INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Evaluation effizienter Schlüsselbäume für hierarchische identitätsbasierte Signaturen im IoT

Tobias Treutner

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Evaluation effizienter Schlüsselbäume für hierarchische identitätsbasierte Signaturen im IoT

Tobias Treutner

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Nils gentschen Felde Tobias Guggemos
Abgabetermin:	12. Juni 2018

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 12. Juni 2018

. . . . . .
(Unterschrift des Kandidaten)

Abstract

In der stetig wachsenden IoT spielt die Sicherheit eine immer größere Rolle. Um den Absender einer Nachricht zu identifizieren oder Integrität zu gewährleisten, kommen asymmetrische Signaturverfahren zum Einsatz. Durch die beschränkten Ressourcen im IoT sind die meisten herkömmlichen Verfahren aufgrund des Speicherverbrauchs und des Rechenaufwands keine Option. Hierarchische identitätsbasierte Signaturen, die auf *Elliptic Curve Cryptography* beruhen, bieten ein hohes Sicherheitsneveau bei geringem Ressourcenverbrauch. In HIBS ist jeder private Schlüssel von dem seines Vorgängers abhängig, sodass die Gültigkeit eines Zertifikats mathematisch validiert werden kann. Durch die Gruppenorganisation in Baumstrukturen wird die Last der Schlüsselgeneration auf mehrere Gruppenmitglieder verteilt. Aufgrund der mathematischen Abhängigkeit der Schlüssel in herkömmlichen IBS-Verfahren, müssen alle Schlüssel im Falle eines Gruppenaustritts erneuert werden. HIBS hingegen verringert die Anzahl der betroffenen Gruppenmitglieder durch die Organisation ihrer Schlüssel in Baumstrukturen.

Diese Arbeit beschäftigt sich mit effektiven Schlüsselbäumen zur Gruppenorganisation in einem HIBS. Dazu wurde ein logischer und ein rollenbasierter Prototyp zum Schlüsselmanagement in einer Baumstruktur designed, implementiert und evaluiert. Beide Protokolle basieren auf der Kryptografiebibliothek Relic und sind für das eingebettete Betriebssystem RIOT entwickelt.

Inhaltsverzeichnis

1 Einleitung

In der Vergangenheit bestanden Rechnernetze meist nur aus Computern, die über *Local area Networks* oder dem Internet verbunden waren. Doch in den letzten 20 Jahren wurden immer kleinere Geräte miteinander vernetzt. Professor Ashotn, der Leiter des MIT „Auto ID-Labs“ erkannte bereits 1999, dass es sich nicht mehr um das herkömmliches Internet handelt, bei dem der Datenaustausch meist vom Menschen initiiert wird. Viel mehr sind nun eingebettete Systeme, die z.B. Sensorwerte überwachen, über Funk vernetzt. Da die Geräte nun völlig autark miteinander kommunizieren, gab er dem Netzwerk den Namen *„Internet of Things“* oder auf deutsch *„Internet der Dinge“*. [PS12]

Die US Amerikanische Behörde NIC *„National Intelligence Council“* erwartete schon 2008, dass 2025 Alltagsgegenstände wie zum Beispiel Lebensmittelverpackungen, Möbel und schriftliche Dokumente internetfähig sein werden. Des Weiteren rechnete sie damit, dass die Möglichkeit des Fernzugriffes auf Alltagsgegenstände auch erhebliche Sicherheitsrisiken mit sich bringen wird. (vgl.[Cou08]) Jetzt ist erst knapp mehr als die Hälfte der Zeit bis 2025 verstrichen und die Vernetzung hat berreits in Autos, Smart Homes und Sensornetzen in der Industrie Einzug erhalten. Diese Systeme arbeiten oft mit sensiblen Daten, die vor dem Abhören oder Änderungen geschützt werden müssen, weshalb die Sicherheit eine besonders große Rolle spielt. Schließlich es wäre katastrophal, wenn unberechtige Personen oder Schadsoftware z.B. in einem Auto einen Airbag auslösen und eine Vollbremsung einleiten könnte. Oder ein Eibrecher in der Lage wäre ein intelligentes Türschloss durch ein gefälschtes Signal zu entriegeln. Solch einen Angriff auf die Smart Home Glühbirne Hue von Phillips demonstrierten 2016 zwei Hacker und ein Doktorand der Kryptografielegende Adi Shamir. Sie flogen mit einer Drohne an ein Bürogebäude herran und konnten schon aus einem Abstand von ca. 200 Metern die Hue Birnen nach belieben aus- und anschalten. Auch wenn diese Schwachstelle auf den ersten Blick nur ein schwaches Sicherheitsrisiko darstellt, so kann sie dennoch dazu benutzt werden, um z.B. Daten aus einem Hochsicherheitsbereich über das an- und ausschalten der Lampe nach außen zu übertragen [Rie16].

Um solche Angriffe zu vermeiden kommen moderne kryptografische Verfahren zum Einsatz. Zur Identifikation des Nachrichtensenders werden digitale Zertifikate verwendet, die auf der asymmetrischen Public-Key Kryptografie beruhen. Da die meisten eingebetteten Systeme von einer Batterie gespeist werden und über einen Mikroprozessor verfügen der nur spartanisch mit Speicher und Rechenleistung ausgestattet ist, sind Speichereffektivität und geringer Rechenaufwand, neben der Sicherheit, die Hauptanforderungen an ein Public-Key Verfahren im IoT. Die etablierten Verfahren DSA und RSA die u.A. für SSH oder die x.509 Zertifikate im Internet verwendet werden, basieren auf sehr rechenaufwendigen und speicherintensiven mathematischen Problemen mit großen Primzahlen. Aufgrund der Hardwarebeschränkungen im IoT sind sie für viele eingebettete Systeme keine Option. Eine effektivere Ressourcenallokation bietet die „*Elliptic Curve Cryptography* (ECC)“. Ein „*Elliptic Curve Digital Signature Algorithm* (ECDSA)“ erreicht ein äquivalentes Sicherheitsniveau wie DSA/RSA bei wesentlich geringeren Schlüssellängen und Rechenaufwand. [Inf18]

Daher bieten sich Signaturverfahren, die auf ECC beruhen, besonders zum Einsatz im

IoT an. Das Ziel der Arbeit ist die Implementierung und Bewertung eines „*Hierarchical Identity-Based Signature* (HIBS)“ Schemas für das eingebettete Systeme mit dem RIOT Betriebssystem. Dazu wurde das im Paper „Hierarchical ID-Based Cryptography“ von Craig Gentry und Alice Silverberg vorgestellte HIBS [GS02] mithilfe der Kryptografiebibliothek Relic [AG] implementiert. Des Weiteren wurden zwei verschiedene Protokolle zum Schlüsselmanagement in einer Baumstruktur entworfen und implementiert. Ein Protokoll basiert auf einer rollenbasierten Baumstruktur und das Andere auf einem logischen Schlüsselbaum. Anschließend erfolgt eine Evaluation bzgl. auszutauschenden Nachrichten, Rechenaufwand und Speicherverbrauch beim Gruppenein- und Austritt.
Die Arbeit ist folgerndermaßen strukturiert. Im Grundlagenkapitel 2 wird der verwendete mathematische Hintergrund zu der ECC und digitalen Signaturverfahren 2.2 erklärt. Das Designkapitel 3 geht auf die Anforderungen an das Design eines hierarchischen identitätsbasierten Signaturverfahrens, sowie auf zwei verschiedene Ansätze zum Gruppenmanagement in einer Baumstruktur ein. Die Schlüsselstellen der Implementierungen werden in Kapitel 4 beschrieben und anschließend gegen die Anforderungen aus dem vorherigen Kapitel evaluiert. Kapitel 5 rundet diese Arbeit durch eine Zusammenfassung und einen Ausblick ab.

2 Grundlagen

In diesem Kapitel beschäftigt sich diese Arbeit mit den zum Verständnis nötigen Grundlagen. Zu Beginn wird auf das *Internet of Things* und Grundlagen zu digitalen Signaturen eingegangen. Ab Kapitel 2.3 werden die mathematischen Grundlagen zu zyklischen Gruppen, elliptischen Kurven und Pairings erläutert. Kapitel 2.3.3 behandelt die Diffie-Hellman Probleme, und unter 2.2 werden zu identitätsbasierte Signaturen erläutert. In Kapitel 2.5 *Related Work* werden weitere Verfahren aufgezeigt, die sich mit der Kryptografie befassen.

2.1 Internet of Things

Der Begriff *Internet of Things* lässt schon darauf schließen, dass es um die Vernetzung verschiedener „Dinge“ geht, z.B. in Autos oder Alltagsgegenständen in Smart Homes. Diese sollen dem Menschen, ohne in abzulenken oder gar aufzufallen, den Alltag erleichtern. Die Unterstützung reicht von der Ausgabe einer Warnung oder eines Hinweises bis zur automatischen Steuerung von Anlagen. Auf die Entstehung und konkrete Beispiele für das IoT wurde bereits in der Einleitung (Kapitel 1) eingegangen. Dies wird daher an dieser Stelle nicht weiter ausgeführt. Eingebettete Systeme sind nur spartanisch mit Rechenleistung und Speicher ausgestattet, da meist nur Sensorwerte ausgelesen, und anschließend über Funk übermittlet werden. Um nicht für alle verschiedenen Mikrocontroller leichtgewichtige Netzwerkprotokolle wie IEEE 802.15.4 6LoWPAN [SB09] entwickeln zu müssen, gibt es leichtgewichtige Betriebssysteme für eingebettete Systeme. Die Herausforderung beim Design eines solchen Betriebssystems ist es den optimalen Kompromiss zwischen Portierbarkeit, Funktionalität und Ressourcenverbrauch einzugehen. Drei eingebettete Betriebssysteme werden in Kapitel 4.1.3 näher vorgestellt. Eine weiteres Problem ist die Hardwareausstattung in puncto Sicherheit. Moderne kryptographische Verfahren sind sehr rechen- und speicherintensiv. Daher ist bei der Wahl des kryptographischen Verfahrens auf den Speicher- und Rechenaufwand bei der Keygeneration, Signierung und Verifikation zu achten. Denn es macht wenig Sinn, wenn z.B. ein Arduino Uno [AUN] mit 16 Mhz und 2 kB RAM Zehn Minuten zum Erstellen einer Signatur benötigt. In dieser Arbeit wurde sich deshalb bewusst für die Kryptografiebibliothek Relic (Kapitel 4.2) entschieden, da diese bereits für das eingebettete Betriebssystem RIOT portiert ist, und so maximale Effizienz durch eine C Implementierung verspricht.

2.2 Digitale Signaturen

Kryptographie im Allgemeinen zielt auf den Schutz von gespeicherten oder übertragenen Daten ab. Diese können laut ISO 27001 hinsichtlich von vier Gesichtspunkten geschützt werden.

1. **Vertraulichkeit:** Unberechtigten Personen soll es unmöglich gemacht werden, Daten oder Nachrichten auszulesen.

2. **Integrität:** Daten müssen nachweislich vor Veränderungen geschützt sein.

3. **Authentizität:** Der Verfasser der Daten/Nachricht muss nachweislich identifizierbar sein.

4. **Verbindlichkeit:** Der Verfasser darf nicht in der Lage sein, seine Urheberschaft zu leugnen.

Eine digitale Signatur hat die wesentlichen Eigenschaften einer handschriftlichen Unterschrift. Durch sie kann die Integrität, Authentizität und Verbindlichkeit einer Nachricht bestätigt werden. Da bei einer Signatur nicht die ganze Nachricht verschlüsselt wird, ist die Vertraulichkeitseigenschaft keinesfalls garantiert. Sie besteht aus dem Hash einer Nachricht, der mithilfe des privaten Schlüssels eines asymmetrischen Verschlüsselungsverfahrens verschlüsselt wird. Der Empfänger entschlüsselt die digitale Signatur und berechnet seinerseits den Hash der Nachricht. Anschließend vergleicht er ob entschlüsselte Signatur und berechneter Hash gleich sind. Das asymmetrische Verschlüsselungsverfahren muss gewährleisten, dass die Signatur nur durch Kenntnis des geheimen Schlüssels generiert werden kann. Zur Verteilung der öffentlichen Schlüssel wird üblicherweise eine *Public-Key-Infrastruktur* (PKI) genutzt, bei der die Schlüssel über einen sicheren Kanal ausgetauscht werden. Der Ablauf eines digitalen Signaturverfahrens wird in der Abbildung 2.1 visualisiert.

