auch das Hashen nach $\mathbb{G}_2$ ist jetzt einfacher möglich, wie in Kapitel 4.6 gezeigt werden wird. Ein weiterer großer Vorteil des modifizierten Weil-Pairings ist auch, dass durch die Wahl von $\mathbb{G}_2 = \mathbb{G}_1$ sowohl die Elemente aus $\mathbb{G}_1$ als auch die Elemente aus $\mathbb{G}_2$ Elemente aus der kleineren Gruppe $E(\mathbb{F}_q)$ und nicht aus der viel größeren Gruppe $E(\mathbb{F}_{q^2})$ sind. Somit sind die Berechnungen im Gruppensignatur-Schema für Elemente aus $\mathbb{G}_2$ jeweils nur Operationen in $E(\mathbb{F}_q)$ und nicht die kostspieligeren Operationen in $E(\mathbb{F}_{q^2})$. Einzig für die Berechnung des Weil-Pairings muss man zwischenzeitlich in $E(\mathbb{F}_{q^2})$ arbeiten. Daher wird für die Umsetzung des Gruppensignatur-Schemas das modifizierte Weil-Pairing $\hat{e}$ benutzt.

4.5.6. Anmerkung zur Länge von p und q

Bisher wurde noch nicht erklärt, warum für die Bitlängen von p und q 160 Bit beziehungsweise 512 Bit gewählt wurde. Für p wurde eine Länge von 160 Bit gewählt, damit in der Gruppe $\mathbb{G}_1$ der diskrete Logarithmus auf elliptischen Kurven nicht effizient berechnet werden kann. Diese Länge gilt als ausreichend sicher für Verfahren zur Berechnung des diskreten Logarithmus, die auf elliptischen Kurven basieren [FST06, Kap. 1].

Die Wahl der Länge von q ergibt sich durch folgenden Zusammenhang: Die erste Anwendung des Weil-Pairings war der sogenannte MOV-Algorithmus [HPS08, S. 326ff]. Durch das Weil-Pairing wird $\mathbb{G}_1$ in $\mathbb{F}^*_{q^2}$ eingebettet. Der MOV-Algorithmus benutzt dies, um den diskreten Logarithmus in $\mathbb{F}^*_{q^2}$ mit Hilfe des Index-Calculus-Algorithmus zu berechnen und kann anhand dieses Ergebnisses zusammen mit der Bilinearität des Weil-Pairings auch den diskreten Logarithmus in $\mathbb{G}_1 \subset E(\mathbb{F}_q)$ berechnen. Dazu geht er wie folgt vor: Seien $E, \mathbb{G}_1, \mathbb{G}_T$ und $\hat{e}$ definiert wie im vorherigen Abschnitt. Seien $P, Q \in \mathbb{G}_1$

mit $Q = aP$ für ein $a \in \mathbb{Z}$. Berechne $b = \hat{e}(P, P)$ und $c = \hat{e}(P, Q) = \hat{e}(P, aP) = \hat{e}(P, P)^a = b^a$. Kann man $log_b(c) = a$ in $\mathbb{F}_{q^2}$ berechnen, so erhält man dadurch auch die Lösung von $log_P(Q) = a$ in $E(\mathbb{F}_q)$.

Für allgemeine Kurven ist dies keine Gefahr, da dort der Einbettungsgrad k sehr groß ist und damit $\mathbb{F}_{q^k}$ so groß wird, dass man dort nicht mehr effizient rechnen kann.

Die vorliegende Kurve E wurde allerdings unter anderem deshalb gewählt, weil sie einen kleinen Einbettungsgrad hat, um so das Weil-Pairing möglichst effizient berechnen zu können. Neben einem kleinen Einbettungsgrad möchte man aus der Sicht einer schnellen Berechnung auch q möglichst klein wählen (in etwa in der Größenordnung von p), damit man „kleine“ Körper $\mathbb{F}_q$ und $\mathbb{F}_{q^2}$ erhält und dort sehr schnell rechnen kann. Genau dann ist allerdings auch der MOV-Algorithmus eine Gefahr für die Sicherheit des gesamten Verfahrens, da er für ein derart gewähltes q den diskreten Logarithmus in $\mathbb{F}_{q^2}$ berechnen kann und damit auch in $E(\mathbb{F}_q)$, womit die gesamte Konstruktion hinfällig ist.

Daher ist eine Kombination gesucht, bei der das Weil-Pairing noch halbwegs effizient berechnet werden kann, $\mathbb{F}_{q^2}$ allerdings so groß ist, dass der MOV-Algorithmus keine Gefahr darstellt. Hat q eine Länge von 512 Bit, so haben die Elemente aus $\mathbb{F}_{q^2}$ eine Länge von 1024 Bit. Dies wird aktuell noch als hinreichend sicher in Bezug auf den Index-Calculus-Algorithmus angesehen, so dass der MOV-Algorithmus keine Gefahr darstellt [FST06, Kap. 1]. Als Konsequenz erfolgen dann allerdings auch sämtliche Operationen in $E(\mathbb{F}_q)$ über einem relativ großen Körper $\mathbb{F}_q$, sodass die Operationen entsprechend langsam sind.

4.6. Hashfunktion H_0

Die Gruppensignatur erfordert eine Hashfunktion H_0, die eine Abbildung von $\{0,1\}^*$ nach $\mathbb{G}_2 \times \mathbb{G}_2$ durchführt und natürlich die Anforderungen an eine kryptografische Hashfunktion erfüllt. Die Anforderung liest sich erst einmal wenig spektakulär, aber bei genauerem Hinsehen stellt sich die Frage, wie man überhaupt in ein Element aus $\mathbb{G}_2$ hashen kann. Es sei noch einmal daran erinnert, dass bei unserer Wahl der Gruppen $\mathbb{G}_1 = \mathbb{G}_2$ gilt.

Ein erster Schritt könnte so aussehen, eine allgemeine kryptografische Hashfunktion zu benutzen und den Hashwert h modulo q zu nehmen, um ein Element aus $\mathbb{F}_q$ zu erhalten. Multipliziert man dieses Element mit dem Generator $g_2 \in \mathbb{G}_2$, so erhält man ein eindeutiges Element $L \in \mathbb{G}_2$ und das Problem scheint gelöst zu sein.

Allerdings handelt man sich bei diesem Vorgehen ein neues Problem ein. Das Diskreter-Logarithmus-Problem ist für L sehr einfach zu lösen, es ist nämlich gerade $h \mod q$, der diskrete Logarithmus ist also öffentlich bekannt. Dieses würde die Sicherheit des Verfahrens gefährden, daher muss ein anderes Vorgehen gewählt werden.

Der folgende Ansatz wird in [BF03, Kap. 5.2] und [Sco05b] beschrieben. Zuerst berechnet man wieder ein $h_q = h \mod q$ mit Hilfe einer allgemeinen kryptografischen Hashfunktion. Anschließend benutzt man dasselbe Verfahren wie in Abschnitt 4.4, nur nimmt man anstelle einer Zufallszahl den Wert h_q. Man setzt $y := h_q$ und löst wie zuvor die Gleichung $x^3 = y^2 - 1$. Mit $R := (x, y)$ berechnet man $L = \frac{q+1}{p} \cdot R$ und erhält einen Punkt der Ordnung p. In Abschnitt 4.4 wurde gezeigt, dass jeder Punkt der Ordnung p in $\mathbb{G}_1$ beziehungsweise $\mathbb{G}_2$ liegt. Damit ist der gefundene Punkt L zum einen ein Element aus $\mathbb{G}_2$, zum anderen ist der diskrete Logarithmus von L in Bezug auf g_2 nicht bekannt.

Ein theoretischer Fall muss allerdings noch behandelt werden: Der Punkt R könnte zufällig Ordnung $\frac{q+1}{p}$ haben, damit wäre $L = \mathcal{O}$. In Abschnitt 4.4 wurde schon gezeigt, dass dieser Fall praktisch nicht vorkommt. Für eine saubere Definition muss allerdings auch dieser Fall behandelt werden. In [BF03, Kap. 5.2] wird vorgeschlagen, in diesem Fall den Startwert mehrfach zu hashen, eventuell sogar mit unterschiedlichen Hashfunktionen, um so einen anderen Hashwert h zu erhalten und damit zu einem anderen Punkt zu gelangen.

5. Implementierung und Evaluierung

5.1. Beschreibung der Zielplattform

Der ausgewählte Halbleiter ist ein Samsung ARM Chip und trägt die Typenbezeichnung S3FS9CI. Er besitzt 32 KB ROM, 768 KB Flash-Speicher und 50 KB RAM, wobei davon 2 KB als Crypto-RAM reserviert sind. Als Prozessor kommt eine 32-Bit CPU zum Einsatz und arbeitet mit einer internen Taktfrequenz von bis zu 20 MHz.

Der Halbleiter zeichnet sich insbesondere durch seine Vielzahl an Schnittstellen zur Kommunikation mit der Außenwelt aus. Zum einen unterstützt er das in der Chipkartenwelt übliche serielle I/O nach ISO 7816 für die Protokolle T=0 und T=1 zur Kommunikation mit einem Standard-Kartenleser. Zusätzlich unterstützt er aber auch die High-Speed-Protokolle USB, MMC und das Single-Wire-Protokoll (SWP). Gerade durch die Unterstützung von SWP sind neuartige Produkte entstanden, an die vor ein paar Jahren noch nicht zu denken war, so zum Beispiel im NFC-Umfeld[1] (Near Field Communication).

Aus kryptografischer Sicht von Interesse ist die Hardwareunterstützung von DES, Triple-DES und AES. Zusätzlich enthält der Halbleiter eine Einheit zur Berechnung einer Montgomery-Multiplikation namens TORNADO-II™, die für RSA-Berechnungen vorgesehen ist. Zusätzlich bietet sie auch noch weitere Funktionalitäten an, auf die in Kapitel 5.4 eingegangen wird.

Besonders auffällig an den Daten ist zum einen der relativ große Flash-Speicher, vor allem aber das extrem große RAM mit 50 KB. Üblich für so einen Chip sind eher 8–12 KB. Durch die großzügige Ausstattung eignet sich der Chip sehr gut zur Prototypenentwicklung, da man der Optimierung der Codegröße und des Speicherbedarfs nicht die oberste Priorität einräumen muss, sondern sich erst einmal auf die Umsetzung der geforderten Funktionalität konzentrieren kann.

Flash-Halbleiter bieten außerdem den Vorteil, dass man den Programmcode nach der Entwicklung und Test auf einem Emulator anschließend auch direkt in die echte Smart Card laden kann. Im ROM der Smart Card existiert dazu ein sogenannter Bootloader, mit dessen Hilfe der Programmcode und die nötigen Daten in den Flash-Speicher der Smart Card geladen werden können. Anschließend wird der geladene Programmcode aktiviert und kann direkt genutzt beziehungsweise getestet werden. Für klassische ROM/EEPROM-Halbleiter funktioniert so etwas nicht, dort muss mit dem fertigen Programmcode erst in einem mehrwöchigen Prozess beim Halbleiterhersteller eine ROM-Maske erstellt werden, die dann den Programmcode unveränderbar im ROM des Chips enthält.

[1] http://www.sagem-orga.com/index.php?myELEMENT=All-rounderNearFieldCommunication:

Betriebssystem

Das Chipkartenbetriebssystem *SIMbiOS* der Firma Sagem Orga besteht aus einer Vielzahl hierarchisch gegliederter Komponenten. Auf unterster Ebene liegt der sogenannte Microkernel. Eine der Hauptaufgaben des Microkernels ist die Kapselung der Zugriffe auf die Hardware und Bereitstellung einer plattformunabhängigen Programmierschnittstelle für die darüber liegenden Komponenten. Da für die Berechnung der Gruppensignatur zwingend Zugriff auf die Hardware des Samsung ARM nötig ist, wurden sämtliche für die Berechnung notwendigen Funktionalitäten als Erweiterung des Microkernels eingebracht.

5.2. Testumgebung

Die Smart Card wurde mit dem Tool UTE (Universal Test Environment) getestet. UTE ist ein proprietäres Tool der Firma Sagem Orga zum Testen von Chipkarten. Es ist in Java geschrieben und unterstützt neben diversen Kartenlesern auch die Ansteuerung von unterschiedlichen Simulationsumgebungen, die von einigen Chip-Herstellern bereitgestellt werden. Mit der UTE werden Testtreiber ausgeführt, die ebenfalls in Java geschrieben sind. In ihnen werden einzelne Kommandos spezifiziert, die dann nach und nach an die Chipkarte übertragen werden.

Für Tests des Microkernels wird ein spezielles Protokoll benutzt, mit dem es möglich ist, direkt einzelne Funktionen anzusteuern. Dazu werden erst die benötigten Parameter an die Chipkarte übermittelt, bevor die eigentliche Funktion aufgerufen wird. Dieses Verfahren wurde genau so für die Tests der Gruppensignatur benutzt. So war es möglich, Schritt für Schritt die einzelnen Funktionalitäten wie modulare Arithmetik, elliptische Kurvenarithmetik, Weil-Pairing und schlussendlich die komplette Berechnung der Gruppensignatur zu testen.