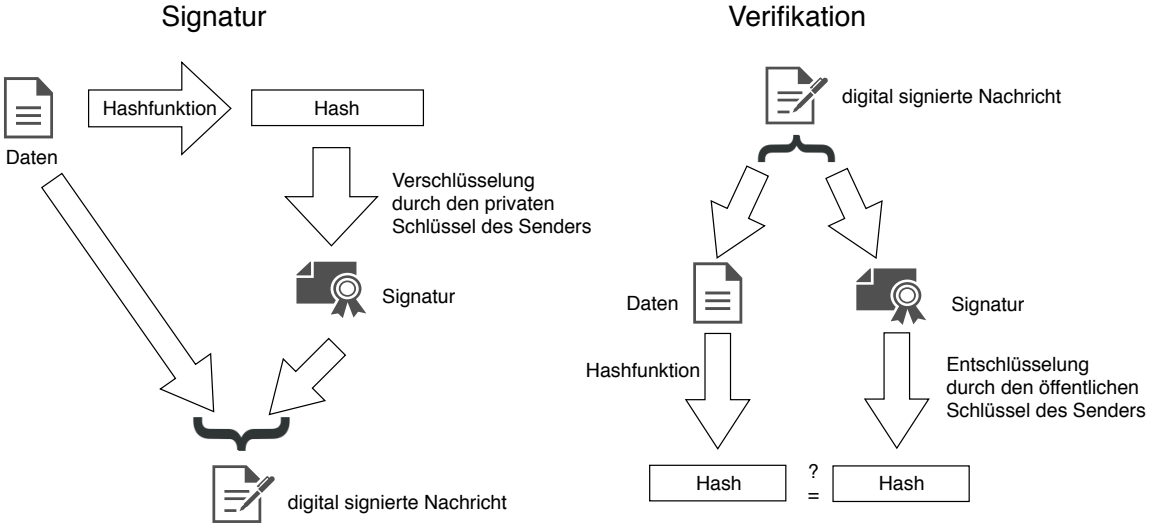

Abbildung 2.1: Signierung und Verifikation

2.3 Mathematische Grundlagen

Die moderne Kryptographie basiert auf komplexen Problemen aus der Mathematik. Dieses Kapitel erläutert die, für identitätsbasierte Signaturen relevanten, mathematischen Grundlagen.

2.3.1 Gruppen

Die verwendeten Definitionen zu Gruppen und zyklischen Gruppen entsprechen [MSP11].

Definition 2.1. *Eine Gruppe ist eine Menge G, die mit einem Operator $\circ$ folgender maßen verknüpft ist: $\circ : G \times G \to G$, so dass folgende Eigenschaften erfüllt sind:*

- *Es gibt ein neutrales Element $\mathbb{1}$: $\forall g \in G \; \exists \, \mathbb{1} \in G : \mathbb{1} \circ g = g \circ \mathbb{1} = g$.*
- *Es gibt ein inverses Element: $\forall g \in G \; \exists \, g^{-1} \in G : g \circ g^{-1} = g^{-1} \circ g = \mathbb{1}$.*
- *Die Verknüpfung ist assoziativ: $\forall g, h, r \in G : g \circ (h \circ r) = (g \circ h) \circ r$.*

Des Weiteren heißt eine Gruppe G abelsch (kommutativ) falls $\forall g, h \in G : g \circ h = h \circ g$ gilt.

Anstatt g $\circ$ h wird bei additiven Gruppen die Schreibweise g + h, und bei multiplikativen Gruppen g $\cdot$ h verwendet.

Zyklische Gruppen

Man spricht von einer zyklischen Gruppe, wenn eine Gruppe alle Eigenschaften aus Definition 2.1 und Definition 2.1.1 erfüllt.

Definition 2.1.1. *Eine zyklische Gruppe G kann aus einem einzigen Element, dem Generator, erzeugt werden. Das heißt G ist zyklisch falls $\exists \, a \in G : G = \langle a \rangle$.*

Ein Beispiel für eine multiplikative zyklische Gruppe ist die ($\mathbb{Z}_p$, *) mit $p = 7$ und dem Generator $\langle a \rangle = \langle 5 \rangle$. Wobei die Multiplikation wie folgt definiert ist: $x * y = x * y \; mod \; p$.

i	1	2	3	4	5	6
$a^i \; mod \; p$						
$5^i \; mod \; 7$	5	4	6	2	3	1

Tabelle 2.1: Zyklische Gruppe

Da $a^6 = 1$ ist $a^7 = a^6 * a^1$. Dies kann auf jedes beliebige i $\in \mathbb{N}$ angewandt werden. Man nehme z.B. das Produkt aus $a^8 * a^9$:

$$a^8 * a^9 = a^{8+9} = a^{17} = a^6 * a^6 * a^5 = 1 * 1 * 3$$

Endlicher Körper (Galoiskörper)

Ein endlicher Körper(engl. „*finite field*“) beinhaltet nur eine endliche Anzahl an Elementen, auf denen die Multiplikation und Addition definiert sind ($\mathbb{Z}_p, +_p, *_p$). Das einfachste Beispiel

$+_2$	0	1
0	0	1
1	1	0

$*_2$	0	1
0	0	0
1	0	1

Tabelle 2.2: Galoiskörper

hierfür ist der Körper $(\mathbb{Z}_2, +_2, *_2)$, der in Tabelle 2.2 abgebildet ist. Dort lautet die Berechnungsvorschrift $x + y \; mod \; p$, so dass z.B. $1 + 1 \; mod \; 2 = 0$.

Endliche Körper die aus p Elementen bestehen, wobei p eine Primzahl ist, werden **Primkörper** (engl. „*prime field*“) genannt.

2.3.2 Elliptische Kurven

Elliptische Kurven wurden erstmals 1985, in den voneinander unabhängigen Arbeiten von Victor S. Miller [Mil85] und Neal Koblitz [Kob87], als Basis für das mathematische *diskrete Logarithmus Problem* (Kapitel 2.3.3) eingesetzt. Aufgrund der geringen Schlüssellänge eignen sich elliptische Kurven besonders gut zum Einsatz in der Kryptografie. Die herkömmlichen asymmetrischen Verfahren RSA/DSA benötigen für das gleiche Sicherheitsniveau, wie ein 200 Bit ECDSA, einen Schlüssel mit einer Mindestlänge von 1900 Bit [Inf18]. In dieser Arbeit werden Elliptische Kurven E über einem endlichen Körper $\mathbb{F}_p$ mit $p \neq 2, 3$ und $a, b \in \mathbb{F}_p$ der Form der kurzen Weierstraß-Gleichung

$$y^2 = x^3 + ax^2 + b \; mod \; p \quad \wedge \quad 4a^3 + 27b^2 \neq 0 \tag{2.1}$$

verwendet. Denn $p = 2 \vee p = 3$ stellen Ausnahmefälle dar, und liefern unterschiedliche elliptische Kurvengleichungen. Im Fall von $p = 2$ würde die Gleichung $y^2 + cy = x^3 + ax + b$ lauten und bei $p = 3 : y^2 = x^3 + ax^2 + bx + c$. Als Identitätselement wählt man einen Punkt im Unendlichen. Dieser erhält die Notation $\mathcal{O}$ und lässt sich nicht durch die Gleichung 2.1 darstellen. Für die Punktaddition, die im Abschnitt 2.3.2 beschrieben wird, muss die Kurve an jedem Punkt differenzierbar sein. Daher müssen Singularitäten (Definitionslücken) ausgeschlossen werden, indem a, b die zusätzliche Bedingung $4a^3 + 27b^2 \neq 0$ erfüllen. Die Gleichung

$$E(\mathbb{F}_p) = \{(x, y) \in \mathbb{F}_p^2 : y^2 = x^3 + ax + b \; mod \; p\} \cup \{\mathcal{O}\} \tag{2.2}$$

beschreibt die Menge aller rationalen Punkte auf der elliptischen Kurve E über einem endlichen Körper $\mathbb{F}_p$. [Inf12]

Gruppengesetz und Punktaddition

Eine elliptische Kurve lässt sich als Gruppe darstellen, wenn die Gruppeneigenschaften aus Kapitel 2.3.1 erfüllt werden. Um auf der Kurve rechnen zu können, wird als Rechenoperator die Addition mit der Notation $\oplus$ verwendet. Dazu werden zwei Punkte, $P = (x_P, y_P)$ und $Q = (x_Q, y_Q)$, auf der elliptischen Kurve addiert und ergeben einen dritten Kurvenpunkt $R = (x_R, y_R)$ mit $P, Q, R \in E(\mathbb{F}_p)$. Die Rechenregeln sind wie folgt definiert [Inf12]:

1. Das Identitätselement $\mathcal{O}$ ist das rechts- und linksneutrale Element:
   $P = \mathcal{O} + P = P + \mathcal{O} = P$.

2. Für jeden Punkt $P = (x_P, y_P)$ gibt es einen inversen Punkt mit $-P = (x_P, -y_P)$, der an der Abszissenachse gespiegelt ist. Zusätzlich gilt $-\mathcal{O} = \mathcal{O}$ und die Addition zweier Punkte mit gleicher Abszisse $P + (-P) = \mathcal{O}$.

3. Die **Addition** zweier verschiedener Punkte P, Q ist durch die Formel $-R = P \oplus Q$ definiert. Zwei Punkte gelten als verschieden falls $\forall Q, P \in E(\mathbb{F}_p), \; \forall R \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\} :$
   $P \neq \mathcal{O}, \; Q \neq \mathcal{O}, \; P \neq \pm Q$, d.h. die Abszissen x_P und x_Q der Punkte P und Q

verschieden sind.
Nun wird die Steigung λ der Sekanten durch P und Q und dem daraus resultierenden dritten Kurvenschnittpunkt $-R$ berechnet.

$$\lambda = \frac{y_Q - y_P}{x_Q - x_P}$$

Anschließend lassen sich die Koordinaten x_R und y_R von R, durch folgende Formeln ermitteln:

$$x_R = \lambda^2 - x_P - x_Q, \quad y_R = \lambda(x_P - x_R) - y_P$$

4. Bei der **Punktverdoppelung** oder auch Addition mit sich selbst, muss zwischen zwei Fällen unterschieden werden.
   - **Fall 1** $P \neq \mathcal{O}, P \neq -P$: Da der Multiplikationsoperator $\otimes$ auf einer elliptischen Kurve nicht definiert ist, entspricht die Addition mit sich selbst $-R := P \oplus P$ der Punktverdoppelung. $-R$ ist somit der inverse Schnittpunkt der Tangente am Punkt P mit der elliptischen Kurve $E(\mathbb{F}_p)$. In Formeln ausgedrückt berechnet sich die Steigung λ der Tangente:
     $$\lambda = \frac{3x_P^2 + a}{2y_P}$$
     Wobei a der Parameter von $E(\mathbb{F}_p)$ ist. Nun lassen sich Abszisse und Ordinate von R durch folgende Formeln berechnen:
     $$x_R = \lambda^2 - 2x_P, \quad y_R = \lambda(x_P - x_R) - y_P$$
   - **Fall 2** $P \neq \mathcal{O}, P = -P$: Das heißt, $y_P = 0$ und somit gilt nach Regel 2 $P + (-P) = \mathcal{O}$

Rechenbeispiel Punktverdoppelung

Zur Veranschaulichung wird die algebraische Punktverdoppelung von $P = 2P = P \oplus P$ anhand eines Beispiels mit der elliptischen Kurve $E(\mathbb{F}_{17}) : y^2 = x^3 - 6x + 10 \ mod\ 17$ und dem Punkt $P(5, 3)$ demonstriert.

$$\lambda = \frac{3x_p^2 - 6}{2y_p} = \frac{3 * 5^2 - 6}{2 * 3} = (2 * 3)^{-1} * (3 * 25 - 6) = 6^{-1} * 69 \equiv 3 * 1 \equiv 3 \ mod\ 17$$

$$x_R = \lambda^2 - 2x_P = 3^2 - 2 * 5 \equiv -1 \ mod\ 17$$

$$y_R = \lambda(x_P - x_R) - y_P = 3 * (5 + 1) - 3 \equiv 15 \ mod\ 17$$

Somit ergibt sich $2P = (-1, 15)$ wobei 3 das multiplikative inverse Element von 6 bei mod 17 ist.

Geometrische Punktaddition

Die einzelnen Schritte der geometrischen Punktaddition, die anhand der Beispielkurve $y^2 = x^3 - 6x + 10$ in Abbildung 2.3.2 veranschaulicht werden, lassen sich gut in einer *step-by-stp* Anleitung beschreiben:

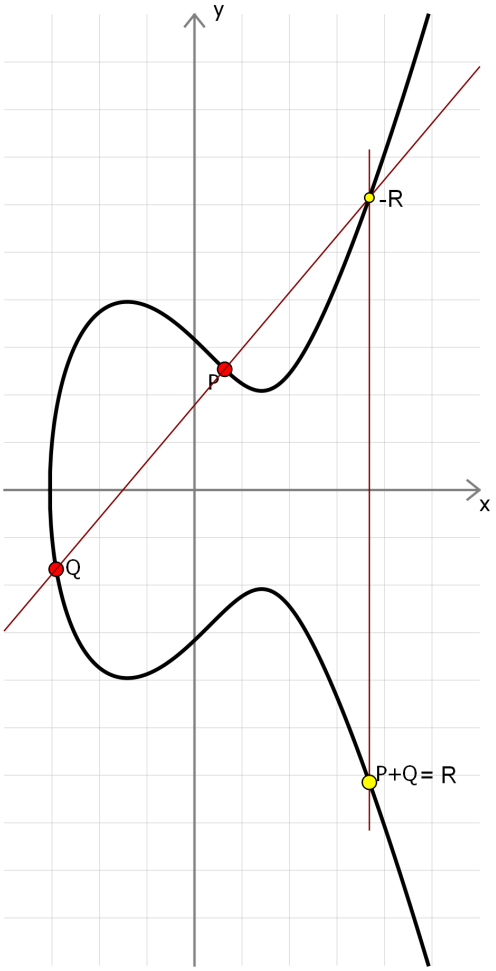

Abbildung 2.2: Punktaddition

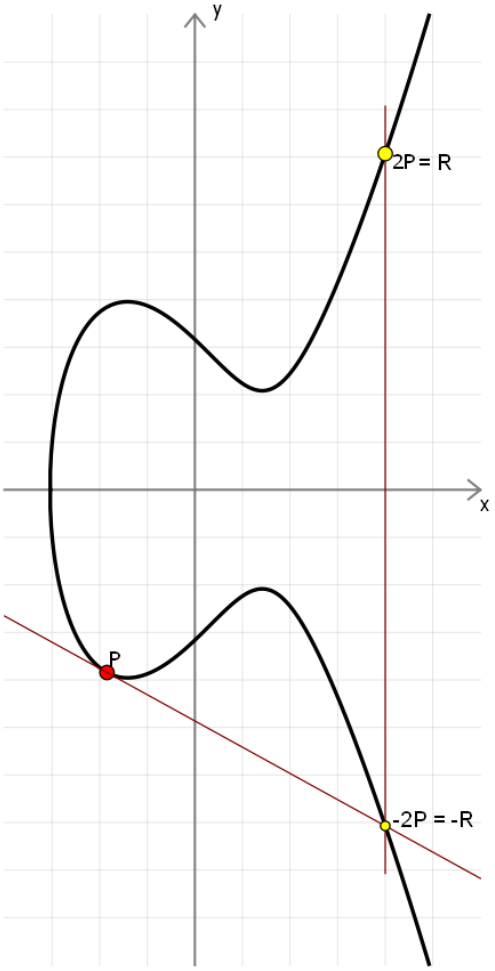

Abbildung 2.3: Punktverdoppelung

1. Zeichne die Gerade $\overline{PQ}$ durch die Kurvenpunkte P und Q.
2. Ermittle den dritten Schnittpunkt $-R$ von $\overline{PQ}$ mit der elliptischen Kurve $E(\mathbb{F}_p)$.
3. Spiegle den Kurvenpunkt $-R$ an der Abszisse und erhalte somit R.

Geometrische Punktverdoppelung

Die Vorgehensweise zur geometrischen Punktverdoppelung (Abbildung 2.3.2) erfolgt analog zur Grafischen Punktaddition 2.3.2 mit dem Unterschied, dass in Schritt 1 der *step-by-step* Anleitung, anstatt einer Geraden durch P und Q, die Tangente t an Punkt P gezeichnet wird. In Schritt 2 wird nun der Schnittpunkt $-R$ der Tangenten t und der Kurve $E(\mathbb{F}_p)$ ermittelt.

2.3.3 Mathematische Berechnungsprobleme

Die Sicherheit in der Kryptographie beruht auf mathematischen Problemen, die von keinem bekannten Algorithmus schneller als in Polynomialzeit gelößt werden können. Genauer gesagt geht es um Funktionen $y = f(x)$, die in Polynomialzeit zu lößen sind. Aber bei gegebenem y keine bekannte Umkehrfunktion $f^{-1}(y) = x$ existiert, die x schneller als in Polynomialzeit berechnen kann. Solche Funktionen werden auch Einwegfunktionen genannt. Ein Beispiel aus dem Alltag wäre die Suche nach einer Telefonnummer bei bekanntem Namen, in einem klassischen Telefonbuch. Die Telefonnummer lässt sich recht schnell ermitteln, da das Telefonbuch alphabetisch sortiert ist. Doch die Rückwärtssuche nach dem Namen zu einer gegeben Nummer ist mit erheblichem Aufwand verbunden, denn im *worst case* muss das ganze Telefonbuch durchsucht werden.

Diskreter Logarithmus

Im Gegensatz zu RSA, dessen mathematisches Problem die Primfaktorzerlegung sehr großer Zahlen (mehr als 100-stellige Dezimalzahlen) ist, kommt in der ECC das diskrete Logarithmus Problem zum Einsatz. Der diskrete Logarithmus ist durch die Formel $a^x \equiv m \ mod \ p$ definiert. Wobei $x \in \mathbb{Z}_p$ und $a, m, p \in \mathbb{N}$ sowie p eine Primzahl ist. Das *diskrete Logarithmus Problem* **DLP** ist wie folgt definiert. [McC90]

DLP: In einigen zyklischen Gruppen ist es schwer, bei gegebenem $g \in G,\ a \in \langle g \rangle$ ein $x \in \mathbb{N}$ zu finden, sodass $g^x = a$ gilt.

In der ECC werden die mit DLP verwandten *Computational Diffie Hellman Problem* **CDHP** und *Decisional Diffie-Hellman Problem* **DDHP** verwendet.[Che06]

CDHP: Bei einer gegebenen zyklischen Gruppe $\mathbb{G} = \langle g \rangle$ und den Werten g^a, g^b darf ein Angreifer nicht in der Lage sein $g^a b$ zu berechnen.

DDHP: Bei einer gegebenen zyklischen Gruppe $\mathbb{G} = \langle g \rangle$ und den Werten g^a, g^b, g^c darf ein Angreifer nicht feststellen können ob $g^c = g^{ab}$

2.3.4 Pairings

Pairings oder im Deutschen auch *bilineare Abbildungen* kamen zum ersten mal 2004, im Paper „A One Round Protocol for Tripartite Diffie-Hellman“ von Antoine Joux [Jou04], in der modernen Kryptographie zum Einsatz. Joux erkannte dass Pairings die Komplexität des diskreten Logarithmus Problems verringern. Dies machten sich Boneh, Lynn und Shacham zu nutze und entwarfen ein erstes Pairing basiertes Signaturschema [BLS04]. Ein Pairing ist eine biliniare Abbildung der Form

$$e \ : \ \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$$

wobei $\mathbb{G}_1$ und $\mathbb{G}_2$ additive, und $\mathbb{G}_T$ multiplikative zyklische Gruppen sind, die folgende Bedingungen erfüllen [BLS04]:

- Bilinearität: $\forall u \in \mathbb{G}_1, v \in \mathbb{G}_2, a, b \in \mathbb{Z} : e(u^a, v^b) = e(u, v)^{ab}$
- Nicht-Degenerativität: $\forall g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2 : e(g_1, g_2) \neq \mathbb{1}$

- Berechenbarkeit: Die bilineare Abbildung e muss durch einen effizienten Algorithmus berechenbar sein.

Aufgrund der Bilinearität von Pairings gelten $\forall S \in \mathbb{G}_1, T \in \mathbb{G}_2$ folgende Rechenregeln:

- $e(S, \infty) = \mathbb{1}$ und $e(\infty, T) = \mathbb{1}$
- $e(S, -T) = e(-S, T) = e(S, T)^{-1}$
- $\forall a, b \in \mathbb{Z} : e(aS, bT) = e(S, T)^{ab}$
- falls $\mathbb{G}_1 = \mathbb{G}_2$ dann gilt zusätzlich $e(S, T) = e(T, S)$

Grundsätzlich unterscheidet man drei verschiedenen Typen von Pairings [KU16]:

1. $\mathbb{G}_1 = \mathbb{G}_2 : \mathbb{G}_1$ muss aus Elementen einer supersingulären elliptischen Kurve bestehen
2. $\mathbb{G}_1 \neq \mathbb{G}_2$ und es gibt einen effizient berechenbaren Homomorphismus $\phi : \mathbb{G}_1 \to \mathbb{G}_2$: Elemente aus $\mathbb{G}_2$ können nicht effizient gehasht werden.
3. $\mathbb{G}_1 \neq \mathbb{G}_2$ und es gibt keinen effizient berechenbaren Homomorphismus $\phi : \mathbb{G}_1 \to \mathbb{G}_2$: Effizientes Hashing ist möglich.

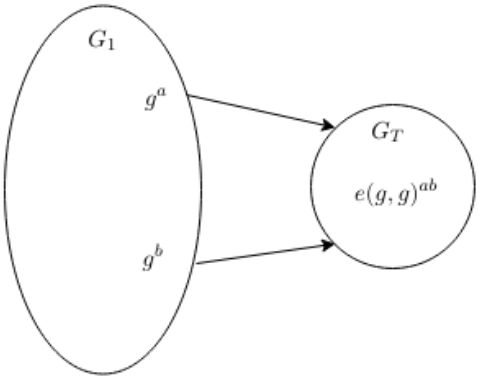

Abbildung 2.4: Type-1 Pairing

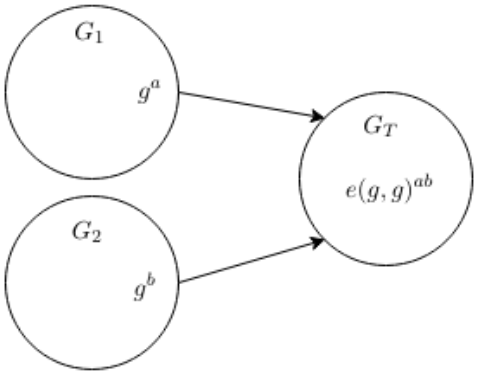

Abbildung 2.5: Type-2/3 Pairing

Bei Pairings vom Typ-1 ($\mathbb{G}_1 = \mathbb{G}_2$) spricht man auch von symmetrischen, und bei Typ- 2/3 ($\mathbb{G}_1 \neq \mathbb{G}_2$) von asymmetrischen Pairings. In der modernen Kryptographie werden meist keine Pairings vom Typ-1 verwendet, da diese auf supersingulären elliptischen Kurven basieren. Hashverfahren auf supersingulären Kurven lassen sich kaum optimieren und verursachen hohen Rechenaufwand. Pairings vom Typ-1 werden in Abbildung 2.3.4, und Typ-2/3 in Abbildung 2.3.4 veranschaulicht.

2.4 Identitätsbasierte Signaturen

Damit die Nachrichtenverifikation mittels identitätsbasierter Signaturen überhaupt möglich ist, müssen alle Kommunikationspartner in einer Gruppe organisiert sein. Hierbei wird jeder geheime Schlüssel von einem zentralen Geheimnis, dem *master secret key (msk)*, abgeleitet. Jeder Teilnehmer besitzt ein Schlüsselpaar, das aus einem geheimen Schlüssel *secret key (sk)* und einem öffentlichen Schlüssel *public key (pk)* besteht, sowie eine eindeutige Identität *(ID)*. In der realen Welt ist die Identität der eigene Name. In IBS kann für die Identität ein beliebiges Wort, das den Teilnehmer eindeutig identifiziert, wie z.B. eine E-Mail-Adresse oder MAC-Adresse, gewählt werden. Der Vorteil von identitätsbasierten Signaturen (IBS) gegenüber herkömmlichen Verfahren, bei denen Zertifikate von einer Zertifizierungsstelle (CA) ausgegeben werden, ist die mathematische Falsifizierbarkeit einzelner Zertifikate. Im herkömmlichen Fall haben Zertifikate eine gewisse Gültigkeitsdauer. Während dieser Zeit lässt sich das Zertifikat immer mit dem öffentlichen Schlüssel der CA verifizieren. Wird ein Zertifikat jedoch z.B. wegen eines kompromittierten geheimen Schlüssels zurückgerufen, so muss jeder Gruppenteilnehmer eine *authority revocation list* führen. In dieser sind alle zurückgerufenen Zertifikate aufgelistet, deren Zeitstempel noch gültig sind. In IBS hingegen kann jedes Gruppenmitglied sofort die Gültigkeit eines Zertifikats mathematisch validieren. Als Adi Shamir 1984 [Sha84] das erste IBS entwickelte, verwendete er schlicht die *ID* als *pk*. Um Overhead bei der Verifikation zu sparen, lässt sich auch in modernen Verfahren der *pk* rechnerisch aus der *ID* eines Teilnehmers ermitteln. Da bei einem IBS-Schema eine Abhängigkeit der privaten Schlüssel aller Gruppenmitglieder bestehen muss, wird eine vertrauenswürdige dritte Partei benötigt, das KGC (*key generation center*). Das KGC besitzt einen initialen geheimen Schlüssel *msk*, aus dem alle anderen privaten *secret keys* der Gruppenmitglieder berechnet werden. Daher ist die Vertrauenswürdigkeit des KGC unabdingbar, weshalb es auch als TTP (*trusted third party*) bezeichnet wird. Die TTP stellt einen *single point of failure* dar, denn falls es einem Angreifer gelingt, diese zu kompromittieren, so ist er in der Lage Nachrichten im Namen jedes einzelnen Gruppenmitgliedes zu signieren. Jedes IBS Schema sollte folgende Funktionalitäten aufweisen [GS02]:

Root Setup: Die TTP wählt einen Sicherheitsparameter K und generiert die Systemparameter *msk* und *master public key (mpk)*.

Lower-level Setup: Gruppenmitglieder dürfen keine eigenen Sicherheitsparameter generieren, da diese aus K des KGC errechnet werden.

Schlüsselgeneration (Extract): Das KGC berechnet mit Kenntnis der *ID* des neuen Gruppenmitgliedes dessen *user secret key (usk)*. Es ist unbedingt erforderlich, dass dieser über einen gesicherten Kanal zwischen KGC und Empfänger übertragen wird.

Signierung: Der Sender berechnet die Signatur (*sig*) aus der Nachricht und seinem *usk*.

Verifikation: Der Enempfänger entschlüsselt die Signatur mit dem *upk* und der *ID* des Senders und ermittelt, wie in Tabelle 2.1 beschrieben, die Validität der Signatur.