5.3. Datentypen und ihre Übertragung zur Smart Card

Der Basisdatentyp für Elemente aus $\mathbb{F}_q$ ist eine lange Integerzahl der Byte-Länge $\text{qLen} = \lceil \log(q)/8 \rceil$. Wie in Abschnitt 4.1.2 erläutert, besteht ein Element aus $\mathbb{F}_{q^2}$ aus den zwei Komponenten $a, b \in \mathbb{F}_q$. Dieses wurde auch im Code so umgesetzt, beides sind lange Integerzahlen der Byte-Länge qLen. Auf eine spezielle Kodierung bei der Übertragung vom Kartenleser zur Karte und umgekehrt wurde verzichtet, a und b werden einfach konkateniert und als Bytearray der Länge $2 \cdot \text{qLen}$ übertragen. Der Empfänger muss im Vorhinein wissen, dass es sich um ein Element aus $\mathbb{F}_{q^2}$ handelt, um den Wert korrekt zu interpretieren. Auf gleiche Art und Weise werden auch die Punkte aus $E(\mathbb{F}_q)$ übertragen, die aus den zwei Koordinaten $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$ bestehen. Auf Verfahren zur Punktkompression, die einen Punkt verkürzt darstellen, wurde erst einmal verzichtet.

5.4. Arithmetik in $\mathbb{F}_q$ und $\mathbb{F}_{q^2}$

Der Samsung ARM Halbleiter stellt die modulare Arithmetik und damit die Arithmetik über $\mathbb{F}_q$ in Form einer Crypto-Toolbox bereit. Für den Modul werden Bitlängen bis 2048 Bit unterstützt. Für die Umsetzung der Gruppensignatur ist das ausreichend, da q

maximal eine Länge von 512 Bit hat. Die Toolbox stellt Funktionen für die modulare Addition, Subtraktion, Multiplikation, Exponentiation und zum Berechnen eines inversen Elements bereit. Der erste Schritt der Implementierung bestand daher darin, Elemente aus $\mathbb{F}_q$ aus dem externen Format in Form eines Bytearrays in das für die Toolbox vorgegebene interne Format umzuwandeln und damit die entsprechenden Toolbox-Funktionen anzusteuern. Die Ergebnisse der Funktionen müssen entsprechend wieder vom internen in das externe Format konvertiert werden, bevor sie als Antwort von der Smart Card an den Kartenleser übertragen werden können.

Operationen in $\mathbb{F}_q$

Die Funktionen der Toolbox für die modulare Multiplikation und Exponentiation verwenden intern den Crypto-Coprozessor des Chips namens TORNADO-II™. Dessen primäre Aufgabe ist die schnelle Berechnung einer modularen Exponentiation basierend auf der Montgomery-Multiplikation [HMV04, 2.2.4] [Mon85].

Die modulare Multiplikation wurde vom Halbleiterhersteller wie folgt realisiert: Zuerst werden die Parameter ins Crypto-RAM kopiert, dort vom internen Format in ein spezielles Montgomery-Format konvertiert, die Montgomery-Multiplikation wird durchgeführt und anschließend wird das Format wieder in das interne Format konvertiert. Dies alles geschieht mit speziellen Funktionen des Coprozessors.

Vor dem Start der Implementierung wurde schon vermutet, dass einzig die Ausnutzung der Hardwarebeschleunigung bei der Multiplikation die Chance bietet, halbwegs akzeptable Zeiten für die Berechnung der Gruppensignatur zu erzielen, denn die Multiplikation ist die Basisfunktionalität für die meisten Berechnungen der Gruppensignatur. Die Zeit, die hier verloren geht, lässt sich später nicht wieder kompensieren.

Leider bietet der Coprozessor und damit auch die Toolbox keine Schnittstelle an, bei der die Parameter schon im Montgomery-Format vorliegen, denn gerade wenn mehrere Multiplikationen hintereinander durchgeführt werden, könnte man auf das ständige Konvertieren verzichten, womit die Multiplikation vermutlich noch schneller laufen würde. Die Optimierung der Hardware ist allerdings kein Ziel dieser Diplomarbeit, hier wird die Hardware als gegeben vorausgesetzt.

Ferner bietet die Toolbox auch keine spezielle Funktion zum Quadrieren eines Elementes. Daher muss für die Quadrierung die normale Multiplikationsfunktion verwendet werden. Als Folge davon gilt für die Abschätzungen $S = M$. Des weiteren haben Messungen ergeben, dass auf dem Samsung ARM die Inversion bei einer Bitlänge von 512 Bit etwa soviel Zeit benötigt wie fünfzehn Multiplikationen, also $I \approx 15M$.

Die Operationen in $\mathbb{Z}_p$, die für die Gruppensignatur benötigt werden, lassen sich auf gleiche Art und Weise mit den Funktionen der Toolbox umsetzen, der Modul ist dann entsprechend p und nicht q.

Operationen in $\mathbb{F}_{q^2}$

Ein Element aus $\mathbb{F}_{q^2}$ besteht aus den zwei Elementen $a, b \in \mathbb{F}_q$. In Kapitel 4.1.3 wurde beschrieben, wie sich die Operationen in $\mathbb{F}_{q^2}$ durch Operationen in $\mathbb{F}_q$ realisieren lassen. So sind zum Beispiel für eine Addition zweier Elemente $x, y \in \mathbb{F}_{q^2}$ zwei Additionen in $\mathbb{F}_q$ nötig. Entsprechend den Formeln wurden die Operationen in $\mathbb{F}_{q^2}$ mit den Funktionen der Crypto-Toolbox zum Rechnen in $\mathbb{F}_q$ realisiert.

Für die Berechnung des Wertes R_2 bei der Erstellung einer Gruppensignatur wird auch eine Exponentiation in $\mathbb{F}_{q^2}$ benötigt. Dafür wurde ein einfacher Square-And-Multiply-Algorithmus realisiert, der intern als Operationen die Quadrierung und Multiplikation in $\mathbb{F}_{q^2}$ benutzt.

Im Rahmen des Miller-Algorithmus werden Operationen zwischen Elementen aus $\mathbb{F}_q$ und $\mathbb{F}_{q^2}$ benötigt. Sei $x \in \mathbb{F}_q, y = (y_a, y_b) \in \mathbb{F}_{q^2}$. Eine Möglichkeit zur Umsetzung wäre, x in ein Element $(x, 0) \in \mathbb{F}_{q^2}$ umzuwandeln und anschließend die entsprechende Operation in $\mathbb{F}_{q^2}$ anzuwenden. Diese Umwandlung verschwendet aber zum einen unnötigen Speicherplatz, zum anderen müssen Additionen, Subtraktion oder Multiplikationen mit 0 gar nicht erst durchgeführt werden, da das Ergebnis schon vorher klar ist. Die Toolbox-Funktionen behandeln die 0 allerdings wie jede andere Zahl, so benötigt eine Multiplikation mit 0 genauso lange wie eine Multiplikation mit einer beliebigen anderen Zahl. Daher wurden spezielle Funktionen geschaffen, die bei Operationen mit Elementen aus $\mathbb{F}_q$ und $\mathbb{F}_{q^2}$ nur die wirklich nötigen Operationen ausführen. So ergibt sich zum Beispiel für die Multiplikation: $x \cdot (y_a, y_b) = (xy_a, xy_b)$. Es werden also nur zwei Multiplikationen in $\mathbb{F}_q$ benötigt, bei einer normalen Multiplikation $(x, 0) \cdot (y_a, y_b)$ wären es drei gewesen.

5.5. Arithmetik in $E(\mathbb{F}_q)$ und $E(\mathbb{F}_{q^2})$

Die Arithmetik in $E(\mathbb{F}_q)$ wurde für das affine, projektive und Jacobische Koordinatensystem realisiert. Die Formeln für die Punktaddition und Punktmultiplikation für das jeweilige Koordinatensystem aus Kapitel 4.2.1 enthalten ausschließlich Operationen in $\mathbb{F}_q$, die entsprechend mit den Funktionen aus dem vorherigen Kapitel umgesetzt wurden. Für die Berechnungen im projektiven und Jacobischen Koordinatensystem müssen etliche Zwischenergebnisse in temporären Variablen gespeichert werden. Hier galt es eine Berechnungsreihenfolge zu finden, die den Bedarf an temporären Variablen möglichst minimiert, um so den Speicherbedarf niedrig zu halten.

Die Optimierung mit der gemischten Addition (siehe Seite 41) wurde derart umgesetzt, dass in der Additionsfunktion im projektiven und Jacobischen System anfangs überprüft wird, ob die Z-Koordinate eins ist. In diesem Fall werden die entsprechenden Multiplikationen mit Z ausgelassen. Für die Optimierung bei der Punktverdopplung wird schon beim Einrichten der Kurve überprüft, ob $a = 0$ gilt und dies in einem extra Flag vermerkt. Bei der Punktverdopplung wird dieses Flag ausgewertet und bei $a = 0$ entsprechend die Multiplikationen aZ^2 beziehungsweise aZ^4 eingespart.

Für die Punktmultiplikation wurde entsprechend Algorithmus 1 von Seite 41 für jedes Koordinatensystem ein Double-And-Add-Algorithmus realisiert, der die jeweilige optimierte Punktaddition und Punktverdopplung benutzt. Die Eingabe- und Ausgabeparameter sind grundsätzlich Punkte im affinen Koordinatensystem. Im Double-And-Add-Algorithmus für projektive und Jacobische Koordinatensystem werden die Punkte als erster Schritt in das jeweilige System konvertiert, dann wird die Punktmultiplikation durchgeführt und am Ende wird das Ergebnis wieder zurück ins affine Koordinatensystem umgewandelt.

Zusätzlich dazu wurde für jedes Koordinatensystem auch noch ein Double-And-Add-Algorithmus realisiert, der erst die NAF-Darstellung des Skalars berechnet (vgl. Algorithmus 2 auf Seite 43) und dann entsprechend Algorithmus 3 eine Punktmultiplikation

Double-And-Add, Algorithmus 1	Min	Max	∅
$t(i \cdot \mathcal{A})$	1301 ms	1543 ms	1429 ms
$t(i \cdot \mathcal{P})$	742 ms	889 ms	819 ms
$t(i \cdot \mathcal{J})$	601 ms	745 ms	675 ms
Double-And-Add-NAF, Algorithmus 3	Min	Max	∅
$t(i \cdot \mathcal{A})$	1182 ms	1361 ms	1289 ms
$t(i \cdot \mathcal{P})$	672 ms	779 ms	736 ms
$t(i \cdot \mathcal{J})$	536 ms	634 ms	594 ms

Tabelle 5.1.: Aufwand zur Berechnung einer Punktmultiplikation in $E(\mathbb{F}_q)$

durchführt.

Für die Werte in Tabelle 5.1 wurden jeweils 1000 Durchläufe der jeweiligen Algorithmen zur Punktmultiplikation $i \cdot P$ auf dem Samsung ARM Halbleiter ausgewertet. Die Primzahlen p, q wurden zufällig entsprechend ihrer Vorgaben aus Kapitel 4.3 gewählt. Der Punkt P wurde zufällig entsprechend Kapitel 4.4 konstruiert, i ist ein zufälliges Element aus $\mathbb{Z}_p$. Für diese und für alle weiteren Versuche wurde die interne Taktfrequenz auf 20 MHz eingestellt. Der schnellste Double-And-Add-Algorithmus ist derjenige im Jacobischen Koordinatensystem. Er ist im Durchschnitt um mehr als den Faktor zwei schneller als die Punktmultiplikation im affinen System. Gegenüber dem projektiven System ist er immerhin noch um gut 17% schneller. Betrachtet man die Version mit der NAF-Darstellung, so ist sie noch einmal 12% schneller gegenüber der Standard Double-And-Add-Version. Dieser Gewinn liegt geringfügig unterhalb der theoretischen Abschätzung: Mit den Werten aus Tabelle 4.1 von Seite 44 benötigt eine Punktaddtion im Jacobischen System $t(\mathcal{J} = \mathcal{J} + \mathcal{A}) = 8M + 3S$, eine Punktverdopplung $t(2\mathcal{J}) = t^*(2\mathcal{J}) = 3M + 4S$. i hat eine Bitlänge von 160, daher werden im Mittel beim Double-And-Add-Algorithmus 160 Punktverdopplungen und $\frac{160}{2}$ Punktadditionen durchgeführt. Mit $S = M$ ergibt sich:

$$t(i \cdot \mathcal{J}) = 160(3M+4S)+80(8M+3S) = 160 \cdot 7M+80 \cdot 11M = 1120M+880M = 2000M$$

Bei der NAF-Darstellung reduziert sich die Anzahl der Punktadditionen im Mittel auf $\frac{160}{3}$, gegenüber $\frac{160}{2}$ werden dadurch ein Drittel der Punktadditionen eingespart. Damit ergibt sich:

$$t(i \cdot \mathcal{J}) = 160(3M + 4S) + \frac{160}{3}(8M + 3S) \approx 1120M + 587M \approx 1707M$$

Nach der Abschätzung wäre insgesamt ein Gewinn von etwa 14% zu erwarten, allerdings wird dort auch nicht berücksichtigt, dass die Berechnung der NAF ebenfalls etwas Zeit benötigt.

Insgesamt benötigt die NAF-Version im Jacobischen Koordinatensystem im Durchschnitt 58% weniger Zeit als die einfache Double-And-Add-Variante im affinen System. Die Abweichungen nach oben und unten liegen im Bereich von 10%. Die 58% sind sozusagen der Gesamtgewinn, der durch die Optimierungen erreicht wurde.