Rekey: Falls ein Gruppenmitglied wieder aus der Gruppe ausgeschlossen werden soll, muss dessen KGC einen neuen *sk* und *pk* generieren.

2.4.1 Hierarchische identitätsbasierte Signaturen

Bei zentralisierten IBS generiert die TTP für jedes Gruppenmitglied einen *usk*, baut einen sicheren (z.B. verschlüsselten) Kanal zwischen TTP und neuem Gruppenmitglied auf, und überträgt dessen *upk*. Da die Generierung des *user secret keys* und der Aufbau des sicheren Kanals mit erheblichem Rechenaufwand verbunden ist, kann die TTP in größeren Netzwerken schnell zum Flaschenhals werden. Ferner muss beim Ausschluss eines Gruppenmitglieds dessen KGC einen *Rekey* durchführen. In zentralisierten IBS bedeutet dies, dass alle Gruppenmitglieder, inklusive der TTP, einen *Rekey* durchführen müssen. Dies ist zum einen eine enorme Belastung der TTP, zum anderen muss auch jedes Gruppenmitglied eine Verbindung mit der TTP aufbauen. Im IoT ist Energie kostbar, da jede zusätzliche Funkkommunikation und jeder Rechenaufwand, ein schnelleres Ableben der Batterie bewirkt.

An dieser Stelle kommen hierarchische identitätsbasierte Signaturen (HIBS) ins Spiel. In HIBS bekleidet nicht nur die TTP die Rolle des KGC, sondern die Last der Schlüsselgeneration wird auf mehrere Geräte verteilt. Hier bietet sich die Organisation der Gruppe als Baumstruktur mit der TTP als Wurzel besonders an. Im Grunde kann jede Baumstruktur verwendet werden, solange nur eine Wurzel existiert. Alle Knoten im Baum können nun als KGC fungieren. Das heißt die TTP generiert die *user secret keys* nur für ihre direkten Kinder. Jedes Kind kann nun wieder als Vaterknoten fungieren und einen *usk* für weitere Knoten generieren, wobei der *usk* in Rang $n + 1$ von dem *usk* in Rang n abgeleitet ist. So kann im einfachsten Beispiel ein k-Baum verwendet werden, bei dem jeder Vaterknoten maximal $k = 2$ Kinder besitzt (Vgl. Abbildung 2.6). Die ersten beiden Gruppenmitglieder verwenden nun die TTP als KGC. Falls ein drittes Gerät in die Baumstruktur eintritt, wird der erste Knoten zu einem KGC und generiert den *usk* des dritten Knotens.

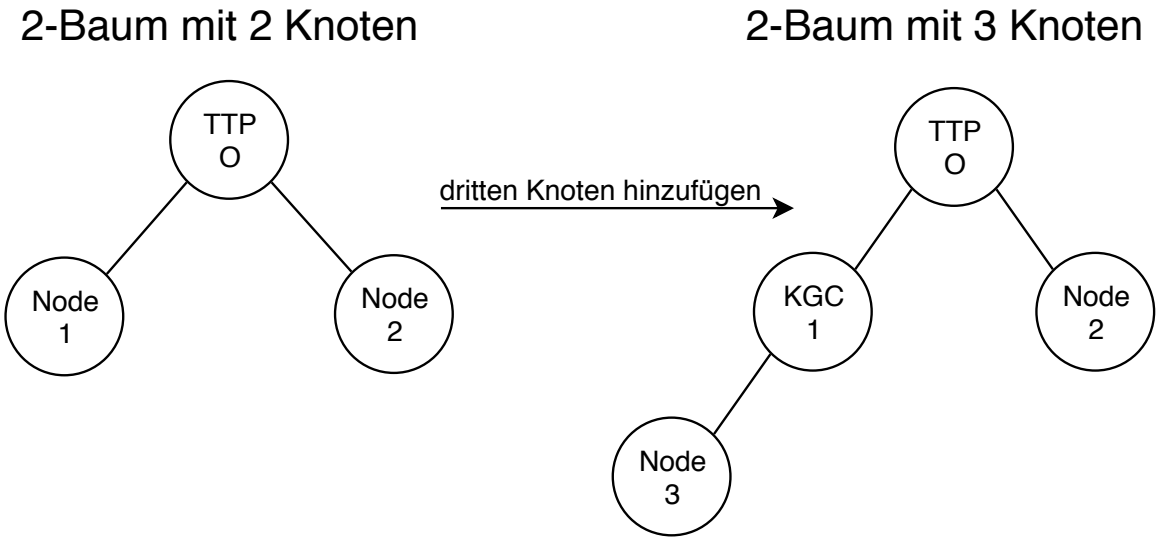

Abbildung 2.6: Gruppeneintritt in einen 2-Baum

Die Entlastung der TTP ist aber nicht der einzige Vorteil eines HIBS. Falls ein Gruppenmitglied aus der Baumstruktur entfernt wird, müssen nun nicht mehr alle Geräte einen *Rekey* vornehmen. Bei HIBS führt nur noch der direkte Vaterkonten und alle rekursiven

Kinder des Vaterknotens einen *Rekey* durch. Falls im vorherigen Beispiel Knoten drei aus dem Baum entfernt werden soll, muss nur noch Knoten eins einen *Rekey* vornehmen. In größeren Baumstrukturen ist es sinnvoll, die Anzahl der beim *Rekey* betroffenen Knoten zu minimieren. Dafür ist es vorteilhaft ständige Mitglieder nahe der Wurzel anzusiedeln, und häufig wechselnde an den Blättern. Außerdem ist es sinnvoll, logische Baumstrukturen zu verwenden. Das heißt alle inneren Knoten, sowie die Wurzel sind auf der TTP emuliert. Dadurch wird die TTP zwar nicht entlastet, aber bei einem Gruppenaustritt müssen nicht mehr alle Knoten mit neuen Schlüsseln ausgestattet werden. Die Vor- und Nachteile von logischen Bäumen werden in Kapitel 3.3 genauer diskutiert.

Die Verwendung einer Hierarchie bringt allerdings nicht nur Vorteile mit sich. Jeder Knoten im Baum kann in HIBS als KGC auftreten, weshalb die *user public keys* nicht mehr von der *ID* abgeleitet werden können. Das führt zu erhöhtem Speicherverbrauch, denn jeder Knoten muss nun alle *user public keys* der inneren Knoten, sowie den *ID*-Pfad von seiner Position im Baum bis zur Wurzel kennen.

2.4.2 GS02 - Hierarchisches identitätsbasiertes Signatur Schema

Es gibt zahlreiche Veröffentlichungen, in denen verschiedene Ansätze zur Implementierung eines HIBS präsentiert werden. Diese Bachelorarbeit beschäftitgt sich mit der effizienten Implementierung eines HIBS für Gruppenkommunikation von leistungsschwachen Geräten, weshalb ein besonderes Augenmerk auf die Effizienz beim *Rekey* gelegt wurde. Das HIBS aus der Abhandlung „Hierarchical ID-Based Cryptography“ [GS02] von Alice Silverberg und Craig Gentry hat einige Vorteile gegenüber vielen anderen Verfahren. Die maximale Anzahl der Levels im HIBS muss noch nicht im Setup festgelegt sein. Werden Gruppenmitglieder in neue Levels hinzugefügt, wächst nur die Größe der Signatur und die Anzahl der gespeicherten *public keys*. Des Weiteren müssen bei einem *Rekey* nur der Vaterknoten und dessen rekursive Kinder mit neuen *usks* ausgestattet werden. Silverberg und Gentry leisteten Pionierarbeit, denn ihre Arbeit ist eine der ersten Publikationen, in der ein konkretes HIBS vorgestellt wurde. Wie in den meisten Veröffentlichungen, sind nur die vier Phasen *Setup, Extract, Sign und Verify* beschrieben, da der *Rekey* abhängig von der zugrundeliegenden Baumstruktur ist. In diesem Verfahren kann die TTP keine Nachrichten signieren, sondern ist ausschließlich für die Schlüsselgeneration zuständig. Tabelle 2.3 veranschaulicht die Rechenoperationen in den einzelnen Phasen, die von Silverberg und Gentry definiert wurden. Der *secret key* setzt sich in aus einem geheimen Kurvenpunkt S_t und einer geheimen Zufallszahl *usk* bzw. *msk* zusammen. Er wird als pk_t wird als Q_t betitelt, wobei t immer das aktuelle Level ist.

Phase	Mathematische Berechnung	Beschreibung
Root Setup	$msk_0 \leftarrow \mathbb{Z}_p$	msk_0 ist eine Zufallszahl aus $\mathbb{Z}_p$
	$S_0 = \mathbb{1}$	S_0 ist das Identitätselement
	$Q_0 = msk_0 * P_0$	$Q_0, P_0 \in \mathbb{G}_1$ wobei $\langle P_0 \rangle = \mathbb{G}_1$
Lower-Level Setup *(level t)*	$usk_t \leftarrow \mathbb{Z}_p$	usk_t ist eine Zufallszahl aus $\mathbb{Z}_p$
	$Q_t = usk_0 * P_t$	$Q_t, P_0 \in \mathbb{G}_1$ wobei $\langle P_0 \rangle = \mathbb{G}_1$
Extract	$P_t = H_1(ID_1, ..., ID_t)$	Hashe ID-Pfad ohne Wurzel auf Kurvenpunkt $P_t \in \mathbb{G}_1$
	$S_t = S_{t-1} \oplus msk_{t-1} * P_t$	Geheimer Kurvenpunkt $S - t$ des neuen Gruppenmitgliedes
	$S, Q_1, ..., Q_{t-1}$	Sende S und alle $Q_0...t$
Sign	$P_M = H_3(ID_1, ..., ID_t, M)$	Hashe die ID-Ast mit Nachricht
	$S = S_t \oplus msk_t * P_M$	Multiplikation geheimer Punkt mit dem Hash
	$Sig = (M, S, Q_1, ...Q_t)$	Versende $Nachricht$, $Signatur$ und Q-Ast
Verify	$a = e(Q_0, H_1(ID_1))$	Berechne Pairings a, b, c aus Sig
	$b = e(Q_t, H_3(ID_1, ..., ID_t, M))$	t ist das Level des Senders
	$c = \prod_{i=2}^{t} e(Q_{i-1}, H_1(ID_1, ..., ID_t))$	
	$e(P_0, S) \stackrel{?}{=} a * b * c$	Vergleiche Signatur mit Pairings

Tabelle 2.3: GS02 Verfahren [GC17]

2.5 Related Work

Adi Shamir legte 1984 in seiner richtungsweisenden Arbeit „*Identity-Based Cryptosystems and Signature Schemes*“ [Sha84] den Grundstein für identitätsbasierte Signaturverfahren. Sophia Grundner-Culemann evaluierte in ihrer Masterarbeit [GC17] sechs bekannte IBS anhand von Sicherheit, Schlüsselgrößen und Rechenaufwands. In der Abhandlung [FGCG18] wird die Authentifizierung und Gruppenkommunikation, anhand von identitätsbasierten Signaturen für eingeschränkte Geräte auf verschiedener Hardware im IoT-Lab evaluiert. Andrian Melnikov implementierte in seiner Bachelorarbeit [Mel18] das zentralisierte IBS BLMQ [BLM05], sowie ein Testbed, mit dem unterschiedliche IBS evaluiert werden können. GS02 ist natürlich nicht das einzige HIBS-Schema, es gibt neben zufallszahlbasierten Ansätzen wie GS02 oder [YD08] auch zufallszahlenunabhängige Verfahren wie „Practical Hierarchical Identity Based Encryption and Signature schemes Without Random Oracles“ von Au, Liu, Yuen und Wong [ALYW06]. Neben IBS existieren viele weitere Signaturverfahren, die auf ECC beruhen, wie *Attribut-Based Signatures*[SSN09], [MPR11] bei denen die Identifikation nicht nur anhand einer *ID* erfolgt, sondern von mehreren Attributen abhängt. Auch beim Pendant zu HIBS aus der Verschlüsselung, die *Hierarchical Identity Based Encryption* kommt ECC zum Einsatz. Ein Beispiel hierfür ist die Veröffentlichung von Alice Silverberg und Craig Gentry „Hierarchical ID-Based Cryptography“ [GS02], das neben dem in dieser Bachelorarbeit verwendete HIBS-Schema, auch zwei *Hierarchical-Identity-Based-Encryption*-Schemas definiert.

3 Entwurf eines HIBS-Protokolls

Dieses Kapitel beschreibt die Anforderungen an ein HIBS-Protokoll und zwei verschiedene Designmöglichkeiten. Ein besonderes Augenmerk liegt auf der Baumstruktur, die dem Gruppenmanagement zugrunde liegt. Diese ist ausschlaggebend für effiziente Rechenlastverteilung und Speicherverbrauch in den einzelnen Phasen (Initialisierung, Gruppeneintritt, Kommunikation, Gruppenaustritt).

3.1 Anforderungsanalyse

In diesem Abschnitt werden funktionale-, nichtfunktionale- sowie Sicherheitsanforderungen, die ein HIBS erfüllen sollte, beschrieben.

3.1.1 Sicherheitsanforderungen

Die essentiellsten Anforderungen an ein kryptographisches System sind die Sicherheitsanforderungen. Dabei ist darauf zu achten, dass die Gesamtsicherheit nur so groß ist wie die Sicherheit des schwächsten Glieds in der Kette (Betriebssystem, Kryptografiebibliothek, Protokoll).

Betriebssystem: Um geheime Schlüssel sicher unter Verschluss zu halten, darf das Betriebssystem keinem anderen Thread den Zugriff auf Speicherbereiche des Protokolls gewähren.

Kryptografiebibliothek: Die Wahl einer passenden Kryptografiebibliothek für ECC mit einem sicheren Zufallszahlengenerator und elliptischen Kurven mit ausreichender Schlüssellänge (Das BSI empfiehlt 250 Bit für ECC [Inf18]).

Hierarchische Schlüsselunabhängigkeit: Ein Knoten darf nur in der Lage sein, Schlüssel für seine Kinder generieren zu können. Andere Schlüssel dürfen nicht berechenbar sein.

Datenauthentizität, Integrität und Verbindlichkeit: Jedes Gruppenmitglied muss in der Lage sein, die Echtheit der Nachricht und deren Verfasser nachzuweisen, Veränderungen der Nachricht festzustellen, sowie die Urheberschaft der Nachricht nicht bestreiten zu können.

Vertrauenswürdige TTP: Jedes Gruppenmitglied muss die TTP (ggf. auch mehrere) als vertrauenswürdig erachten.

Eindeutige ID: Alle Gruppenmitglieder müssen über eine eindeutige *ID* verfügen, mit der sie sich identifiziert lassen.

3.1.2 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen beschreiben die Nutzbarkeit, Zuverlässigkeit und Effizienz der Anwendung.

Zuverlässigkeit: Das System soll auch nach beliebig vielen Gruppenein- und austritten stabil laufen. Fehleingaben sollen die Stabilität nicht gefährden und abgefangen werden.

Skalierbarkeit: Das Protokoll soll auch große Gruppen unterstützen. Die maximale Clientanzahl kann jedoch im Vorhinein festgelegt werden.

Ressourceneffizienz: Aufgrund der beschränkten Hardwareausstattung von Mikrocontrollern sollen Rechenaufwand und Arbeitsspeicherverbrauch möglichst gering gehalten werden.

3.1.3 Funktionale Anforderungen

Funktionale Anforderungen spezifizieren die einzelnen Funktionalitäten der Implementierung. Es muss ein HIBS Protokoll für einen Server (TTP), der das Gruppenmanagement übernimmt, und für Clients implementiert werden. Clients treten je nach Implementierung selbst als KGC auf, oder beschränken sich nur auf Gruppenkommunikation. Die übertragenen Nachrichten sollen durch das Protokoll signiert werden, sodass Authentizität, Integrität und Verbindlichkeit gewährleistet sind. Die Auswahl der Elliptischen Kurve, sowie die Eigenschaften der Baumstruktur sollen wählbar sein. Die Kommunikation zwischen Knoten und TTP soll über verschiedene Ports erfolgen.

3.2 Rollenbasierte Baumstruktur

In diesem Schlüsselbaum übernimmt jeder Knoten die Rolle als *key generation center*, weshalb diese Baumstruktur in dieser Arbeit als „rollenbasiert“ bezeichnet wird. Dadurch verteilt sich die Rechenlast bei häufigen Gruppenein- und austritten, von der TTP auf mehrere Knoten. Neue Knoten werden immer an der ersten freien Stelle im Baum eingefügt, sodass der Baum in die Tiefe wächst. Dank dieser Struktur benötigt die TTP nur äquivalente Hardwareausstattung wie die Knoten. Da sich im GS02 Verfahren der *upk* nicht aus der *ID* berechnen lässt, geht mit dieser Baumstruktur leider der Nachteil einher, dass jeder Client alle *public keys* der inneren Knoten kennen muss. Je tiefer der Baum wird, desto mehr Speicherplatz wird zum Speichern der *public keys* benötigt. Da die TTP in diesem Verfahren keine Nachrichten signieren kann, wurde in der Implementierung eine abgewandelte Form eines K-Baums mit der TTP als Wurzel gewählt. In dieser ist die TTP über der kommunizierenden Gruppe angesiedelt, und besitzt nur ein direktes Kind. Daher kann die TTP als „außenstehende Partei“ betrachtet werden, die mehrere HIBS Gruppen verwaltet. Jeder andere Knoten im Baum besitzt maximal k Kinder. In Abbildung 3.1 wird eine K-Baumstruktur mit $k = 2$ (Binärbaum) dargestellt.

Initialisierung

Die Initialisierung der TTP muss stets vor dem Gruppeneintritt der Knoten abgeschlossen sein. Sie berechnet mit Hilfe des GS02 Verfahrens 2.3 den msk sowie den mpk. Des Weiteren erstellt sie einen leeren IP-Baum und geht in den wartenden Zustand „Message Handler“

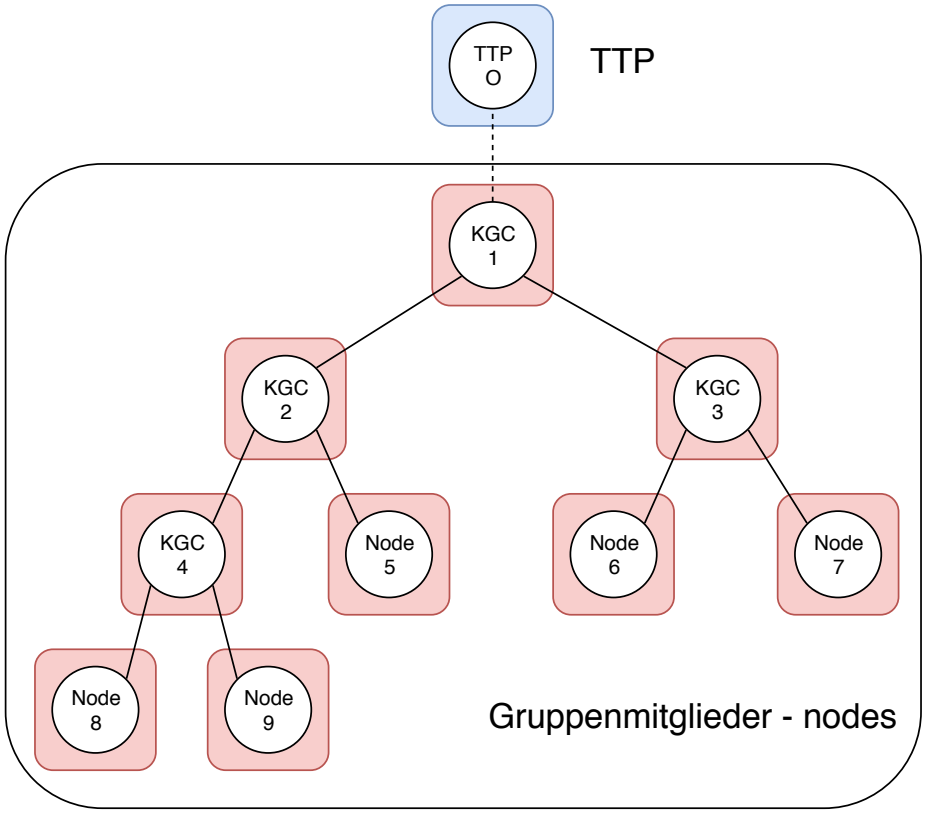

Abbildung 3.1: Rollenbasierte Baumstruktur

über. Jeder Client berechnen nur die geheime Zufallszahl *usk* und seinen *upk* und wartet anschließend auf die Benutzerinteraktion zum Gruppeneintritt (join).

Gruppeneintritt

Der Gruppeneintritt wird immer vom Client nach folgendem Schema initiiert:

1. Der Client sendet eine Anfrage nach seinem persönlichen Vater (KGC) an die TTP.
2. Die TTP fügt den Client an der nächsten freien Stelle im Baum ein, berechnet dessen Vater und antwortet mit dessen IP-Adresse und der Baum-ID (Diese ID hat nichts mit der eindeutigen persönlichen *ID* zu tun, sondern beschreibt lediglich die Position des Clients im Baum).
3. Der Client sendet einen *Extract Request* und seine eindeutige *ID* an seinen Vaterknoten. Dieser berechnet den geheimen Kurvenpunkt und schickt diesen über einen sicheren Kanal an den Client zurück. Des Weiteren teilt er dem Client alle inneren *user public keys* mit.
4. Im letzten Schritt des Gruppeneintritts teilt der Client allen anderen Gruppenmitgliedern seinen *public key* mit und geht in den wartenden Zustand über.

Somit ist er der Gruppe komplett beigetreten und kann als KGC fungieren sowie Nachrichten Signieren und Verifizieren.

Kommunikation

Der Client befindet sich im „Message Handler Modus“ und wartet auf ankommende signierte Nachrichten und *Extract Requests*, oder auf Kommandozeileneingaben zum versenden einer signierten Nachricht.

- ***Extract Request:*** In einer rollenbasierten Baumstruktur werden TTP und Clients KGC genannt, da beide den *Extract* für neue Gruppenmitglieder ausführen. Empfängt ein KGC einen *Extract Request*, berechnet es mithilfe eines eigenen geheimen Kurvenpunkt S_t und der vom Client mitgesandten eindeutigen *ID*, dessen Kurvenpunkt S_{t+1}. Anschließend wird dieser zusammen mit dem eigenen *Public-Key-Ast* über einen gesicherten Kanal zurückgesandt.
- **Nachrichtenempfang:** Die Nachricht wird anhand der mitgesandten Signatur, dem *ID*-Pfad von der Wurzel bis zum Sender und der ID im Baum folgendermaßen verifiziert:
  1. Der *Public Key* Pfad von der Wurzel bis zum Absender wird anhand seiner Baum-ID berechnet.
  2. Die Signatur wird nun, wie in Tabelle 2.4.2 beschrieben, anhand des berechneten *Public Key* Pfades, des mitgesandten *ID*-Pfades und der Signatur verifiziert.
  3. Das Programm teilt dem Benutzer die Validität der Signatur mit.
- **Nachrichtenversand:** Eine vom Nutzer eingegebene Nachricht wird durch den eigenen geheimen Kurvenpunkt, sowie der geheimen Zufallszahl nach Tabelle 2.4.2 signiert. Diese Nachricht kann nun inklusive Signatur, öffentlichen Schlüssel und eigenem *ID*-Ast beliebig via Unicast oder Multicast an Gruppenteilnehmer versandt werden.

Gruppenaustritt

Tritt ein Client aus der Gruppe aus, so wird als erstes der IP-Baum der TTP aktualisiert. Denn falls der entfernte Knoten ein KGC war, hängen nun seine Kinder in der Luft. Dies kann entweder durch Nachrücken des tiefsten Teilbaumes oder durch Verschieben eines Blattes behoben werden. Danach weißt die TTP den Vaterknoten des ausgeschlossenen Knotens an, einen *Rekey* durchzuführen. Dieser berechnet eine neue Zufallszahl und führt die Prozedur des Gruppeneintritts (Abschnitt 3.2) erneut durch. Anschließend teilt die TTP allen anderen Clients die Ungültigkeit des entfernten *public keys* mit. Falls der entfernte Knoten ein rekursiver Vaterknoten eines Gruppenmitgliedes war, muss dieses ebenso einen *Rekey* durchführen.

3.3 Logischer Schlüsselbaum

Bei der Gruppenorganisation in einem logischen Schlüsselbaum sind alle Clients auf der untersten Baumebene angesiedelt und daher Blätter. Clients fungieren nun nicht mehr als KGC und müssen deshalb nicht mehr vertrauenswürdig sein. Alle inneren Knoten liegen als logische Knoten auf der TPP. Das bringt den Vorteil mit sich, dass in dieser Baumstruktur ein Client nicht alle *public keys* kennen muss, da die TTP ohnehin vertrauenswürdig ist. Es reicht wenn die Clients alle öffentlichen Schlüssel der untersten logischen Ebene der TTP

kennen. Im Beispiel (Abbildung 3.3) sind das die Schlüssel aus Rang zwei. Die TTP kann nun direkt an der Gruppenkommunikation teilnehmen, indem sie Nachrichten anhand der *ID* eines logischen Knotens signieren kann. Daher steht sie nicht mehr über der Baumstruktur. Logische Knoten ändern sich nicht unerwartet, sodass diese Baumstruktur eine weitere Optimierungsmöglichkeit bietet. Die TTP kann schon einen Teil der Pairings vorberechnen, die in der Verifizierung von Nachrichten benötigt werden, und diese an die Clients verteilen. Da bei der Verifizierung immer das Pairing von dem *pubic key* aus Rang i mit dem gehashten *ID*-Pfad dessen Kindes aus Rang $i + 1$ berechnet wird, können im Beispielbaum (Abbildung 3.3) zwei Pairings pro logischen Ast vorberechnet werden. Bei diesem Verfahren muss die TTP über eine bessere Hardwareausstattung verfügen, da die gesamte Speicher- und Rechenlast bei der Schlüsselgeneration von ihr getragen wird.