Double-And-Add	Min	Max	∅
$t(i \cdot \mathcal{A})$	2123 ms	2504 ms	2324 ms
$t(i \cdot \mathcal{J})$	1758 ms	2192 ms	1974 ms
Double-And-Add-NAF	Min	Max	∅
$t(i \cdot \mathcal{A})$	1922 ms	2216 ms	2100 ms
$t(i \cdot \mathcal{J})$	1495 ms	1827 ms	1700 ms

Tabelle 5.2.: Aufwand zur Berechnung einer Punktmultiplikation in $E(\mathbb{F}_{q^2})$

Operationen in $E(\mathbb{F}_{q^2})$

Die Arithmetik in $E(\mathbb{F}_{q^2})$ wurde für das affine und das Jacobische Koordinatensystem realisiert. Da im projektiven Koordinatensystem sowohl die Punktaddition als auch die Punktverdopplung nach den Betrachtungen in Kapitel 4.2.2 langsamer sind als im Jacobischen Koordinatensystem, wurden die Operationen für das projektive Koordinatensystem gar nicht erst umgesetzt. Ansonsten wurden die gleichen Algorithmen wie im vorherigen Abschnitt umgesetzt, mit dem einzigen Unterschied, dass als Operationen die entsprechenden Funktionen für den Körper $\mathbb{F}_{q^2}$ aufgerufen werden, anstatt für den Körper $\mathbb{F}_q$.

Die Werte in Tabelle 5.2 wurden wie zuvor durch 1000 Durchläufe der entsprechenden Algorithmen erzeugt. Die Parameter wurden auf die gleiche Art zufällig erzeugt wie im vorherigen Abschnitt, wobei der zu multiplizierende Punkt jetzt ein zufälliger Punkt aus $E(\mathbb{F}_{q^2})[p]$ war.

Als erstes fällt auf, dass der Unterschied zwischen affinem und Jacobischem System deutlich geringer ist als noch in $E(\mathbb{F}_q)$, er beträgt nur noch 15%. In Kapitel 4.2.2 wurde vermutet, dass die Punktmultiplikation im Jacobischen System in $E(\mathbb{F}_{q^2})$ etwa um den Faktor 3 langsamer ist als in $E(\mathbb{F}_q)$, dies bestätigt sich in den Messungen. Für das affine System beträgt der Faktor nur circa 1,6. Damit schneidet das affine System in $E(\mathbb{F}_{q^2})$ besser ab, als man es vermutet hätte.

Die Variante mit der NAF-Darstellung gewinnt im affinen System 10%, im Jacobischen System sogar fast 15%. Im Gegensatz zum affinen System sind hier die Punktadditionen im Verhältnis zu den Punktverdopplungen deutlich teurer, daher gewinnt man durch die Einsparungen der Punktadditionen prozentual mehr Zeit. Der Unterschied zwischen der NAF-Variante im Jacobischen System und dem Double-And-Add-Algorithmus im affinen System beträgt im Durchschnitt etwa 27%, der Gesamtgewinn durch die Optimierungen ist hier also geringer als in $E(\mathbb{F}_q)$, wo der Unterschied bei 58% lag.

5.6. Miller-Algorithmus

Für den Miller-Algorithmus wurde als erstes Algorithmus 4 von Seite 55 implementiert. Entsprechend Kapitel 4.5.5 seien $P, Q' \in \mathbb{G}_1 \subset E(\mathbb{F}_q)$ und sei $Q = \phi(Q') \in E(\mathbb{F}_{q^2})[p] \backslash E(\mathbb{F}_q)$. P und Q haben beide Ordnung p. Wählt man $R \in \mathbb{G}_1, R \neq P, \mathcal{O}$, so ist R weder Nullstelle noch Polstelle von f_Q, denn alle Punkte, die im Divisor $div(f_Q)$ vorkommen, sind aus $E(\mathbb{F}_{q^2}) \backslash E(\mathbb{F}_q)$. Analog wählt man $S \in E(\mathbb{F}_{q^2})[p] \backslash E(\mathbb{F}_q), S \neq Q, \mathcal{O}$, dann ist S weder Nullstelle noch Polstelle von f_P, denn alle Punkte, die im Divisor $div(f_P)$ vorkommen, sind Elemente aus $E(\mathbb{F}_q)$ (siehe auch [ASN07, Kap. 2.2]).

Algorithmus 5: Zwischenversion des Miller-Lite-Algorithmus zum leichteren Verständnis von Algorithmus 6

Input : $P, R \in E(\mathbb{F}_q)[p],\ Q, S \in E(\mathbb{F}_{q^2})[p],\ p = p_t, p_{t-1}, \ldots, p_0$
Output : $f = f_P(\mathcal{A}_Q) = \frac{f_P(Q+S)}{f_P(S)} \in \mathbb{F}_{q^2}^*$

$\hat{f}_1 = V_{P+R}(Q+S)/L_{P,R}(Q+S)$
$\hat{f} = \hat{f}_1$
$\bar{f}_1 = V_{P+R}(S)/L_{P,R}(S)$
$\bar{f} = \bar{f}_1$
$Z = P$
for $i = t-1$ **downto** 0 **do**
    $\hat{f} = \hat{f}^2 \cdot T_Z(Q+S)/V_{2Z}(Q+S)$
    $\bar{f} = \bar{f}^2 \cdot T_Z(S)/V_{2Z}(S)$
    $Z = 2Z$
    if $m_i = 1$ **then**
        $\hat{f} = \hat{f} \cdot \hat{f}_1 \cdot L_{Z,P}(Q+S)/V_{Z+P}(Q+S)$
        $\bar{f} = \bar{f} \cdot \bar{f}_1 \cdot L_{Z,P}(S)/V_{Z+P}(S)$
        $Z = Z + P$
return $f = \hat{f}/\bar{f}$

Mit dieser Wahl lässt sich $e_p(P,Q) = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)} = \frac{f_P(Q+S) \cdot f_Q(R)}{f_P(S) \cdot f_Q(P+R)}$ berechnen. Für den Miller-Algorithmus benötigt man im Prinzip zwei Ausprägungen: Variante A wird benötigt, um $f_P(\mathcal{A}_Q) = \frac{f_P(Q+S)}{f_P(S)}$ zu berechnen, Variante B wird benötigt, um $f_Q(\mathcal{A}_P) = \frac{f_Q(P+R)}{f_Q(R)}$ zu berechnen.

Bei der Berechnung von f_P laufen im Miller-Algorithmus die Punktverdopplungen und Punktadditionen über den Punkt P, es sind Operationen in $E(\mathbb{F}_q)$. Um den Code einfach zu halten wurden die Punktverdopplung und Punktaddition erst einmal nur mit affinen Koordinaten realisiert. In den Geradengleichungen L, T, V sind die Koeffizienten aus $\mathbb{F}_q$, während X, Y aus $\mathbb{F}_{q^2}$ sind (da $Q, S \in E(\mathbb{F}_{q^2})$). Dieser Durchlauf wird in der Literatur auch mit „Miller-Lite“ bezeichnet.

Bei der Berechnung von f_Q ist es hingegen quasi umgekehrt. Die Punktverdopplungen und Punktadditionen laufen über den Punkt Q, die Operationen finden also in der großen Gruppe $E(\mathbb{F}_{q^2})$ statt und sind damit deutlich langsamer. In den Geradengleichungen L, T, V sind die Koeffizienten jetzt aus $\mathbb{F}_{q^2}$, während X, Y aus $\mathbb{F}_q$ sind (da $P, R \in E(\mathbb{F}_q)$). Dieser Durchlauf wird in der Literatur auch mit „Miller-Full“ bezeichnet.

Unter den genannten Voraussetzungen lässt sich das Weil-Pairing jetzt berechnen, indem man zweimal die Miller-Lite-Variante zur Berechnung von $f_P(Q+S)$ und $f_P(S)$ durchführt und zweimal die Miller-Full-Variante zur Berechnung von $f_Q(P+R)$ und $f_Q(R)$.

Auch wenn die Laufzeit mit diesem Ansatz insgesamt noch sehr hoch war, wurde damit zumindest schon einmal das korrekte Ergebnis für das Weil-Pairing berechnet. Eine Laufzeitanalyse erfolgt in Abschnitt 5.6.2 anhand einer optimierten Version.

Algorithmus 6: 2. Optimierte Version des Miller-Lite-Algorithmus

Input : $P, R \in E(\mathbb{F}_q)[p],\ Q, S \in E(\mathbb{F}_{q^2})[p],\ p = p_t, p_{t-1}, \ldots, p_0$
Output : $f = f_P(\mathcal{A}_Q) = \frac{f_P(Q+S)}{f_P(S)} \in \mathbb{F}_{q^2}^*$

$g_1 = V_{P+R}(Q+S) \cdot L_{P,R}(S)$
$h_1 = L_{P,R}(Q+S) \cdot V_{P+R}(S)$
$g = g_1$
$h = h_1$
$Z = P$
for $i = t-1$ **downto** 0 **do**
    $g = g^2 \cdot T_Z(Q+S) \cdot V_{2Z}(S)$
    $h = h^2 \cdot V_{2Z}(Q+S) \cdot T_Z(S)$
    $Z = 2Z$
    if $m_i = 1$ **then**
        $g = g \cdot g_1 \cdot L_{Z,P}(Q+S) \cdot V_{Z+P}(S)$
        $h = h \cdot h_1 \cdot V_{Z+P}(Q+S) \cdot L_{Z,P}(S)$
        $Z = Z + P$
return $f = g/h$

5.6.1. Implementierte Optimierungen für den Miller-Algorithmus

Folgende Optimierungen wurden für den Miller-Algorithmus umgesetzt:

1. Gleichzeitige Berechnung von $f_P(Q+S)$ und $f_P(S)$ in einem Funktionsaufruf.
2. Einsparung der finalen Division $\frac{f_P(Q+S)}{f_P(S)}$ durch Verschiebung in die Operationen innerhalb der Schleife.
3. Getrennte Verwaltung von Zähler und Nenner.
4. Einsparungen durch gleichzeitige Berechnung von $\frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)}$.

Gleichzeitige Berechnung von $f_P(Q+S)$ und $f_P(S)$

Diese Optimierung wurde schon in Kapitel 4.5.4 angesprochen, nämlich die Auswertung der Funktion bei den beiden Punkte $Q+S$ und S in einem einzigen Aufruf des Miller-Algorithmus durchzuführen.

Die Optimierung ist in Algorithmus 5 eingeflossen. Die Berechnungen von $f_P(Q+S)$ und $f_P(S)$ werden dabei komplett getrennt in den Variablen $\hat{f}$ beziehungsweise $\bar{f}$ durchgeführt. Ganz am Ende des Algorithmus erfolgt die Division $\hat{f}/\bar{f}$, um den Wert $f_P(\mathcal{A}_Q)$ zu erhalten. Als Gewinn gegenüber der zweimaligen Ausführung von Algorithmus 4 werden die Punktverdopplungen und Punktadditionen zur Berechnung von pP nur einmal ausgeführt, nicht zweimal.

Einsparung der finalen Division

Anstatt die Division $\hat{f}/\bar{f}$ erst am Ende von Algorithmus 5 durchzuführen, kann man diese auch schon direkt in die vorherigen Berechnungen miteinbeziehen und somit die

zusätzliche Division komplett einsparen. So ergibt sich aus den Zeilen 1–4 von Algorithmus 5:

$$f_1 = \frac{\hat{f}_1}{\bar{f}_1} = \frac{V_{P+R}(Q+S)/L_{P,R}(Q+S)}{V_{P+R}(S)/L_{P,R}(S)} = \frac{V_{P+R}(Q+S)\cdot L_{P,R}(S)}{L_{P,R}(Q+S)\cdot V_{P+R}(S)} \tag{5.1}$$

Die Zeilen 7–8 werden zu:

$$\begin{aligned} f_{2k} = \frac{\hat{f}_{2k}}{\bar{f}_{2k}} &= \frac{\hat{f}_k^2 \cdot T_Z(Q+S)/V_{2Z}(Q+S)}{\bar{f}_k^2 \cdot T_Z(S)/V_{2Z}(S)} = \left(\frac{\hat{f}_k}{\bar{f}_k}\right)^2 \cdot \frac{T_Z(Q+S)/V_{2Z}(Q+S)}{T_Z(S)/V_{2Z}(S)} \\ &= f_k^2 \cdot \frac{T_Z(Q+S)\cdot V_{2Z}(S)}{V_{2Z}(Q+S)\cdot T_Z(S)} \end{aligned} \tag{5.2}$$

Die Zeilen 11–12 werden zu:

$$\begin{aligned} f_{k+1} = \frac{\hat{f}_{k+1}}{\bar{f}_{k+1}} &= \frac{\hat{f}_k \cdot \hat{f}_1 \cdot L_{Z,P}(Q+S)/V_{Z+P}(Q+S)}{\bar{f}_k \cdot \bar{f}_1 \cdot L_{Z,P}(S)/V_{Z+P}(S)} \\ &= \frac{\hat{f}_k}{\bar{f}_k} \cdot \frac{\hat{f}_1}{\bar{f}_1} \cdot \frac{L_{Z,P}(Q+S)/V_{Z+P}(Q+S)}{L_{Z,P}(S)/V_{Z+P}(S)} \\ &= f \cdot f_1 \cdot \frac{L_{Z,P}(Q+S)\cdot V_{Z+P}(S)}{V_{Z+P}(Q+S)\cdot L_{Z,P}(S)} \end{aligned} \tag{5.3}$$

Diese Umformungen sind die Basis für Algorithmus 6.