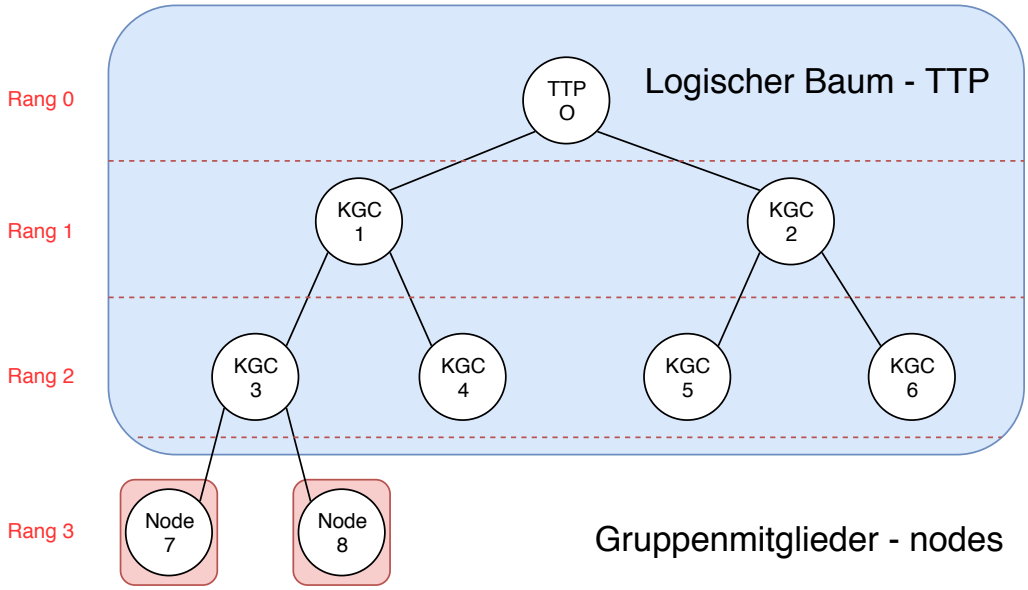

Abbildung 3.2: Logischer Baum

Initialisierung

In diesem Protokoll muss die Tiefe des Schlüsselbaumes im Vorhinein festgelegt werden. Danach berechnet die TTP analog zur Initialisierung in Kapitel 3.2 ihren *msk* und *mpk*. Anschließend berechnet und speichert sie alle *user secret keys* und *user public keys* für alle Knoten in Rang eins, wie unter Gruppeneintritt beschrieben. Dieses Prozedere wird rekursiv für jede logische Ebene wiederholt, bis alle logische Knoten über ein Schlüsselpaar verfügen. Nun bildet und speichert die TTP Pairings aus *pbk*-Pfad und *ID*-Pfad von der Wurzel bis zu jedem Knoten der untersten logischen Ebene (im Beispiel Rang 2). Im letzten Schritt wird ein leerer IP-Baum gebildet und in den „Message Handler“ übergegangen.

Gruppeneintritt

Beim Gruppeneintritt im logischen Verfahren ist weniger Nachrichtenaustausch vonnöten, da der Vater jedes neuen Gruppenmitgliedes immer ein logischer Knoten auf der TTP ist. Im Gegensatz zum ersten Verfahren, welches neue Knoten immer in der Tiefe ansiedelt, wird

hier der Baum in der Breite erweitert. D.h. alle Gruppenmitglieder sind auf dem gleichen Rang und haben Väter aus der letzten logischen Schicht der TTP. Der Gruppeneintritt (join) besteht nur aus einem einzigen Schritt:

- Der Client sendet einen *Extract Request* mit seiner eindeutigen *ID* an die TTP. Diese hängt ihn an einen Ast, fügt dessen IP in den IP-Baum ein und extrahiert den geheimen Kurvenpunkt. Dieser wird über einen gesicherten Kanal zum Client zurrückgesandt. Alle vorberechneten Pairings und die öffentlichen Schlüssel der untersten logischen Ebene werden ebenfalls an das neue Gruppenmitglied übermittelt.

Nach erfolgreichem Gruppenbeitritt geht der Client in den wartenden Zustand „Message Handler“.

Kommunikation

Der Client befindet sich im „Message Handler“ und wartet auf ankommende signierte Nachrichten oder auf Kommandozeileneingaben zum versenden einer signierten Nachricht.

- ***Extract Request:*** In einer logischen Baumstruktur darf nur die TTP als KGC auftreten. Empfängt ein Client einen *Extract Request* so verwirft er ihn einfach oder antwortet dem Absender mit der IP der TTP. Ist der *Request* an die TTP adressiert, wird wie unter Gruppeneintritt (Abschnitt 3.3) verfahren.
- **Nachrichten Empfang:** Die Nachricht wird anhand der mitgesandten Signatur und *ID*-Pfad folgendermaßen verifiziert:
  1. Der Vater des Absenders wird anhand seiner Baum ID berechnet.
  2. Die Signatur wird mit Hilfe des vorgehashten Baumastes, der *public keys* des Absenders und dessen Vaters, des *ID*-Pfades und der Signatur verifiziert.
  3. Das Programm teilt dem Benutzer die Validität der Signatur mit.
- **Nachrichten Versand:** Eine vom Nutzer eingegebene Nachricht wird durch den eigenen geheimen Kurvenpunkt sowie der geheimen Zufallszahl nach (Tabelle 2.4.2) signiert. Diese Nachricht kann nun inklusive Signatur, öffentlichem Schlüssel und eigenem *ID*-Ast beliebig via Unicast oder Multicast an Gruppenteilnehmer versandt werden.

Gruppenaustritt

Wird ein Client aus der Gruppe ausgeschlossen, so führt dessen logischer Vaterknoten einen *Rekey* durch. Anschließend wird dessen *public key* von der TTP an alle Gruppenmitglieder verteilt und all seine Kinder angewiesen, ebenfalls einen *Rekey* durchzuführen.

3.3.1 Zusammenfassung

Die Wahl der Baumstruktur entscheidet über die Verteilung der Rechnelast, den Speicherverbrauch auf Clients und TTP, sowie der Anzahl der Clients die einen *Rekey* durchführen müssen. Viele Kinder pro Knoten bewirken eine erhöhte Rechenlast der einzelnen Knoten beim *Rekey*, doch mit der Tiefe des Baums steigt auch der Speicherverbrauch und der Rechenaufwand bei der Verifikation auf jedem Knoten. Um zu veranschaulichen welche Daten beim Gruppenein- und austritt, sowie bei der Übermittlung einer signierten Nachricht im

jeweiligen Verfahren übertragen werden, sind diese anhand der Sequenzdiagramme (Abbildung 3.3 und 3.4) gegenübergestellt. Der Funktionsumfang der Protokolle für Clients und der TTP wird in den Zustandsdiagrammen (Abbildung 3.6 bis 3.7) abgebildet.