Getrennte Verwaltung von Zähler und Nenner

In Algorithmus 6 ist zusätzlich noch eine Optimierung aus [ASN07, Kap. 2.2] umgesetzt worden. Anstatt die Divisionen T_Z/V_{2Z} und $L_{Z,P}/V_{Z+P}$ direkt innerhalb der Schleife auszuwerten, werden stattdessen der Zähler in der Variablen g und der Nenner in der Variablen h getrennt verwaltet. Anstatt mehrerer Divisionen innerhalb der Schleife existiert nur noch eine einzige Division am Ende des Algorithmus, bei der der Zähler durch den Nenner dividiert wird.

Unter Berücksichtigung von $f = \frac{g}{h}$ wird der Ausdruck (5.1) in Zeile 1 und 2 von Algorithmus 6 berechnet. Analog dazu wird (5.2) in den Zeilen 7–8 und (5.3) in den Zeilen 11–12 von Algorithmus 6 berechnet.

Einsparungen durch gleichzeitige Berechnung von $f_P(\mathcal{A}_Q)$ und $f_Q(\mathcal{A}_P)$

Eine kleine Verbesserung lässt sich noch erreichen, wenn man $\frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)}$ innerhalb einer einzigen Funktion berechnet [ASN07, Kap. 2.2]. Auch wenn man dort die meisten Berechnungen nach wie vor separat durchführen muss, sind die Berechnungen $g = g^2, h = h^2, g = g \cdot g_1, h = h \cdot h_1$ innerhalb der Schleife nur einmal nötig, nicht zweimal, man spart also einige wenige Operationen ein. Die Anpassungen sind in Algorithmus 7 dargestellt. Die Punktmultiplikation pP wird nach wie vor in der Variablen Z berechnet, während die Punktmultiplikation pQ in der Variablen W vorgehalten wird. Zur Berechnung von $e_p(P,Q)$ reicht jetzt ein einmaliger Aufruf von Algorithmus 7, da in ihm die Miller-Lite-Variante und die Miller-Full-Variante vereint sind.

Algorithmus 7: Finale Variante des Miller-Algorithmus zur Berechnung des Weil-Pairing

Input : $P, R \in E(\mathbb{F}_q)[p],\ Q, S \in E(\mathbb{F}_{q^2})[p],\ p = p_t, p_{t-1}, \ldots, p_0$

Output : $f = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)} = \frac{f_P(Q+S) \cdot f_Q(R)}{f_P(S) \cdot f_Q(P+R)} \in \mathbb{F}_{q^2}^*$

$g_1 = V_{P+R}(Q+S) \cdot L_{P,R}(S)$
$h_1 = L_{P,R}(Q+S) \cdot V_{P+R}(S)$
$g_1 = g_1 \cdot L_{Q,S}(P+R) \cdot V_{Q+S}(R)$
$h_1 = h_1 \cdot V_{Q+S}(P+R) \cdot L_{Q,S}(R)$
$g = g_1$
$h = h_1$
$Z = P$
$W = Q$
for $i = t-1$ **downto** 0 **do**
  $g = g^2 \cdot\ T_Z(Q+S) \cdot V_{2Z}(S)\ \cdot\ V_{2W}(P+R) \cdot T_W(R)$
  $h = h^2 \cdot\ V_{2Z}(Q+S) \cdot T_Z(S)\ \cdot\ T_W(P+R) \cdot V_{2W}(R)$
  $Z = 2Z$
  $W = 2W$
  if $m_i = 1$ **then**
    $g = g \cdot g_1 \cdot\ L_{Z,P}(Q+S) \cdot V_{Z+P}(S)\ \cdot\ V_{W+Q}(P+R) \cdot L_{W,Q}(R)$
    $h = h \cdot h_1 \cdot\ V_{Z+P}(Q+S) \cdot L_{Z,P}(S)\ \cdot\ L_{W,Q}(P+R) \cdot V_{W+Q}(R)$
    $Z = Z + P$
    $W = W + Q$
return $f = g/h$

Anmerkung

Im Rahmen der Umsetzung der obigen Optimierungen wurden zusätzlich auch die Berechnungen der Punktverdopplung und Punktaddition dahingehend angepasst, dass sie neben dem eigentlichen Ergebnis auch zusätzlich noch die Steigung λ zurückliefern (siehe Gleichungen (4.2) und (4.1) auf Seite 38), denn diese wird gerade für die Auswertung der Funktionen $L_{Z,P}$ und T_Z benötigt. In einer ersten Version war die Steigung noch einmal explizit neu berechnet worden.

5.6.2. Laufzeitanalyse

An dieser Stelle soll noch einmal eine theoretische Abschätzung des Aufwandes an Multiplikationen innerhalb der Schleife von Algorithmus 7 erfolgen. Zur Erinnerung, I wird mit $15M$ angesetzt, $S = M, M' = 3M$. Für den Punktverdopplungsblock, also die Zeilen

	Min	Max	∅
$t(e_p(P,Q)$ basierend auf Algorithmus 6)	7262 ms	8578 ms	7850 ms
$t(e_p(P,Q)$ mit Algorithmus 7)	6497 ms	7940 ms	7340 ms

Tabelle 5.3.: Aufwand zur Berechnung von $e_p(P,Q)$

10–13, ergibt sich folgender Aufwand:

$$\begin{aligned}
t(\text{Berechnung } g) &= S' + 2M' + M' + M' + 2M' = 20M \\
t(\text{Berechnung } h) &= S' + M' + 2M' + 2M' + M' = 20M \\
t(\text{Berechnung } 2Z) &= I + 2M + 2S = I + 4M = 19M \\
t(\text{Berechnung } 2W) &= I' + 2M' + 2S' = I + 12M + 2M = I + 14M = 29M \\
t(\text{Punktverdopplungsblock}) &= 20M + 20M + (I+4M) + (I+14M) = 2I + 58M = 88M
\end{aligned}$$

Für den Punktadditionsblock, also die Zeilen 15–18, ergibt sich folgender Aufwand:

$$\begin{aligned}
t(\text{Berechnung } g) &= M' + 2M' + M' + M' + 2M' = 21M \\
t(\text{Berechnung } h) &= M' + M' + 2M' + 2M' + M' = 21M \\
t(\text{Berechnung } Z + P) &= I + 2M + S = I + 3M = 18M \\
t(\text{Berechnung } W + Q) &= I' + 2M' + S' = I + 10M + 2M = I + 12M = 27M \\
t(\text{Punktadditionsblock}) &= 20M + 20M + (I+3M) + (I+12M) = 2I + 57M = 87M
\end{aligned}$$

Der Punktverdopplungsblock wird $\log(p)$-mal ausgeführt, der Punktadditionsblock im Mittel etwa $\frac{\log(p)}{2}$-mal, wobei p eine Bitlänge von 160 Bit hat. Damit ergibt sich als Abschätzung für die gesamte Schleife:

$$\begin{aligned}
t(\text{Miller-Schleife}) &= 160 \cdot t(\text{Punktverdopplungsblock}) + 80 \cdot t(\text{Punktadditionsblock}) \\
&= 160 \cdot 88M + 80 \cdot 87M = 14080M + 6960M \\
&= 21040M
\end{aligned}$$

Tabelle 5.3 zeigt den Aufwand für die Berechnung von $e_p(P,Q)$ auf dem Samsung ARM Halbleiter, einmal basierend auf Algorithmus 6 und einmal mit der zusätzlichen Optimierung von Algorithmus 7. Wie erwartet ist Algorithmus 7 im Mittel etwas schneller. Entsprechend der Werte aus den Tabellen 5.1 und 5.2 werden von den gut 7 Sekunden $1{,}43 + 2{,}32 = 3{,}75$ Sekunden für die implizite Berechnung von pP und pQ benötigt, also mehr als die Hälfte der Zeit. Der andere Teil wird demzufolge für die eigentliche Auswertung der Funktionen f_P und f_Q (also die Berechnung der jeweiligen Variablen g und h) benötigt.

5.6.3. Weitere Optimierungsmöglichkeiten

In diesem Abschnitt werden weitere Optimierungsmöglichkeiten aufgezeigt, die bisher nicht umgesetzt wurden. Für die Optimierung kann man an verschiedenen Punkten ansetzen:

1. Nutzung einer anderen Paarung anstelle des Weil-Pairings
2. Weitere Optimierungen am Miller-Algorithmus
3. Spezielle Wahl der Parameter

Punkt 1 wurde im Implementierungsteil der Diplomarbeit nicht näher betrachtet, hier lag der Fokus erst einmal auf dem Weil-Pairing, um die Kompatibilität zu einer Anwendung aus dem TURBINE-Projekt gewährleisten zu können, die ebenfalls auf dem Weil-Pairing basiert. Prinzipiell könnte man anstelle des Weil-Pairings aber eine andere Paarung wie zum Beispiel das Tate-Pairing benutzen, dieser Punkt wird in Abschnitt 5.8 beim Vergleich zu anderen Arbeiten etwas näher beleuchtet.

Algorithmische Optimierungen

Bezüglich Punkt 2 sind auf jeden Fall noch weitere Optimierungen am Miller-Algorithmus möglich. Im Algorithmus findet implizit eine Punktmultiplikation statt, daher liegt es nahe, die bei der Punktmultiplikation gefundenen Optimierungen auch hier anzuwenden. Anstatt die Schleife im Miller-Algorithmus über die Binärdarstellung von p laufen zu lassen, könnte man auch hier die NAF-Darstellung von p benutzen (vgl. Seite 42). Durch die NAF-Darstellung würde der Block mit der Punktaddition entsprechend den Zeilen 11–13 von Algorithmus 6 beziehungsweise Zeilen 15–18 von Algorithmus 7 weniger oft durchlaufen. Dadurch spart man nicht nur Punktadditionen ein, sondern auch Aktualisierungsberechnungen von g und h. Zur Erinnerung noch einmal die Abschätzung für die Laufzeit der Miller-Schleife von Algorithmus 7:

$$t(\text{Miller-Schleife}) = 160 \cdot 88M + 80 \cdot 87M = 14080M + 6960M = 21040M$$

Für die NAF-Darstellung würde der Additionsblock nur noch etwa $\frac{\log(p)}{3}$-mal durchlaufen werden. Damit ergibt sich:

$$t(\text{Miller-Schleife-NAF}) = 160 \cdot 88M + 53 \cdot 87M = 14080M + 4611M = 18691M$$

Dies entspricht einem Gewinn von etwa 11%, spart also ungefähr 807 Millisekunden bei Algorithmus 7 ein. Daher wurde zum Beispiel auch in [CSB05] die NAF-Darstellung genutzt.

Eine weitere Optimierung besteht darin, die Berechnungen mit den Punkten P, Q, R, S nicht im affinen Koordinatensystem durchzuführen, sondern im Jacobischen Koordinatensystem. Als erstes erhält man dadurch den Geschwindigkeitsgewinn entsprechend der Tabellen 5.1 und 5.2, da im Miller-Algorithmus ja implizit pP beziehungsweise pQ berechnet werden. Der Unterschied zwischen affinem und Jacobischem Koordinatensystem beim Double-And-Add für die beiden Berechnungen liegt im Durchschnitt bei $(1429 - 675) + (2324 - 1974) = 1104$ Millisekunden. Durch die Umstellung sollte man im ersten Schritt also erst einmal eine gute Sekunde gewinnen. Auf der anderen Seite müssen dann aber auch die Funktionen L, T, V im Jacobischen System berechnet werden. Bei der Auswertung von L und T im affinen System wurde jeweils nur eine Multiplikation in $\mathbb{F}_{q^2}$ benötigt, im Jacobischen System gilt dies nicht.

- [IT03, Kap. 3.2] zeigt die Formeln von L, T, V im Jacobischen System nach einem einfachen Ansatz bei der Berechnung des Tate-Pairings, daher bezieht es sich

nur auf die Miller-Lite-Variante. Sei $U = (X_1, Y_1, Z_1), V = (X_2, Y_2, 1)$, $V + U = (X_3, Y_3, Z_3), 2U = (X_4, Y_4, Z_4)$. Dann ist

$$\begin{aligned} L_{U,V}(x,y) &= Z_3(y - Y_2) - (Y_2 Z_1^3 - Y_1)(x - X_2) \\ V_{U+V}(x,y) &= Z_3^2 x - X_3 \\ T_U(x,y) &= (Z_4 Z_1^2 y - 2Y_1^2) - (3X_1^2 + aZ_1^4)(Z_1^2 x - X_1) \end{aligned}$$

Für die Berechnung der zusätzlichen Koeffizienten von L im Punktadditionsblock benötigen die Autoren $2M + S$, für die Auswertung von L und V zusammen $2 \cdot (3k \cdot M) = 12M$, insgesamt ergibt dies $15M$. Auf der anderen Seite reduziert sich t(Berechnung $Z + P$) durch das Jacobische System von $18M$ auf $11M$, zusätzlich wird ein M' eingespart, was bisher für die Auswertung von L im affinen System benötigt wurde. Insgesamt werden $10M$ eingespart, aber $15M$ zusätzlich benötigt, die Berechnung wäre insgesamt langsamer.

Ein ähnliches Bild entsteht im Punktverdopplungsblock, für die Berechnung der zusätzlichen Koeffizienten werden $3M + 1S$ benötigt, für die Auswertung von T und V in Summe $2 \cdot (3k \cdot M) = 12M$, macht insgesamt $16M$. Eingespart werden $12M$ durch die schnellere Punktverdopplung und M' bei der Auswertung von T, in Summe also $15M$, auch hier wäre die die Berechnung insgesamt langsamer. Ohne zusätzliche Optimierungen lohnt sich der Einsatz des Jacobischen System auf dem Samsung ARM für den Miller-Algorithmus also nicht. Das Ergebnis würde anders aussehen, wenn I deutlich größer wäre als $15M$, da dann die Einsparungen höher wären.

- In [CSB05, Kap. 3] werden ebenfalls Formeln für die Auswertung von L und T im Jacobischen System angegeben. Unter besonderer Ausnutzung der dort verwendeten Distortion-Map benötigen die Autoren für die Auswertung von T nur vier zusätzliche Multiplikationen in $\mathbb{F}_q$, wobei sie hier auch Zwischenergebnisse wiederverwenden, die bei der Punktverdopplung aufgetreten sind. Für die Auswertung von L benötigen sie drei Multiplikationen in $\mathbb{F}_q$. Die Ergebnisse lassen sich nicht direkt auf die in dieser Diplomarbeit genutzte Konstellation abbilden, da sich die Distortion-Maps deutlich unterscheiden.

Es lässt sich festhalten, dass der Gewinn bei der Punktmultiplikation durch die zusätzlichen Multiplikationen bei der Auswertung der Geradengleichungen wieder reduziert wird. Nach dem einfachen Ansatz sogar so sehr, dass es sich auf dem Samsung ARM nicht lohnt. Ob es auch für die hier genutzte Distortion-Map eine Optimierung gibt, die die Nutzung des Jacobischen System im Miller-Algorithmus im Endeffekt auf dem Samsung ARM doch lohnend macht, müsste erst eine genauere Untersuchung zeigen.

Wahl der Parameter

Bezüglich Punkt 3 kann man als weitere Möglichkeit versuchen, durch geschickte Wahl der Parameter einen Geschwindigkeitsgewinn zu erzielen. Dies bezieht sich in erster Linie auf die Wahl der Kurve E, der Primzahlen p, q und des Einbettungsgrades k.

In der Literatur findet man häufig den Hinweis, die Primzahl p (oder auch p und q) so zu wählen, dass sie ein sehr geringes Hamming-Gewicht hat, es sind also nur sehr wenige Bits gesetzt. Hier bieten sich die sogenannten Solinas-Primzahlen an, diese haben die Form $2^a \pm 2^b \pm 1, 0 < b < a$ (siehe zum Beispiel [KM05, 6.2]). Bei Solinas-Primzahlen sind insgesamt also nur 3 Bits gesetzt, damit wird der Block mit der Punktaddition insgesamt sogar nur zweimal durchlaufen, während er bei einem zufälligen p etwa $\frac{\log(p)}{2}$-mal durchlaufen wird. Der Geschwindigkeitsgewinn durch diese Optimierung ist daher beachtlich. Betrachtet man noch einmal die Abschätzung von $t(\text{Miller-Schleife}) = 21040M$, so würde sich diese Abschätzung bei einer Solinas-Primzahl reduzieren auf:

$$t(\text{Miller-Schleife-Solinas-Primzahl}) = 160 \cdot 88M + 2 \cdot 87M = 14080M + 174M = 14254M$$

Dies entspricht einem Gewinn von knapp einem Drittel, spart bei Algorithmus 7 also etwa 2,4 Sekunden ein.

Allerdings ist nicht klar, inwieweit sich die besondere Form von p auf die Sicherheit auswirkt. Auf jeden Fall kann man für solche Primzahlen speziell angepasste Algorithmen zur Berechnung des diskreten Logarithmus verwenden [KM05, 6.2]. Aufgrund dieser Sicherheitsbedenken wurde p im Rahmen der Implementierung für diese Diplomarbeit nicht als Solinas-Primzahl gewählt, sondern zufällig entsprechend Kapitel 4.3, auch wenn man dadurch auf ein großes Optimierungspotential verzichtet.

In [PKY05] stellen die Autoren eine Variante des Miller-Algorithmus vor, bei der der Wert $e_p(P, \phi(Q'))$ mittels zweier Miller-Lite-Aufrufe berechnet werden kann, falls die Distortion-Map ϕ bestimmte Anforderungen erfüllt. Mit solchen speziellen Distortion-Maps lässt sich f_Q berechnen als $f_Q = f_{\phi(Q')} = \lambda \cdot f_{Q'} \circ \phi^{-1}$. Anstatt also im Miller-Full-Aufruf die Punktmultiplikation pQ mit $Q \in E(\mathbb{F}_{q^2})[p]$ in der großen Gruppe $E(\mathbb{F}_{q^2})$ zu berechnen, läuft hier die Punktmultiplikation pQ' mit $Q \in E(\mathbb{F}_q)[p]$ in der kleinen Gruppe $E(\mathbb{F}_q)$ ab, es ist also quasi ein zweiter Miller-Lite-Aufruf. Als geeignete Distortion-Map für die Kurve $Y^2 = X^3 + 1$ wird $\phi(x, y) = (\alpha x, y)$ mit $\alpha^2 + \alpha + 1 = 0$ angegeben.

Bezogen auf die Abschätzung in Kapitel 5.6.2 für den Aufwand der Miller-Schleife würde im affinen System der Aufwand $t(\text{Berechnung } 2W)$ von $29M$ auf $19M$ verringert und $t(\text{Berechnung } W + Q)$ von $27M$ auf $18M$. Damit wäre

$$t(\text{Miller-Schleife-2×Miller-Lite}) = 160 \cdot 78M + 80 \cdot 78M = 18720M.$$

Hierdurch wird ein Gewinn von etwa 11% erzielt, das entspräche etwa 809 Millisekunden. Betrachtet man den Unterschied einer Punktmultiplikation in $E(\mathbb{F}_q)$ und $E(\mathbb{F}_{q^2})$ entsprechend Tabelle 4.2 und Tabelle 4.1, so ergibt sich ein leicht höherer Gewinn von $2324 - 1429 = 895$ Millisekunden. Für das Jacobische System ergäbe sich durch diese Optimierung ein Einsparpotential entsprechend der Messwerte für die Punktmultiplikation von etwa 1,3 Sekunden ($1974 - 675 = 1299$ Millisekunden).

Zwischenfazit

Die Laufzeit von gut 7 Sekunden für die Berechnung eines Weil-Pairings mit Algorithmus 7 entspricht den Erwartungen, die man aus der Dauer der Punktmultiplikationen ableiten konnte. Würde man als Optimierung den zweifachen Miller-Lite-Aufruf zusammen mit der NAF-Darstellung umsetzen, so ergibt sich in Summe:

$$t(\text{Miller-Schleife-NAF-2×Miller-Lite}) = 160 \cdot 78M + 53 \cdot 78M = 16614M$$

Dies entspricht einem Gewinn von etwa 21% oder 1544 Millisekunden. Würde man trotz der Sicherheitsbedenken zusätzlich eine Solinas-Primzahl verwenden, so wäre zwar die Optimierung mit der NAF-Darstellung nicht mehr sinnvoll, allerdings würde sich auch ohne die NAF folgender Wert ergeben:

$$t(\text{Miller-Schleife-Solinas-2}\times\text{Miller-Lite}) = 160 \cdot 78M + 2 \cdot 78M = 12636M$$

Dies entspricht einem Gewinn von knapp 40% oder 2932 Millisekunden. Damit wäre eine Laufzeit von 4,4 Sekunden für die Berechnung des Weil-Pairings möglich.

Allerdings liegt man selbst mit einer Zeit von 4,4 Sekunden noch in einem Bereich, der vermutlich nicht als praxistauglich angesehen wird. Für das alleinige Berechnen einer Paarung ist es eventuell gerade noch akzeptabel, aber spätestens im Rahmen der Gruppensignatur mit den zusätzlichen Berechnungen wird die Zeit zu hoch sein.

5.7. Berechnung der Gruppensignatur

Mit all den getroffenen Vorarbeiten ist die konkrete Umsetzung der Gruppensignatur aus Kapitel 3.4.2 möglich. Für die Umsetzung muss man beachten, dass in den Formeln eine multiplikative Schreibweise für $\mathbb{G}_1$ benutzt wird, $E(\mathbb{F}_q)$ ist allerdings eine additive Gruppe. In den Formeln muss man daher für alle Elemente aus $\mathbb{G}_1$ eine Multiplikation durch eine Punktaddition und eine Exponentiation durch eine Punktmultiplikation ersetzen. Die meisten Parameter sind Elemente aus $\mathbb{G}_1$ oder $\mathbb{Z}_p$. Damit kann man die Formeln mit den entsprechenden Funktionen aus den vorherigen Abschnitten umsetzen, dies sind entsprechend Punktadditionen oder Punktmultiplikationen in $E(\mathbb{F}_q)$ (Abschnitt 5.5) oder Operationen in $\mathbb{Z}_p$ (Abschnitt 5.4). Für die Punktmultiplikation wird der Algorithmus genutzt, der intern im Jacobischen Koordinatensystem und mit der NAF-Darstellung des Skalars arbeitet.

Etwas aufwendiger ist hingegen die Berechnung des Wertes $R_2 \in \mathbb{F}^*_{q^2}$, denn genau hier wird das Weil-Pairing benötigt. Dabei ist folgendes zu beachten: Nach Definition benötigt man drei Auswertungen von $\hat{e}$, um den Wert R_2 zu berechnen. Aufgrund der Bilinearität von $\hat{e}$ lässt sich dies aber wie folgt umformen:

$$\begin{aligned} R_2 &= \hat{e}(T_2, g_2)^{r_x} \cdot \hat{e}(v, w)^{-r_\alpha} \cdot \hat{e}(v, g_2)^{-r_\delta} \\ &= \hat{e}(r_x \cdot T_2, g_2) \cdot \hat{e}(v, w)^{-r_\alpha} \cdot \hat{e}(-r_\delta \cdot v, g_2) \\ &= \hat{e}(r_x \cdot T_2 + (-r_\delta \cdot v), g_2) \cdot \hat{e}(v, w)^{-r_\alpha} \end{aligned}$$

Somit lässt sich R_2 also mit nur zwei Auswertungen von $\hat{e}$ berechnen. Auch wenn man dafür zwei zusätzliche Punktmultiplikationen in Kauf nehmen muss, ist dies immer noch schneller als eine zusätzliche Auswertung von $\hat{e}$. Bezeichne GroupSignatureSign1() die Variante, bei der R_2 mittels 3 Auswertungen von $\hat{e}$ berechnet wird und GroupSignatureSign2() die Variante mit 2 Auswertungen von $\hat{e}$. Für die Berechnung von $\hat{e}$ wird Algorithmus 7 benutzt.

Die Auswertung der Hashfunktion H_0 ist relativ aufwendig, denn entsprechend Kapitel 4.6 und 4.4 werden hier zwei zufällige Punkte aus $E(\mathbb{F}_q)$ jeweils mit dem Co-Faktor $h = \frac{q+1}{p}$ multipliziert. Aufgrund der Wahl von p und q hat h eine Länge von 352 Bit, die Multiplikation mit h ist also eine recht teure Operation. Zum Vergleich: für Punktmultiplikationen von Elementen aus $\mathbb{G}_1$ haben die Skalare maximal eine Länge von 160 Bit.

	Min	Max	∅
$t($ GroupSignatureSign1() $)$	25826 ms	29592 ms	27710 ms
$t($ GroupSignatureSign2() $)$	19985 ms	22387 ms	21337 ms

Tabelle 5.4.: Aufwand zur Berechnung der Gruppensignatur

Für die Zeitmessungen in Tabelle 5.4 wurden 1000 Durchläufe des jeweiligen Algorithmus ausgewertet. Die Primzahlen p, q und die Schlüssel des Gruppensignaturschemas wurden zufällig entsprechend ihrer Vorgaben gewählt. Die Nachricht m hatte eine Länge von 32 Byte und war ebenfalls zufällig gewählt. Wie zu erwarten war, ist die Variante 2 um die Dauer einer Berechnung von $\hat{e}$ abzüglich der zusätzlichen Punktmultiplikationen schneller als Variante 1.

Entsprechend den Werten in Tabelle 5.3 und 5.1 benötigt die Berechnung von zwei Paarungen 14,68 Sekunden. Die Berechnung der 7 Punktmultiplikationen benötigt $7 \cdot 0{,}594 = 4{,}158$ Sekunden. Damit verbleiben noch etwa 2,5 Sekunden für die Berechnung von H_0, einer Exponentiation in $\mathbb{F}_{q^2}$, Berechnung von H und einiger Operationen in $\mathbb{Z}_p$, wobei die Berechnung von H_0 den größten Anteil benötigt.

Die Dauer von 21 Sekunden zur Berechnung der Gruppensignatur ist zwar noch nicht praxistauglich, dafür würde man schon einen niedrigen einstelligen Sekundenwert benötigen. Für die erstmalige Umsetzung dieser komplexen Thematik auf dem Samsung ARM Halbleiter ist es aber ein akzeptabler Wert und liegt im Rahmen dessen, was man vor dem Start der Implementierungen als realistisch angesehen hatte.

Produkt von Paarungen

In [Sco05a, Kap. 4.3] und [GS06] werden Optimierungen für den Fall vorgestellt, dass nur das Produkt von mehreren Paarungen relevant ist, nicht jedoch die einzelnen Werte der Paarungen, so wie es auch hier bei der Gruppensignatur der Fall ist[2].

Bei einem Produkt $\hat{e}(P,Q) \cdot \hat{e}(R,S)$ kann man die Multiplikation direkt wieder auf den Miller-Algorithmus herunterbrechen. Analog zum Schritt von Algorithmus 6 zu Algorithmus 7 von Seite 69 kann man auch hier die Berechnungen gemeinsam innerhalb einer Funktion ausführen. Dadurch genügt es wiederum die Operationen $g = g^2, h = h^2, g = g \cdot g_1, h = h \cdot h_1$ insgesamt nur noch einmal innerhalb der gemeinsamen Schleife auszuführen und nicht je einmal innerhalb der jeweiligen Schleife bei der getrennten Berechnung von $\hat{e}(P,Q)$ und $\hat{e}(R,S)$.