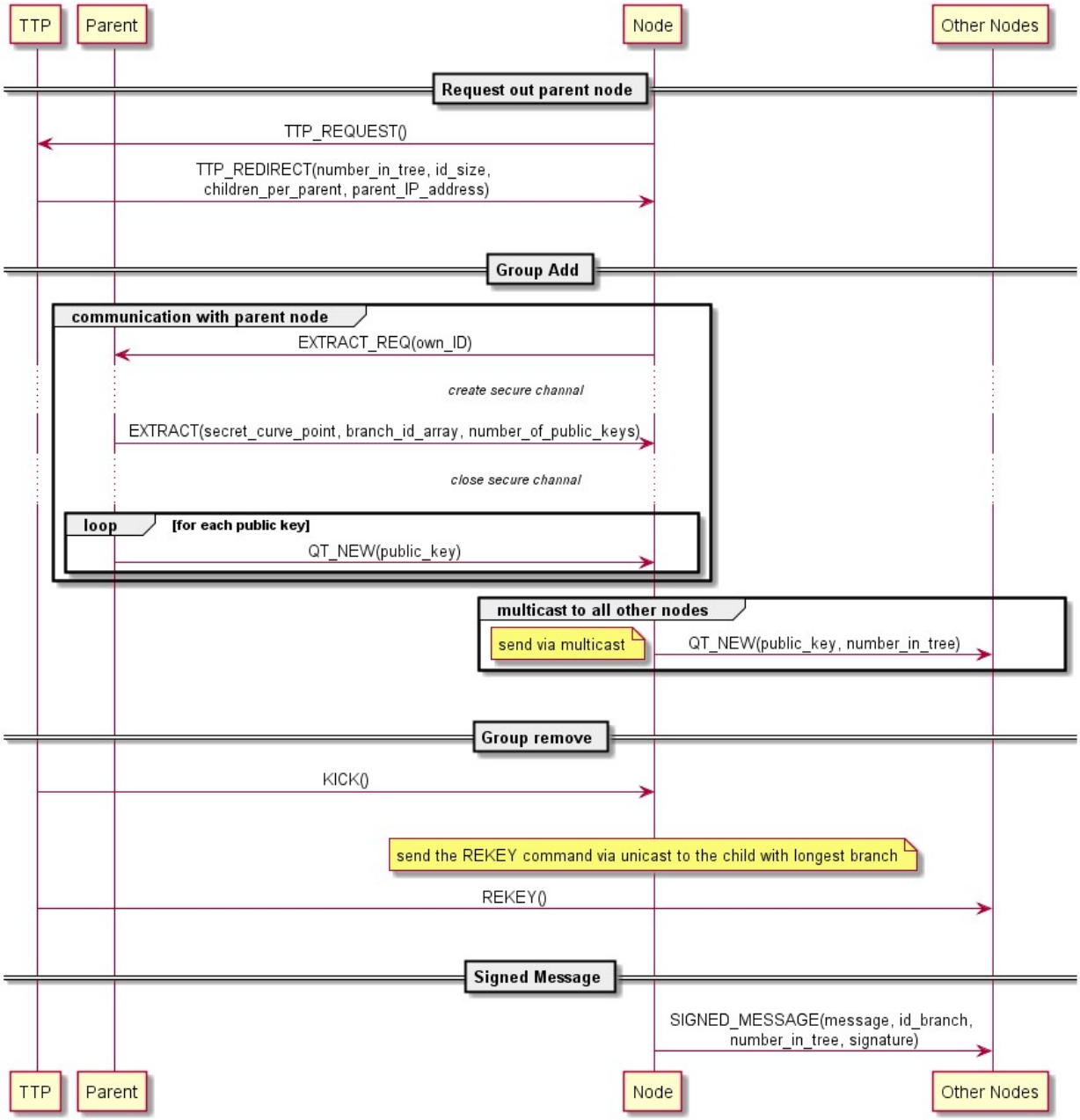

Abbildung 3.3: Sequenzdiagramm rollenbasierte Baumstruktur

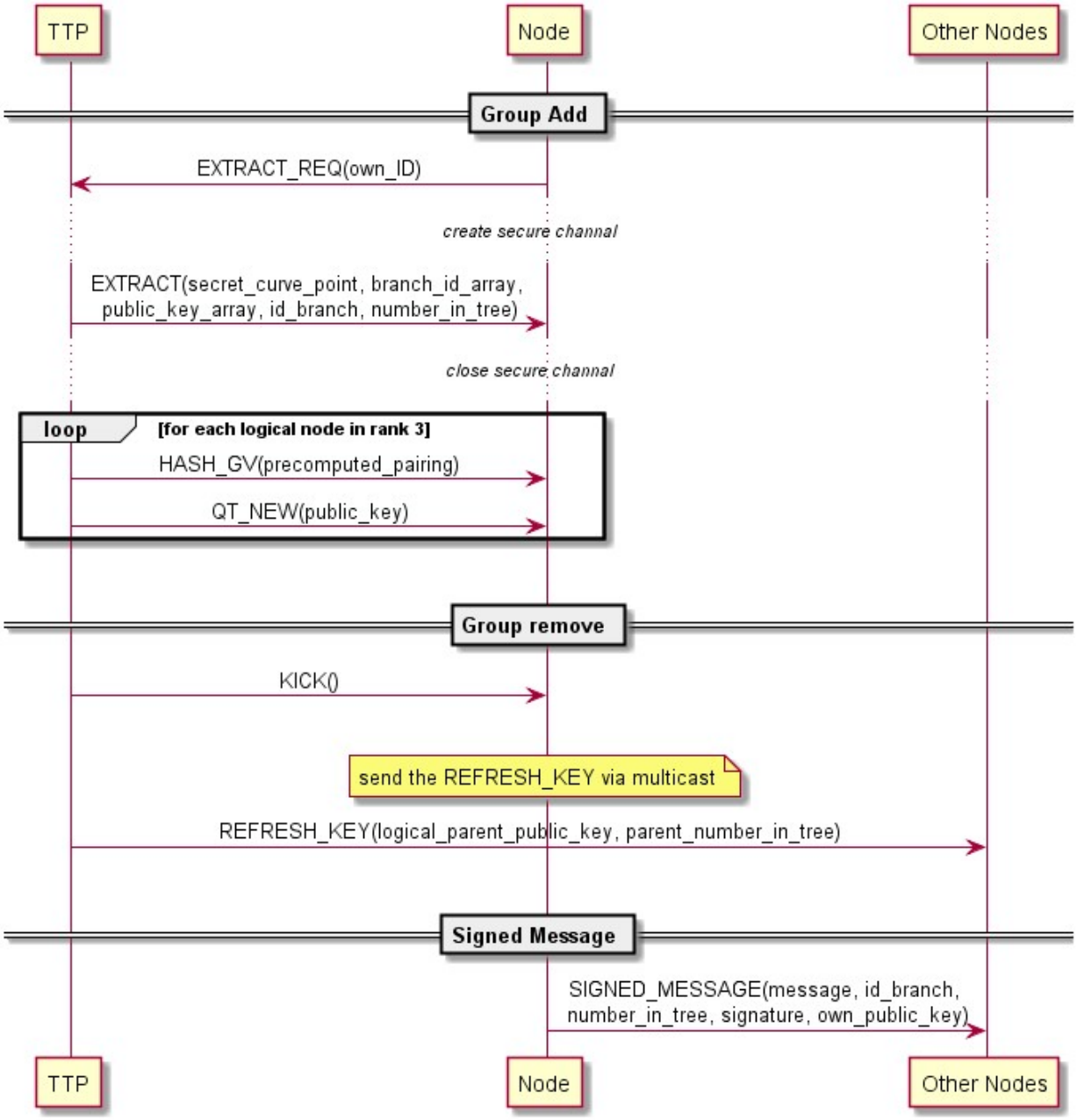

Abbildung 3.4: Sequenzdiagramm logischer Baum

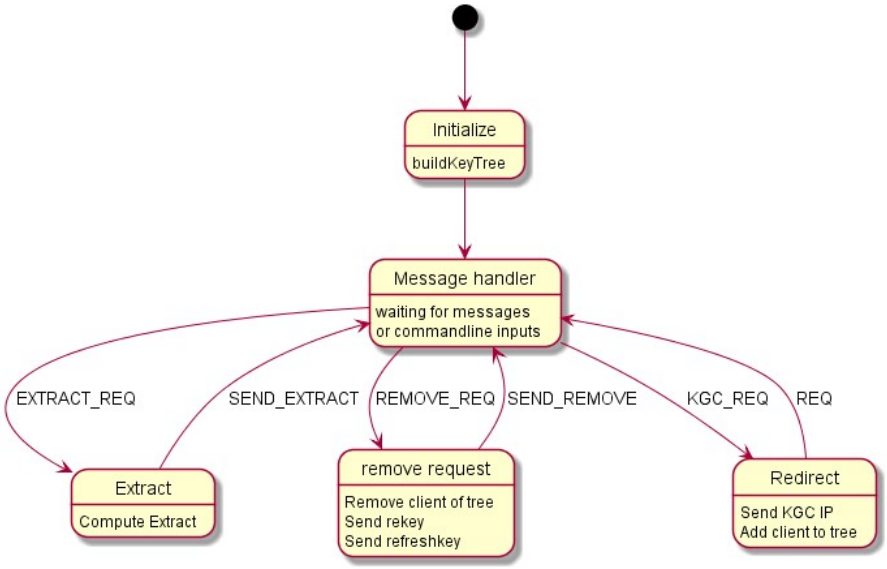

Abbildung 3.5: Zustandsdiagramm rollenbasierte TTP

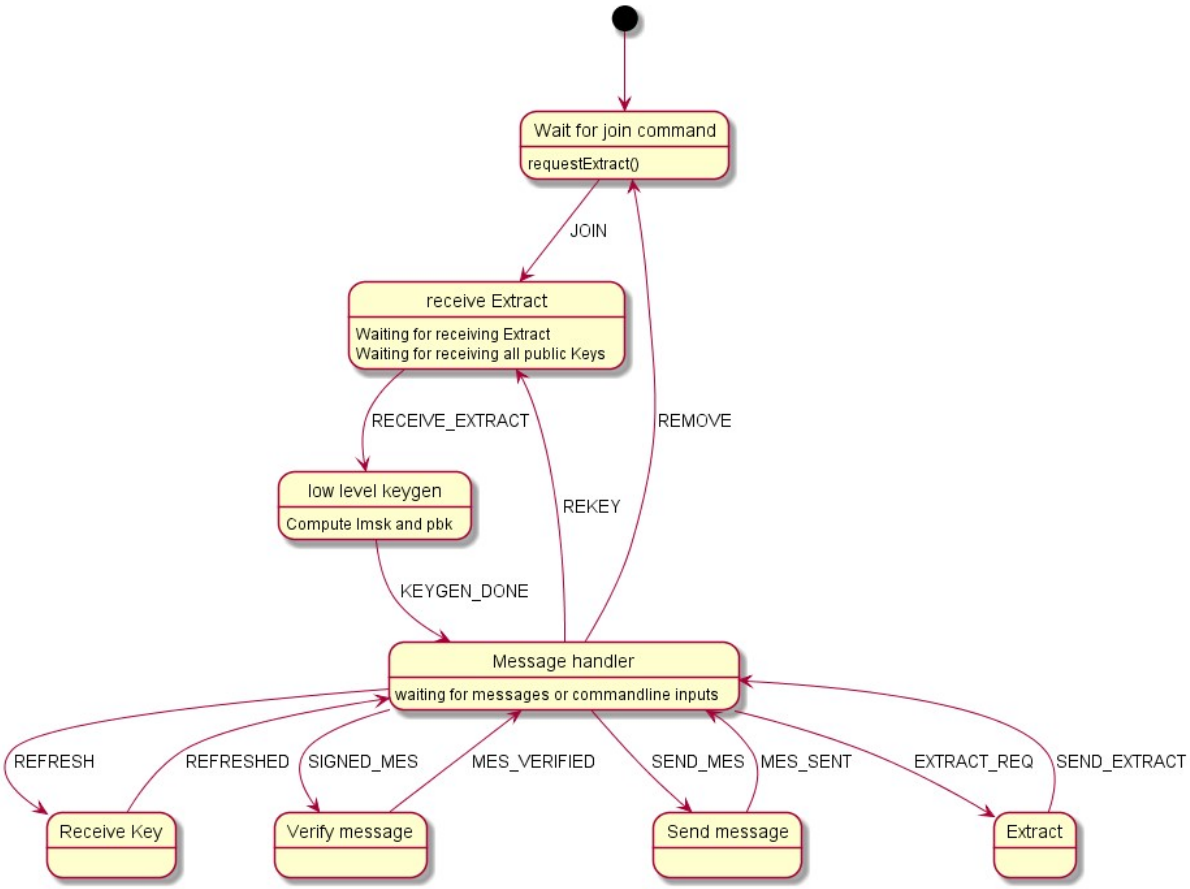

Abbildung 3.6: Zustandsdiagramm rollenbasierter node

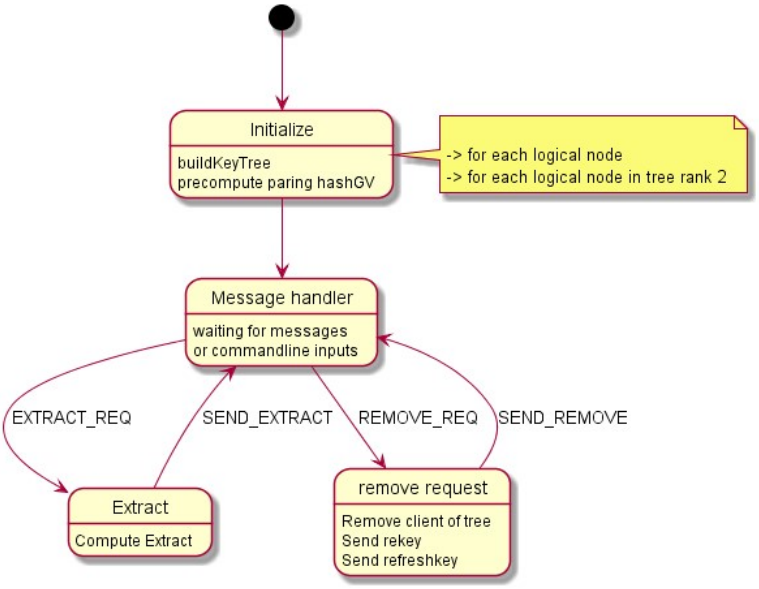

Abbildung 3.7: Zustandsdiagramm logische TTP

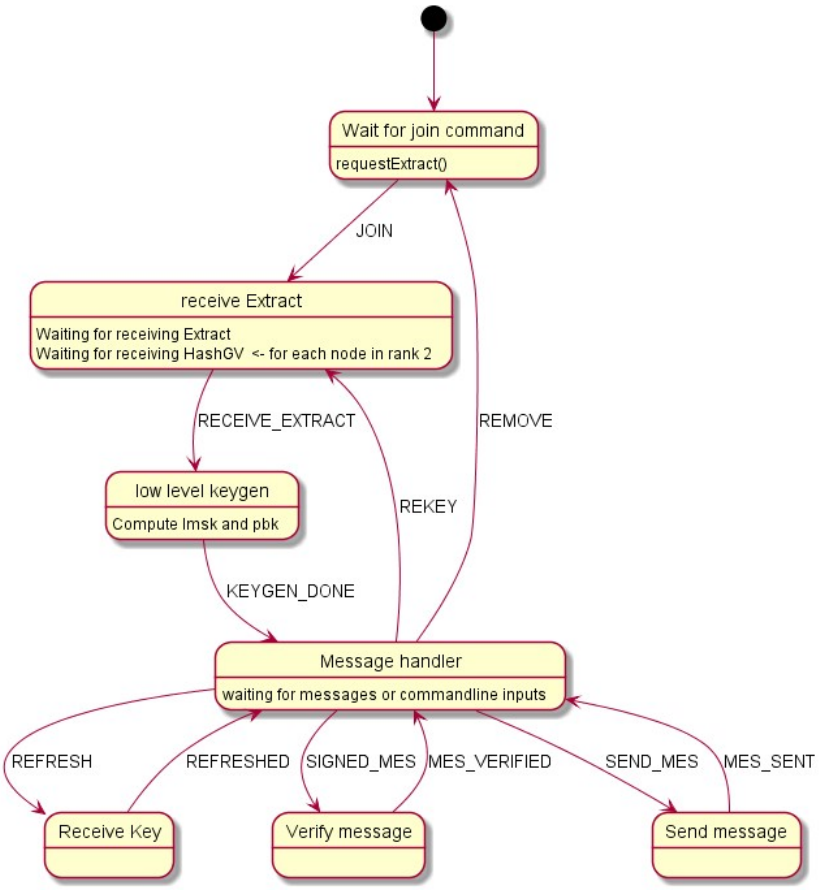

Abbildung 3.8: Zustandsdiagramm logischer node

4 Implementierung

Dieses Kapitel beschäftigt sich erst mit dem, in der Implementierung verwendeten, RIOT Betriebssystem und der Kryptographiebibliothek Relic. Anschließend wird auf allgemeine Konfigurationsmöglichkeiten für beide Baumstrukturen eingegangen. Die Programme wurden nach den Spezifikationen aus dem Kapitel 3 entworfen. Einige Schlüsselstellen der jeweiligen Implementierung werden unter 4.5 bzw. 4.6 näher erläutert.

4.1 RIOT OS

RIOT entstand 2008 aus dem FireWare Projekt der freien Universität Berlin, in dem mithilfe von *Wireless Sensor Neworks* Feuerwehrmänner im Einsatz, überwacht werden sollten. Daher lag bei der Implementierung des Systems das Hauptaugenmerk auf Echtzeitfähigkeit und Zuverlässigkeit. Im laufe der Jahre wurde die Funktionalität stetig erweitert und sämtliche IETF Protokolle wie 6LOWPAN oder TCP für RIOT implementiert, sodass der Einsatz im IoT ermöglicht wurde. Seit 2013 ist RIOT LGPL lizenziert und der Quellcode auf Github veröffentlicht, sodass das Betriebssystem der breiten Allgemeinheit zur Verfügung gestellt wurde.

4.1.1 Softwareanforderungen an RIOT

Auf Grund des großen Einsatzspektrums von IoT Geräten verfügen diese über eine sehr heterogene Hardwareausstattung. Einige eingebettete Systeme überwachen, von einer Batterie gespeißt, nur Sensorwerte und übermitteln diese im Stundentakt über RFID, andere wiederum berechnen komplexe Hashes mit Kryptografiebibliotheken. Daher sind minimale Hardwareanforderungen, leichte Portabilität, Energieeffizienz und eine einfache API die Hauptanforderungen an ein solches Betriebssystem. Betriebssysteme lassen sich auf drei verschiedene Arten implementieren [BHW+12]:

1. **Monolithischer Kernel** Ist die einfachste Art den Kernel zu Implementieren. Aufgrund der mangelnden Modularität wird der Programmcode in größeren Programmen schnell unübersichtlich.

2. **Schichtbasierter Kernel** Der Kernel wird hierarchisch aufgebaut und in Kernel- und Userspace getrennt. Das Betriebssystem wird in viele kleine Funktionen gegliedert, von denen nur eine Hand voll im Kernelmode laufen müssen. Dieser Ansatz bietet den Vorteil im falle einer abstürzenden Funktion kann das Betriebssystem einfach weiter ausgeführt werden.

3. **Mikrokernel Architektur** Entweder es arbeiten alle Programme auf dem Gleichen Speicherbereich (Stack) oder jedes Programm verwaltet seinen eigenen Speicherbereich. Letzteres findet im Fall von Multithreading Anwendung. Des Weiteren spielen auch die unterstützten Programmiersprachen eine wichtige Rolle, da Entwickler lieber auf bekannte Sprachen zurückgreifen.