Eine zusätzliche Optimierung ist möglich, falls für die Punktadditionen und Punktverdopplungen das affine Koordinatensystem genutzt wird. Bei der Berechnung der Steigung ist jeweils die Berechnung eines inversen Elementes nötig. Möchte man allerdings zwei inverse Elemente $\frac{1}{X}, \frac{1}{Y}$ berechnen, so lässt sich dies auch mittels $\frac{1}{X} = \frac{Y}{XY}, \frac{1}{Y} = \frac{X}{XY}$ erreichen. Hier wird nur eine Inversion für das Element XY berechnet, trotzdem erhält man durch die zusätzlichen Multiplikationen beide inversen Elemente $\frac{1}{X}, \frac{1}{Y}$. In der gemeinsamen Schleife werden gleichzeitig Punktadditionen und Punktverdopplungen für

[2] Der Wert R_2 wird mittels $\hat{e}(P,Q) \cdot \hat{e}(R,S)^x$ berechnet, dies lässt sich zu $\left(\hat{e}(\frac{1}{x}P,Q) \cdot \hat{e}(R,S)\right)^x$ umformen, wobei P schon mittels Punktmultiplikationen berechnet wird, sodass die Multiplikation mit $\frac{1}{x}$ jetzt keine zusätzliche Punktmultiplikation ist, sondern nur eine Veränderung der bisher benötigten um den Faktor $\frac{1}{x}$.

P und R sowie Q und S durchgeführt. Führt man die Operationen von P und R beziehungsweise Q und S gemeinsam in einer Funktion durch, kann durch die Optimierung jeweils eine Inversion bei der Berechnung der Steigung eingespart werden.

Das Vorgehen funktioniert allgemein auch für ein Produkt von mehreren Paarungen, nicht nur für zwei. Laut der Autoren ist für das Weil-Pairing jede zusätzliche Multiplikation mit einer Paarung bis zu 30% günstiger als wenn die zusätzliche Paarung getrennt berechnet würde. Wegen Zeitmangels konnte diese Optimierung leider nicht mehr umgesetzt werden. Mit ihr wäre eine Zeit von weniger als 20 Sekunden für die Berechnung der Gruppensignatur möglich.

5.8. Vergleich zu anderen Arbeiten

In [BBC+08] und [SCA06] werden ebenfalls Paarungen auf einer Smart Card berechnet. Dort wird allerdings nicht das Weil-Pairing benutzt, sondern das Tate-Pairing.

Sei $P \in E(\mathbb{F}_q)[p]$, f_P eine Funktion mit Divisor $div(f_P) = p[P] - p[\mathcal{O}]$. Sei $Q \in E(\mathbb{F}_{q^k})[p]$, $\mathcal{A}_Q$ ein Divisor äquivalent zu $[Q] - [\mathcal{O}]$. Dann ist das Tate-Pairing eine Abbildung

$$e_t : E(\mathbb{F}_q)[p] \times E(\mathbb{F}_{q^k})[p] \to \mu_p,\ e_t(P,Q) \mapsto f_P(\mathcal{A}_Q)^{(q^k-1)/p}.$$

μ_p bezeichnet die p-ten Einheitswurzeln in $\mathbb{F}^*_{q^k}$. Das Tate-Pairing ist sehr ähnlich zum Weil-Pairing und lässt sich im Prinzip mit einer Runde der Miller-Lite-Variante, gefolgt von einer finalen Exponentiation mit $\frac{q^k-1}{p}$ berechnen. Allgemein gilt es als mindestens um den Faktor 2 schneller als das Weil-Pairing, da die Miller-Full-Variante erst gar nicht benötigt wird (siehe [Jou04][S. 266]).

Untersuchungen auf einem ST22-Halbleiter

In [BBC+08] erreichen die Autoren auf einem ST22-Halbleiter mit einer Taktgeschwindigkeit von 33 MHz eine Ausführungszeit von 752 ms zur Berechnung eines Tate-Pairings. Gegen diesen Wert wirken die 7 Sekunden, die in dieser Arbeit für die Berechnung eines Weil-Pairings benötigt werden, auf den ersten Blick recht langsam. Daher gilt es zu untersuchen, ob man den Geschwindigkeitsunterschied durch Optimierungen oder spezielle Gegebenheiten erklären kann.

Die Autoren arbeiten ebenfalls auf einer supersingulären Kurve $E : Y^2 = X^3 + X$ über $\mathbb{F}_q$ mit $k = 2$, für p und q gelten Bitlängen von 160 beziehungsweise 512 Bit. Auch auf dieser Kurve existiert eine Distortion-Map, die sich die Autoren zunutze machen. Als Erweiterungskörper dient $\mathbb{F}_{q^2}$. Bis hierher ist die Ausgangslage also relativ gut vergleichbar.

Allerdings existieren für die Berechnung des Tate-Pairings weitere Optimierungen, die für das Weil-Pairing in der Art nicht möglich sind. Da $q-1$ ein Faktor von $\frac{q^2-1}{p}$ ist, werden alle Werte aus $\mathbb{F}^*_q$ durch die finale Exponentiation zu 1. Daher lässt sich bei der gegebenen Kurve das Tate-Pairing vereinfacht berechnen, dort ist nämlich $e_t(P,Q) = f_P(Q)^{(q^2-1)/p}$ (anstatt $\left(\frac{f_P(Q+S)}{f_P(S)}\right)^{(q^2-1)/p}$) [BKLS02, Theorem 1]. Zusätzlich ist durch die spezielle Wahl der Distortion-Map die Auswertung der Funktion V für die Vertikale durch einen Punkt ein Element aus $\mathbb{F}_q$, somit wird diese durch die finale Exponentiation ebenfalls zu 1 und muss daher im Miller-Algorithmus gar nicht erst berechnet werden. Dies ist unter

dem Ausdruck *Denominator Elimination* bekannt. Der optimierte Miller-Algorithmus für das Tate-Pairing – genannt BKLS-Algorithmus – benötigt also deutlich weniger Operationen als der Miller-Algorithmus für das Weil-Pairing. Als weitere Optimierung werden für p und $\frac{q+1}{p}$ Zahlen mit geringem Hamming-Gewicht gewählt, um zum einen kaum im Additionszweig des BKLS-Algorithmus zu landen und zum anderen die finale Exponentiation effizienter berechnen zu können.

Um bei der Punktmultiplikation die kostspieligen Inversionen im Körper $\mathbb{F}_q$ zu vermeiden, benutzen die Autoren zur Darstellung der Punkte modifizierte Chudnowsky-Koordinaten, die gegenüber den Jacobischen Koordinaten scheinbar noch zusätzliche Geschwindigkeitsvorteile bieten.

Weiterhin wurde viel Energie in die Entwicklung einer effizienten modularen Multiplikationsfunktion gesteckt, die spezielle Eigenschaften des Halbleiters ausnutzt, die von den Autoren allerdings nicht näher beschrieben werden. Laut der Autoren ist die Funktion für die gewählten Bitlängen wesentlich besser geeignet als etwa eine Montgomery-Multiplikation. Die 65% höhere Taktfrequenz im Vergleich zum Samsung ARM ist natürlich ebenfalls ein wesentlicher Faktor für die schnellere Berechnung.

Untersuchungen auf einem Philips HiPerSmart™

In [SCA06] stellen die Autoren ihre Implementierung des Tate-Pairings auf einem Philips HiPerSmart™-Halbleiter vor. Bei einer Taktfrequenz von 20 Mhz benötigen sie für die Berechnung eines Tate-Pairings 470 ms. Für die Berechnung wird ebenfalls ein optimierter BKLS-Algorithmus genutzt. Die zuvor angesprochenen Optimierungen wurden auch hier umgesetzt. Als Kurve wird eine gewöhnliche und keine supersinguläre Kurve benutzt, allerdings ebenfalls mit Einbettungsgrad 2, für p wird eine Solinas-Primzahl gewählt. Die Arithmetik in $\mathbb{F}_{q^2}$ wird auf die gleiche Art realisiert, wie in Kapitel 4.1.2 vorgestellt.

Allerdings benutzen die Autoren einen Trick, der in [BBC+08] nicht zum Einsatz kam: Alle Punktverdopplungen und Punktadditionen im BKLS-Algorithmus werden komplett außerhalb der Smart Card vorberechnet, die dabei auftretenden x und y-Koordinaten werden zusammen mit der Steigung λ in Form einer Tabelle eingebracht. Dieses ist allerdings nur möglich, wenn im Vorhinein schon bekannt ist, für welchen Punkt P das Tate-Pairing berechnet werden soll. Die Autoren benutzen als Anwendungsfall für das Tate-Pairing eine Entschlüsselung im Identity-Based-Encryption-Schema entsprechend [BF03]. In diesem Fall entspricht der Punkt P gerade dem privaten Schlüssel und ist daher im Vorhinein bekannt, nur aus diesem Grund ist die Optimierung mit der Vorberechnung überhaupt möglich. Für die Gruppensignatur aus Kapitel 3.4.2 trifft dies nicht zu, denn dort hängen die Punkte bei einer Signaturberechnung zum einen von der Nachricht, zum anderen von weiteren Zufallswerten ab, eine Vorberechnung ist daher nicht möglich.

Im Miller-Algorithmus für das Weil-Pairing in Kapitel 5.6 wurde etwa die Hälfte der Laufzeit für die Punktadditionen und Punktverdopplungen benötigt. Für den BKLS-Algorithmus dürfte der Anteil eher noch höher sein, dieser wird jetzt durch die Vorberechnung komplett eingespart. Die Optimierung wird allerdings teuer erkauft: Nach [Sco05a][S. 299] benötigt man zur Speicherung der vorberechneten Werte etwa 30 KByte an Code. In der Chipkartenwelt sind das Dimensionen, die durchaus die nächstgrößere Variante eines Halbleiters notwendig machen können, was sich sofort negativ im Preis

auswirkt.

Bezüglich des Halbleiters heben die Autoren hervor, dass der Chip einen hochoptimierten Hauptprozessor besitzt und daher keinen Crypto-Coprozessor benötigt. Durch spezielle Befehlserweiterungen kann der Hauptprozessor insbesondere die modulare Multiplikation sehr schnell berechnen. Im Gegensatz zu einem Crypto-Coprozessor, der meist nur auf eine Aufgabe speziell zugeschnitten ist, erlaubt es der Philips HiPerSmart™ daher auch neuartige Algorithmen in ausreichender Geschwindigkeit auszuführen, die noch gar nicht bekannt waren, als der Chip entworfen wurde.

In [DSD07] wird ebenfalls auf dem Philips HiPerSmart™ ein Tate-Pairing über sogenannten Barreto-Naehrig Kurven berechnet. Durch den Einbettungsgrad von 12 müssen dort Operationen über $\mathbb{F}_{q^{12}}$ durchgeführt werden, nicht nur über $\mathbb{F}_{q^2}$. Mit diversen Tricks, um die Operationen in $\mathbb{F}_{q^{12}}$ auf Berechnungen in $\mathbb{F}_{q^2}$ zurückzuführen, benötigt die Berechnung eines Tate-Pairings für dieses komplizierte Setup knapp 10 Sekunden bei einer Taktfrequenz von 20 MHz. Dies soll an dieser Stelle nur verdeutlichen, wie stark die Ausführungszeit von der Wahl der Parameter und der Rahmenbedingungen abhängt.

Untersuchungen auf anderen Plattformen

[Gem05] enthält eine Ankündigung, dass die Chipkarte „Smart IBE“ eine vollständige Implementierung des IBE-Schemas von Boneh und Franklin enthalten würde. Außer der Ankündigung und zugehöriger Pressemitteilungen finden sich allerdings keine konkreten Informationen, wie auch in [BBC+08][Kap. 4.1] angemerkt wird, daher ist kein Vergleich möglich.

In [SOS+08] und [SKSC09] wird von den Autoren untersucht, inwiefern die Berechnung von Paarungen auf Halbleitern möglich ist, die in drahtlosen Sensornetzwerken zum Einsatz kommen. Diese unterliegen scheinbar ähnlichen Limitierungen wie Smart Cards aus dem Low-Cost-Bereich. Da [SKSC09] die aktuelleren (und besseren) Ergebnisse enthält, wird nur hierauf eingegangen.

In der Arbeit wird auf drei unterschiedlichen Plattformen ein Eta-Pairing und ein Tate-Pairing berechnet, wobei im Folgenden nur auf das Tate-Pairing eingegangen wird. Es wird eine sogenannte MNT-Kurve über einem Körper $\mathbb{F}_q$ benutzt, wobei q eine Bitlänge von 160 hat. p hat eine Bitlänge von 157, der Einbettungsgrad ist $k = 4$. Das Setup unterscheidet sich daher schon deutlich von dem in dieser Diplomarbeit gewählten zur Berechnung des Weil-Pairings, was die Vergleichbarkeit erschwert.

Für die Berechnung des Tate-Pairings wird wie zuvor auch eine Variante des BKLS-Algorithmus genutzt. Punktmultiplikationen werden durch Vorberechnungen optimiert. Für die modulare Arithmetik wird ausschließlich handoptimierter Assemblercode genutzt, dadurch wurden bis zu 66% an Laufzeit gegenüber einer C-Implementierung eingespart. Interessanterweise geben die Autoren die Anzahl an Multiplikationen an, die der BKLS-Algorithmus innerhalb der Miller-Schleife und für die finale Exponentiation ausführt, diese beträgt 6301. Zur Erinnerung, Algorithmus 7 benötigt in seiner Schleife zur Berechnung des Weil-Pairings etwa den Aufwand von 21040 Multiplikationen, wobei dieses Multiplikationen in einem Körper $\mathbb{F}_q$ sind, bei dem q eine Länge von 512 Bit hat.

Die untersuchten Plattformen sind ein MICA2 mit einem 8-Bit Prozessor bei einer Taktfrequenz von 7,38 MHz, ein „Tmote Sky“ mit einem 16-Bit Prozessor bei einer Taktfrequenz von 8,192 MHz und ein „Imote2“ mit einem 32-Bit Prozessor bei einer Taktfrequenz von 13 MHz. Für die Berechnung des Tate-Pairings benötigte der MICA2

7,43 Sekunden, der Tmote Sky 4,61 Sekunden und der Imote2 0,62 Sekunden. Der MICA2 hat nur 4 KB RAM, daher verdient es höchsten Respekt, dass es dort überhaupt möglich ist ein Tate-Pairing zu berechnen.

Weiterhin werden in der Arbeit auch Angaben darüber gemacht, wie viele Taktzyklen für die Berechnung einer modularen Multiplikation und Inversion auf den drei Plattformen benötigt werden. Mit Hilfe dieser Werte soll jetzt eine Abschätzung getroffen werden, wie viel Zeit Algorithmus 7 auf diesen Plattformen bei einer fiktiven Taktfrequenz von 20MHz benötigen würde, um so einen Vergleich zu den Ergebnissen auf dem Samsung ARM zu bekommen. Zuerst wird noch einmal eine Abschätzung für den Aufwand der Miller-Schleife von Algorithmus 7 unter Beibehaltung der Inversionen gemacht, da das Verhältnis $I = 15M$ bei den drei Plattformen nicht zutrifft:

$$\begin{aligned} t(\text{Miller-Schleife}) &= 160 \cdot t(\text{Punktverdopplungsblock}) + 80 \cdot t(\text{Punktadditionsblock}) \\ &= 160 \cdot (2I + 58M) + 80 \cdot (2I + 57M) \\ &= 320I + 9280M + 160I + 4560M \\ &= 480I + 13840M \end{aligned}$$

Die Anzahl der Taktzyklen beziehen sich auf ein 160-Bit-Modul, bei Algorithmus 7 wird allerdings ein 512-Bit-Modul genutzt. Daher wird die Annahme getroffen, dass die Operationen entsprechend um den Faktor $\frac{512}{160} = 3{,}2$ aufwendiger sind (wobei der Faktor für die Berechnung des Inversen vermutlich eher größer sein müsste). Mit diesen Annahmen ergeben sich folgende Werte:

$$\begin{aligned} t(\text{Miller-Schleife MICA2}) &= \frac{(480 \cdot 364291 + 13840 \cdot 7547) \cdot 3{,}2}{20\ \text{MHz}} = \frac{893792512}{20\ \text{MHz}} \approx 44{,}69\ \text{s} \\ t(\text{Miller-Schleife Tmote Sky}) &= \frac{(480 \cdot 229724 + 13840 \cdot 4734) \cdot 3{,}2}{20\ \text{MHz}} = \frac{562515456}{20\ \text{MHz}} \approx 28{,}13\ \text{s} \\ t(\text{Miller-Schleife Imote2}) &= \frac{(480 \cdot 49223 + 13840 \cdot 843) \cdot 3{,}2}{20\ \text{MHz}} = \frac{112941312}{20\ \text{MHz}} \approx 5{,}65\ \text{s} \end{aligned}$$

Auch wenn diese Abschätzung sehr vereinfachend und vermutlich etwas ungenau ist, so gibt sie doch zumindest eine Größenordnung vor, wie sich die Plattformen zueinander verhalten. Erwartungsgemäß sind die 8-Bit und 16-Bit Plattformen dem Samsung ARM mit den gemessenen 7,3 Sekunden unterlegen. Anders sieht es beim Imote2 aus, der wie der Samsung ARM ebenfalls einen 32-Bit Prozessor besitzt. Selbst wenn die Abschätzung ungenau ist, so liegt er doch scheinbar zumindest in der gleichen Größenordnung wie der Samsung ARM. Und das obwohl er keine Hardwarebeschleunigung bei der Berechnung einer modularen Multiplikation benutzt. Dieses Ergebnis ist doch etwas überraschend. Auf der anderen Seite bestärkt es die Vermutung aus Kapitel 5.4, dass bei der modularen Multiplikationsfunktion auf dem Samsung ARM noch Optimierungspotential liegen könnte.

Fazit

Der Vergleich mit Implementierungen auf anderen Plattformen hat gezeigt, dass der BKLS-Algorithmus zur Berechnung des Tate-Pairings mit unterschiedlichen Optimierungen deutlich weniger aufwendig ist als Algorithmus 7 zur Berechnung des Weil-Pairings.

Daher lassen sich damit schon einmal prinzipiell bessere Zeiten zur Berechnung eines Pairings erreichen als die hier in der Diplomarbeit gemessenen Zeiten für das Weil-Pairing.

Als nächster Schritt wäre es sehr interessant, auch auf dem Samsung ARM einmal den BKLS-Algorithmus mit den entsprechenden Optimierungen zu implementieren, um so Ergebnisse zu erhalten, die man direkt mit den anderen Arbeiten vergleichen kann, ohne Abschätzungen bemühen zu müssen.

6. Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Machbarkeitsstudie zur Berechnung einer Gruppensignatur nach Boneh und Shacham auf einer Smart Card durchgeführt. Anwendung findet die Berechnung der Gruppensignatur innerhalb des TURBINE-Projektes im Rahmen einer biometrischen Authentisierung. Kernelement der Gruppensignatur ist eine bilineare Abbildung, hierfür wurde das Weil-Pairing genutzt. In der Arbeit wurde ausführlich vorgestellt, wie sich das Weil-Pairing konkret berechnen lässt, unter anderem benötigt man dazu Arithmetik in den endlichen Körpern $\mathbb{F}_q$ und $\mathbb{F}_{q^2}$, sowie Arithmetik auf den elliptischen Kurven $E(\mathbb{F}_q)$ und $E(\mathbb{F}_{q^2})$.

Mit den Algorithmen aus Kapitel 5 wurde auf einem Samsung ARM Halbleiter eine Zeit von etwa 21 Sekunden zur Berechnung einer kompletten Gruppensignatur erzielt. Dieser Wert ist zwar noch nicht praxistauglich, allerdings wird er für die erstmalige Auseinandersetzung mit dieser komplexen Thematik als akzeptabel angesehen und liegt im Rahmen der Erwartungen, die vor dem Start der Diplomarbeit als realistisch angesehen wurden.

Ein Großteil der Gesamtzeit zur Berechnung der Gruppensignatur wurde wie erwartet für die zweifache Berechnung des Weil-Pairings benötigt. In Kapitel 5 wurden weitere Optimierungsmöglichkeiten vorgestellt, um insbesondere diesen Teil weiter zu beschleunigen, da hier natürlich das größte Einsparpotential liegt. Der nächste Schritt bestünde daher darin, diese Optimierungen konkret umzusetzen, um dadurch noch bessere Laufzeiten erzielen zu können. Von Interesse ist auch eine Untersuchung zur Nutzung des Tate-Parings anstelle des Weil-Pairings. Andere Arbeiten haben gezeigt (siehe Kapitel 5.8), dass sich damit gute Ergebnisse erzielen lassen. Die Suche nach paarungsfreundlichen elliptischen Kurven ist ein sehr aktives Forschungsgebiet. Die Wahl unterschiedlicher Kurven bietet ein weiteres Feld für zusätzliche Untersuchungen.

Falls man die prototypische Implementierung in ein echtes Produkt überführen möchte, so müssen auch noch einige Sicherheitsaspekte berücksichtigt werden. Kryptografische Algorithmen auf Smart Cards benötigen generell einen Schutz gegen SPA- und DPA-Angriffe, denn diese Angriffe sind bei Smart Cards ein gängiges Mittel, um anhand des Strom- und Spannungsverlaufs Teile des für das jeweilige Verfahren genutzten geheimen Schlüssels zu ermitteln. Daher sind die Angriffe auch für die Berechnung der Gruppensignatur eine Gefahr. Für Prototypen spielt dieser Aspekt normalerweise aber nur eine untergeordnete Rolle und wurde daher bisher nicht näher untersucht. Erkenntnisse zu diesem Themenbereich finden sich zum Beispiel in [KTH+06].

Als Fazit lässt sich vermelden, dass das Ziel der Diplomarbeit – die Berechnung der Gruppensignatur auf der vorgegebenen Smart Card – erreicht wurde. Auf dem Weg zu einem tatsächlich vermarktbaren Produkt gilt es allerdings noch eine Reihe von Herausforderungen zu meistern.

A. Anmerkungen zur Distortion-Map

In Kapitel 4.5.5 wurde für die Kurve $E : Y^2 = X^3 + 1$ über $\mathbb{F}_{q^2}$ die Distortion-Map

$$\phi : E(\mathbb{F}_q) \to E(\mathbb{F}_{q^2}),\ \phi(x, y) \mapsto (\alpha x, y),\ \text{mit } \alpha^3 = 1, \alpha \neq 1$$

definiert, wobei $\mathbb{F}_{q^2}$ entsprechend Kapitel 4.1.2 konstruiert ist. Zu zeigen sind noch folgende Punkte:

1. $\alpha = \frac{-1+i\delta}{2}$ mit $\delta = 3^{\frac{q+1}{4}} \in \mathbb{F}_q$ ist eine dritte Einheitswurzel in $\mathbb{F}_{q^2}$, für q prim, $q = 3 \mod 4$ und $q = 2 \mod 3$.
2. Sei $P \in E(\mathbb{F}_q)$. Dann ist $\phi(P) \in E(\mathbb{F}_{q^2})$.
3. $\phi(P + Q) = \phi(P) + \phi(Q) \quad \forall P, Q \in E(\mathbb{F}_q)$.

Beweis:

1. a) Zuerst wird gezeigt, dass $\delta^2 \equiv 3 \mod q$ gilt. Mit Hilfe des Quadratischen Reziprozitätsgesetzes [MOV96, 2.146] berechnet sich das Legendre-Symbol von 3 für das gewählte q wie folgt:

$$\left(\frac{3}{q}\right) = -\left(\frac{q}{3}\right) \equiv -(q^{\frac{3-1}{2}}) \equiv -q \equiv -2 \equiv 1 \mod 3.$$

Damit ist 3 ein quadratischer Rest in $\mathbb{F}_q$ und es gilt $3^{\frac{q-1}{2}} \equiv 1 \mod q$. Daraus folgt:

$$\delta^2 \equiv (3^{\frac{q+1}{4}})^2 \equiv 3^{\frac{q+1}{2}} \equiv 3^{\frac{q-1}{2}+1} \equiv 3^{\frac{q-1}{2}} \cdot 3 \equiv 1 \cdot 3 \equiv 3 \mod q.$$

b) Zu zeigen: $\alpha^3 \equiv 1 \mod q$.

$$\begin{aligned}
\alpha^3 &\equiv (\tfrac{-1+i\delta}{2})^3 \equiv \tfrac{1}{8} \cdot (-1 + i\delta)^3 \\
&\equiv \tfrac{1}{8} \cdot ((-1)^3 + 3(-1)^2 i\delta + 3(-1)(i\delta)^2 + (i\delta)^3) && \mid \delta^2 = 3 \\
&\equiv \tfrac{1}{8} \cdot (-1 + 3i\delta - 3 \cdot 3i^2 + 3i^3\delta) && \mid i^2 = -1 \\
&\equiv \tfrac{1}{8} \cdot (-1 + 3i\delta + 9 - 3i\delta) \\
&\equiv \tfrac{1}{8} \cdot (8) \\
&\equiv 1 \mod q.
\end{aligned}$$

α ist also eine nichttriviale dritte Einheitswurzel in $\mathbb{F}_{q^2}$ und insbesondere kein Element von $\mathbb{F}_q$.

2. Da $\alpha^3 = 1$, gilt:

   $\forall (x, y) \in E(\mathbb{F}_q)$ ist $\phi(x, y) \in E(\mathbb{F}_{q^2})$, denn $(\alpha \cdot x)^3 + 1 = \alpha^3 \cdot x^3 + 1 = x^3 + 1 = y^2$.

   Gleichzeitig ist α so gewählt, dass $\alpha \cdot x \notin \mathbb{F}_q$, somit ist $\phi(x, y) \notin E(\mathbb{F}_q)$.

3. Der Beweis orientiert sich an [HPS08, Behauptung 5.51].
   Seien $P = (x_1, y_1), Q = (x_2, y_2)$ zwei unterschiedliche Punkte auf $E(\mathbb{F}_q)$. Seien $\mathrm{x}(R)$ und $\mathrm{y}(R)$ zwei Funktionen, die die $x-$ beziehungsweise y-Koordinate eines Punktes R liefern. Nach der Formel für die Punktaddition (4.1) ergibt sich für die x-Koordinate:

$$\begin{aligned}
\mathrm{x}(\phi(P) + \phi(Q)) &= \left(\frac{y_2 - y_1}{\alpha x_2 - \alpha x_1}\right)^2 - \alpha x_1 - \alpha x_2 \\
&= \frac{1}{\alpha^2}\left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 + \alpha(-x_1 - x_2) \\
&= \frac{\alpha}{\alpha^3}\left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 + \alpha(-x_1 - x_2) \qquad \Big| \ \alpha^3 = 1 \\
&= \alpha\left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 + \alpha(-x_1 - x_2) \\
&= \alpha\left(\left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2\right) \\
&= \alpha \cdot \mathrm{x}(P + Q)
\end{aligned}$$

   Für die y-Koordinate ergibt sich:

$$\begin{aligned}
\mathrm{y}(\phi(P) + \phi(Q)) &= \left(\frac{y_2 - y_1}{\alpha x_2 - \alpha x_1}\right)(\alpha x_1 - \alpha x_3) - y_1 \\
&= \frac{1}{\alpha}\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \cdot \alpha(x_1 - x_3) - y_1 \\
&= \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1 \\
&= \mathrm{y}(P + Q)
\end{aligned}$$

   Analoges lässt sich auch für $P = Q$ zeigen. Somit ist $\phi(P + Q) = \phi(P) + \phi(Q)$. Als Folge davon ist dann auch $\phi(nP) = n\phi(P) \quad \forall P \in E(\mathbb{F}_q)$ und alle $n \geq 1$. Insbesondere erhält ϕ also die Ordnung eines Punktes, denn sei $P \in E(\mathbb{F}_q)$ ein Punkt der Ordnung p, dann ist $pP = \mathcal{O}$ und $p\phi(P) = \phi(pP) = \phi(\mathcal{O}) = \mathcal{O}$, also hat auch $\phi(P)$ Ordnung p.

Literaturverzeichnis

[ASN07] Antonio, Christine A. ; Satoru, Tanaka ; Nakamula, Ken: *Comparing Implementation Efficiency of Ordinary and Squared Pairings*. 2007. – http://eprint.iacr.org/2007/457

[BB08] Boneh, Dan ; Boyen, Xavier: Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. In: *Journal of Cryptology* 21 (2008), Nr. 2, S. 149–177. – http://www.cs.stanford.edu/~xb/joc07/

[BBC+08] Bertoni, Guido M. ; Breveglieri, Luca ; Chen, Liqun ; Fragneto, Pasqualina ; Harrison, Keith A. ; Pelosi, Gerardo: A pairing SW implementation for Smart-Cards. In: *Journal of Systems and Software* 81 (2008), Nr. 7, S. 1240–1247

[BBS04] Boneh, Dan ; Boyen, Xavier ; Shacham, Hovav: Short Group Signatures. In: *Advances in Cryptology—CRYPTO 2004* Bd. 3152, Springer-Verlag, 2004 (Lecture Notes in Computer Science), S. 41–55. – http://www.cs.stanford.edu/~xb/crypto04a/

[BCPZ08] Bringer, Julien ; Chabanne, Hervé ; Pointcheval, David ; Zimmer, Sébastien: An Application of the Boneh and Shacham Group Signature Scheme to Biometric Authentication. In: *IWSEC '08: Proceedings of the 3rd International Workshop on Security*, Springer-Verlag, 2008, S. 219–230

[BF03] Boneh, Dan ; Franklin, Matthew: Identity-Based Encryption from the Weil Pairing. In: *SIAM Journal on Computing* 32 (2003), Nr. 3, S. 586–615. – ISSN 0097–5397

[BHLM01] Brown, Michael ; Hankerson, Darrel ; López, Julio ; Menezes, Alfred: Software Implementation of the NIST Elliptic Curves Over Prime Fields. In: *Topics in Cryptology — CT-RSA 2001* Bd. 2020, 2001 (Lecture Notes in Computer Science), S. 250–265

[BK98] Balasubramanian, R. ; Koblitz, Neal: The Improbability That an Elliptic Curve Has Subexponential Discrete Log Problem under the Menezes-Okamoto-Vanstone Algorithm. In: *Journal of Cryptology* 11 (1998), S. 141–145

[BKLS02] Barreto, Paulo S. L. M. ; Kim, Hae Y. ; Lynn, Ben ; Scott, Michael: Efficient Algorithms for Pairing-Based Cryptosystems. In: *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, Springer-Verlag, 2002, S. 354–368

[BMW03] Bellare, Mihir ; Micciancio, Daniele ; Warinschi, Bogdan: Foundations of group signatures: formal definition, simplified requirements and a construction based on trapdoor permutations. In: Biham, Eli (Hrsg.): *Advances in*

cryptology - EUROCRYPT 2003, proceedings of the international conference on the theory and application of cryptographic techniques Bd. 2656, Springer-Verlag, Mai 2003 (Lecture Notes in Computer Science), S. 614–629

[BMX06] Blake, Ian F. ; Murty, V. K. ; Xu, Guangwu: Refinements of Miller's algorithm for computing the Weil/Tate pairing. In: *Journal of Algorithms* 58 (2006), Nr. 2, S. 134–149

[BS04] Boneh, Dan ; Shacham, Hovav: Group signatures with verifier-local revocation. In: *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, ACM Press, 2004, S. 168–177

[CH91] Chaum, David ; Heyst, Eugene van: Identity-based encryption from the Weil pairing. In: *Advances in Cryptology — Eurocrypt 1991* Bd. 547, Springer-Verlag, 1991 (Lecture Notes in Computer Science), S. 257–265

[CMO98] Cohen, Henri ; Miyaji, Atsuko ; Ono, Takatoshi: Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In: *Advances in Cryptology—ASIACRYPT'98* Bd. 1514, Springer-Verlag, 1998 (Lecture Notes in Computer Science), S. 51–65

[CSB05] Chatterjee, Sanjit ; Sarkar, Palash ; Barua, Rana: Efficient Computation of Tate Pairing in Projective Coordinate over General Characteristic Fields. In: *Information Security and Cryptology – ICISC 2004* Bd. 3506, Springer-Verlag, 2005 (Lecture Notes in Computer Science), S. 168–181

[DhSD06] Devegili, Augusto J. ; hÉigeartaigh, Colm ó ; Scott, Michael ; Dahab, Ricardo: *Multiplication and Squaring on Pairing-Friendly Fields.* 2006. – http://eprint.iacr.org/2006/471

[DORS08] Dodis, Yevgeniy ; Ostrovsky, Rafail ; Reyzin, Leonid ; Smith, Adam: Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In: *SIAM Journal on Computing* 38 (2008), Nr. 1, S. 97–139

[DSD07] Devegili, Augusto J. ; Scott, Michael ; Dahab, Ricardo: Implementing Cryptographic Pairings over Barreto-Naehrig Curves. In: *Pairing-Based Cryptography – Pairing 2007* Bd. 4575, Springer-Verlag, 2007 (Lecture Notes in Computer Science), S. 197–207

[FST06] Freeman, David ; Scott, Michael ; Teske, Edlyn: *A taxonomy of pairing-friendly elliptic curves.* 2006. – http://eprint.iacr.org/2006/372.pdf

[Gem05] Gemplus: *ID based cryptography and Smartcards.* 2005. – http://www.gemplus.com/smart/rd/pubblications/pdf/Joy05iden.pdf

[GS06] Granger, R. ; Smart, N.P.: *On Computing Products of Pairings.* Cryptology ePrint Archive, Report 2006/172, 2006. – http://eprint.iacr.org/2006/172

[Hen04] Henhapl, Birgit: *Zur Effizienz von Elliptische-Kurven-Kryptographie*, Technische Universität Darmstadt, Dissertation, 2004. – http://www.cdc.informatik.tu-darmstadt.de/reports/reports/BirgitHenhapl.diss.pdf

[HMV04] Hankerson, Darrel ; Menezes, Alfred J. ; Vanstone, Scott: *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., 2004

[HPS08] Hoffstein, Jeffrey ; Pipher, Jill ; Silverman, Joseph H.: *An Introduction to Mathematical Cryptography*. Springer-Verlag, 2008 (Undergraduate Texts in Mathematics)

[IT03] Izu, Tetsuya ; Takagi, Tsuyoshi: Efficient Computations of the Tate Pairing for the Large MOV Degrees. In: *Information Security and Cryptology – ICISC 2002* Bd. 2587, Springer-Verlag, 2003 (Lecture Notes in Computer Science), S. 283–297

[JM03] Jain, Anil K. ; Maltoni, David: *Handbook of Fingerprint Recognition*. Springer-Verlag New York, Inc., 2003. – ISBN 0387954317

[Jou02] Joux, Antoine: The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems. In: *Proceedings of the 5th International Symposium on Algorithmic Number Theory* Bd. 2369, Springer-Verlag, 2002 (Lecture Notes In Computer Science), S. 20–32

[Jou04] Joux, Antoine: A One Round Protocol for Tripartite Diffie-Hellman. In: *Journal of Cryptology* 17 (2004), Nr. 4, S. 263–276

[KM05] Koblitz, Neal ; Menezes, Alfred: Pairing-based Cryptography at High Security Levels. In: *Proceedings of Cryptography and Coding 2005* Bd. 3796, Springer-Verlag, 2005 (Lecture Notes in Computer Science), S. 13–36

[KTH+06] Kim, Tae H. ; Takagi, Tsuyoshi ; Han, Dong-Guk ; Kim, Ho W. ; Lim, Jongin: *Side Channel Attacks and Countermeasures on Pairing Based Cryptosystems over Binary Fields*. Cryptology ePrint Archive, Report 2006/243, 2006. – http://eprint.iacr.org/2006/243

[Lyn07] Lynn, Ben: *On the Implementation of Pairing-based Cryptosystems*, Standford University, Dissertation, 2007

[Mau05] Maurer, Ueli: Abstract Models of Computation in Cryptography. In: *Cryptography and Coding* Bd. 3796, Springer-Verlag, 2005 (Lecture Notes in Computer Science), S. 1–12

[Men09] Menezes, Alfred: An introduction to pairing-based cryptography. In: *Contemporary Mathematics* 477 (2009), S. 47–65. – http://www.cacr.math.uwaterloo.ca/~ajmeneze/publications/pairings.pdf

[Mil86] Miller, Victor S.: *Short Programs for functions on Curves*. 1986. – http://crypto.stanford.edu/miller/

[Mil04] Miller, Victor S.: The Weil Pairing, and Its Efficient Calculation. In: *Journal of Cryptology* 17 (2004), Nr. 4, S. 235–261

[MO90] Morain, François ; Olivos, Jorge: Speeding Up The Computations On An Elliptic Curve Using Addition-Subtraction Chains. In: *Theoretical Informatics and Applications* 24 (1990), S. 531–543

[Mon85] Montgomery, Peter L.: Modular Multiplication Without Trial Division. In: *Mathematics of Computation* 44 (1985), S. 519–521. – http://www.jstor.org/stable/2007970

[MOV96] Menezes, Alfred J. ; Oorschot, Paul C. ; Vanstone, Scott A.: *Handbook of applied cryptography*. CRC Press, 1996. – http://cacr.math.uwaterloo.ca/hac

[PKY05] Park, Cheol M. ; Kim, Myung H. ; Yung, Moti: A Remark on Implementing the Weil Pairing. In: *Information Security and Cryptology* Bd. 3822, Springer-Verlag, 2005 (Lecture Notes in Computer Science), S. 313–323

[RE99] Rankl, Wolfgang ; Effing, Wolfgang: *Handbuch der Chipkarten*. Carl Hanser Verlag, 1999

[SBC+09] Scott, Michael ; Benger, Naomi ; Charlemagne, Manuel ; Dominguez Perez, Luis J. ; Kachisa, Ezekiel J.: Fast Hashing to G2 on Pairing-Friendly Curves. In: *Pairing '09: Proceedings of the 3rd International Conference Palo Alto on Pairing-Based Cryptography*, Springer-Verlag, 2009, S. 102–113

[SCA06] Scott, Michael ; Costigan, Neil ; Abdulwahab, Wesam: Implementing Cryptographic Pairings on Smartcards. In: *Cryptographic Hardware and Embedded Systems - CHES 2006* Bd. 4249, Springer-Verlag, 2006 (Lecture Notes in Computer Science), S. 134–147

[Sco05a] Scott, Michael: Computing the Tate Pairing. In: *Topics in Cryptology – CT-RSA 2005* Bd. 3376, Springer-Verlag, 2005 (Lecture Notes in Computer Science), S. 293–304

[Sco05b] Scott, Michael: *Deterministic hashing to points on IBE-friendly elliptic curves*. 2005. – ftp://ftp.computing.dcu.ie/pub/resources/crypto/cth.pdf

[Sho97] Shoup, Victor: Lower Bounds for Discrete Logarithms and Related Problems. In: *Advances in Cryptology — EUROCRYPT '97* Bd. 1233, Springer-Verlag, 1997 (Lecture Notes in Computer Science), S. 256–266

[SKSC09] Szczechowiak, Piotr ; Kargl, Anton ; Scott, Michael ; Collier, Martin: On the application of pairing based cryptography to wireless sensor networks. In: *WiSec '09: Proceedings of the second ACM conference on Wireless network security*, ACM, 2009, S. 1–12

[SOS+08] Szczechowiak1, Piotr ; Oliveira, Leonardo B. ; Scott, Michael ; Collier, Martin ; Dahab, Ricardo: NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks. In: *Wireless Sensor Networks* Bd. 4913, 2008 (Lecture Notes in Computer Science), S. 305–320

[TUR08] TURBINE Consortium: *TURBINE D.1.2.1 Services and schemes for multiple trusted identity*. 2008

[TUR09] TURBINE Consortium: *TURBINE D.3.2.2 Architecture Implementation*. 2009. – confidential