4.1.2 Eigenschaften von RIOT

Um auch auf den kleinsten Mikrocontrollern lauffähig zu sein ist RIOT kompatibel mit 16- und 32-Bit MCU (*microcontroller unit*) die weder über eine *Floating Point Unit*, noch über eine *Memory Management Unit* verfügen müssen. Eine Minimalinstallation ist auf einer Hardware mit 5 kByte ROM und 2 kByte RAM lauffähig. Um Energie zu sparen wurde ein *tickless scheduler* implementiert. Dieser schedult den *idle thread* falls keine Tasks anstehen. Der *idle thread* versetzt das System in den tiefstmöglichen Schlafmodus. Da RIOT in ANSI C implementiert wurde und C/C++ sowie die POSIX API unterstützt, bietet es Entwicklern den Komfort keine neue Sprache lernen zu müssen. Der Netzwerkstack unterschtützt die meisten gängigen Protokolle wie 6LoWPAN, RPL, IPv6, UDP und TCP. Seit 2015 ist der GNRC Netzwerkstack in RIOT implementiert. Er ist vollkommen modular aufgebaut und jede Schicht läuft in einem eigenen Thread, wobei die Priorität der Threads von der MAC-Schicht zur Anwendungsschicht abnimmt. Ankommende Datenpakete werden in einer *message queue* gespeichert. Threads dürfen in RIOT aus Sicherheitsgründen nicht untereinander kommunizieren. Deshalb muss der Empfängerthread mittels *Inter Process Communication* (IPC) die Nachrichten aus der *message queue* laden. Der multithreadingfähige Kernel unterstützt *zero-time interrupt handlers* gepaart mit Threadprioritäten und benötigt nur eine minimale Zeit beim Kontextwechsel. Um Echtzeitfähigkeit zu gewährleisten, führt der Scheduler immer den Thread mit der höchsten Priorität aus, sofern er nicht geblockt oder schlafend ist.

4.1.3 Weitere eingebettete Betriebssysteme

Neben RIOT sind TinyOS [LMP+05] und Contiki [DGV04] die gängigsten Betriebssysteme für eingebettete Systeme. TinyOS ist das kleinste der drei Systeme und wurde im Jahr 2000 von der Berkely Universität entwickelt. Es kommt aufgrund des monolithischen Kernels und einem einfachen FIFO (*First In First Out*) Schedulers mit sehr wenig Speicher zurecht. Contiki wurde 2003 vom schwedischen Programmierer Adam Dunkels entwickelt. Es kann sich vorallem durch den großen Funktionsumfang und einem grafischen Userinterface profilieren. Einer der großen Nachteile beider Betriebssysteme gegenüber RIOT ist, dass beide über einen Ereignisbasierten Scheduler verfügen, wodurch keine Echtzeitfähigkeit gewährleistet werden kann. Ein genereller Vergleich der drei Betriebssysteme wird in Tabelle 4.1 veranschaulicht. Da diese Arbeit in RIOT implementiert wurde, wird auf TinyOS und Contiki an dieser Stelle nicht weiter eingegangen.

	RAM	ROM	C	C++	Multithreading	MMU	Modularität	Echtzeitfähig
Contiki	<2 kB	<30 kB	Teilweise	Nein	Teilweise	Ja	Teilweise	Teilweise
TinyOS	<1 kB	<4 kB	Nein	Nein	Teilweise	Ja	Nein	Nein
RIOT	~1,5 kB	~5 kB	Ja	Ja	Ja	Ja	Ja	Ja

Tabelle 4.1: Vergleich der Betriebssysteme[BHW+12]

4.2 Relic toolkit

Relic [AG] ist eine moderne Kryptografiebibliothek, die von Diego F. Aranha ins Leben gerufen wurde. Seit 2009 ist es unter einer LGPL Lizenz auf Github verfügbar. Relic wird

vor allem zur Erstellung von angepassten kryptographischen Toolkits verwendet. Bei der Implementierung wurde besonders auf maximale Effizienz, flexible Konfiguration, einfache Portierbarkeit und hohe Kompatibilität mit anderen Kryptolibraries geachtet. Bisher wurden folgende Funktionalitäten implementiert:

- Langzahlarithmetik (*Multiple-precision integer arithmetic*)
- Prim- und Binärfeld Arithmetik
- Einige elliptische Kurven über Prim- und Binärfeldern
- Pairings (Bilineare Abbildungen)
- Zahlreiche kryptographische Protokolle wie u.A. RSA, Rabin, ECDSA und das Identitätsbasierte Signaturverfahren vbnn_ibs

Leider ist Relic nur sehr spärlich dokumentiert, so dass Kryptolibrary-Neulinge viel Einarbeitungszeit investieren müssen. Es wurden allerdings Testroutinen implementiert, in denen einzelnen Funktionen der Bibliothek getestet werden. Es ist zu empfehlen sich dort mit allen benötigten Modulen vertraut zu machen. In dieser Arbeit wurden die Module zur Langzahlarithmetik, elliptischen Kurven und Pairings verwendet.

4.3 Versuchsaufbau

Als Basis für die Implementierung wurde aufgrund der Multithreadingfähigkeit und Ressourceneffizienz das RIOT Betriebssystem ausgewählt. Für Client und Server dienen native RIOT Betriebssysteme, die mittels TUN/TAP auf einem Linux Host vernetzt wurden. Abbildung 4.1 zeigt das Testszenario mit einer logischen TTP (links oben im Bild), einem Knoten im Debug-Modus (links unten) und vier weiteren Knoten. Der letzte Knoten ist der Gruppe noch nicht beigetreten und der Knoten mit der Baum-ID acht hat eine signierte Nachricht an den Debug-Knoten versandt. Als Kryptographiebibliothek für ECC kam das äußerst effiziente Relic-toolkit, das bereits für RIOT portiert ist, zum Einsatz. Die Wahl der elliptischen Kurve fiel auf „BN_P254“ aus Relic. Diese Kurve erreicht ein Sicherheitslevel von 128 Bits [KSKK16] und erfüllt somit das, von der NIST bis zum Jahr 2030 empfohlene, Mindestsicherheitslevel von 112 Bit [BR15]. Da in RIOT die Speicherbereiche für verschiedene Threads getrennt sind, erfolgt die Interthreadkommunikation mittels *IPC Messaging*. Clients und Server kommunizieren mittels *UDP Sockets* aus dem GNRC Netzwerkstack auf verschiedenen Ports, wobei das erste Bit immer den Nachrichtentyp spezifiziert *(z.B.* $SIGNED\_MESSAGE = 5$ *oder* $EXTRACT = 3$*)*. Der Programmcode ist in vier *source files* und drei *header files* wie folgt aufgeteilt:

- **main. c:** Implementierung der Threads und das Hauptprogramm.
- **gs02.c** und **gs02.h:** C-Implementierung des GS02 Verfahrens aus der Abhandlung [GS02] mittels Relic.
- **gs02protocol.c** und **gs02protocol.h:** Allgemeine Konfigurationen durch *defines* und Implementierung von Funktionen die von Server und Client verwendet werden.
- **server.c, server.h, client.c** und **client.h:** Konfigurationen und Funktionen spezifisch für Server oder Client.

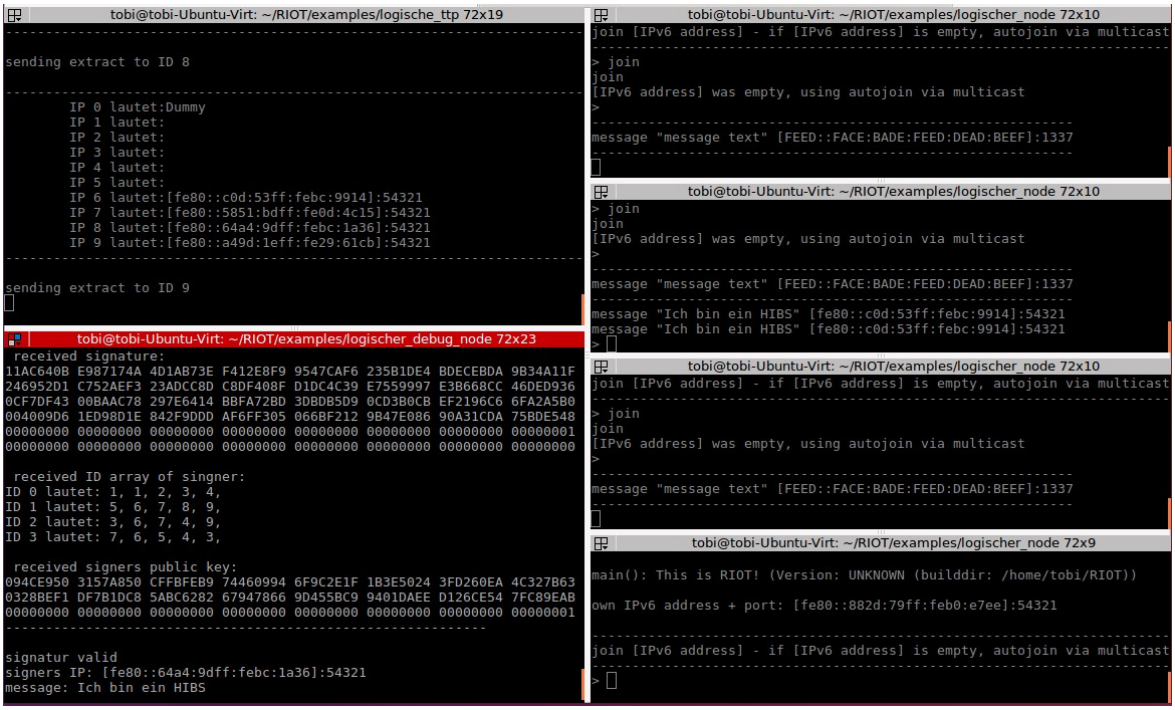

Abbildung 4.1: Versuchsaufbau auf einem Linux Host

4.4 Allgemeine Konfigurationsmöglichkeiten

RIOT verfügt über keine dynamische Heap-Speicherverwaltung. Aufgrund der statischen Speicherallokation muss die maximale Anzahl an Clients und die Art des Baumes (z.B. binär, ternär) schon zum Zeitpunkt des Compilierens feststehen. Dies zieht aber keiner wesentlichen Nachteile mit sich, da ohnehin ausreichend Arbeitsspeicher für einen gefüllten Baum zur Verfügung stehen muss. Die IPv6-Adressen werden von der TTP in einem *char Array* gespeichert. Die Wahl des Datentypen *char Array* ist im Nachhinein betrachtet speicherineffizient, da der in RIOT vordefinierte Datentyp *ipv6_addr_t* nur 16 Byte an Speicherplatz benötigt. Die eigentliche Intention war die Vergabe von besonders kurze IP-Adressen, da eingebettete Systeme meist in *Local Area Networks* kommunizieren und die Adressen daher frei wählbar sind. Die Länge der eindeutigen Client-*ID* wurde einheitlich auf eine feste größe gesetzt, da zu Testzwecken die *ID* einfach aus einem *Array* ausgelesen wird und somit günstiger zu Implementieren ist. Die allgemeinen Konfigurationen können durch folgende *defines* in *gs02protocol.h* konfiguriert werden:

```
1 #define SERVER_PORT 12345     // Port where the server listens
  #define NODE_PORT 54321       // Port where the client listens
3 #define IPV6LEN 36            // Size of the String of IPv6 address
  #define ID_SIZE 5             // Size of unsigned ints of the id
5 #define MAX_MESSAGE_SIZE 50 // Maximal size (characters) of the message
```

Listing 4.1: Allgemeine Konfigurationsmöglichkeiten

4.5 Rollenbasierte Baumstruktur

In der rollenbasierten Baumstruktur übernehmen die Clients die KGC Rolle von der TTP. IP-Adressen und öffentliche Schlüssel sind jeweils in einem *Array* gespeichert, dessen Index die Position in einem, in die Tiefe wachsenden, modifizierten K-Baum bestimmt. Mittels „*#define TREE_K_CHILDREN*“ aus der *ttp.h*, wird die Anzahl der Kinder pro Knoten definiert. Solange der *ID*- und *pk*-Pfad von der Wurzel bis zum Client berechnet werden kann, spielt die Festlegung auf eine fixe maximale Anzahl an Kindern pro Knoten eigentlich keine Rolle. Eine K-Baumstruktur hat jedoch den großen Vorteil, dass Kind- c bzw. Vaterknoten p durch folgende mathematische Formeln berechnet werden können, wobei i das i-te Kind des Vaterknotens beschreibt:

$$p = \lfloor \frac{c-1}{k} \rfloor, \quad c = k * p + i$$

Die TTP wird als außenstehende Partei betrachtet, die mehrere HIBS Bäume verwaltet. Deshalb besitzt diese in der Implementierung immer nur einen direkten Nachfolger. Daher mussten auch die Formeln zur Berechnung von Kind- und Vaterknoten, wie im Codebeispiel 4.2 beschrieben, angepasst werden.
Aufgrund der Nichtberechenbarkeit des öffentlichen Schlüssels muss jeder Client einen Schlüsselbaum in *public key array* führen. Jeder einzelne dieser Schlüssel benötigt 100 Byte Speicher.

```
int calculateParent(int k_tree_children, int own_id){
  if(own_id > 1) return(int)((own_id - 2) / k_tree_children + 1);
  return 0;
}

int calculateChild(int k_tree_children, int own_id, int i){
  return(k_tree_children * (own_id - 1) + i + 2);
}
```

Listing 4.2: Berechnung rollenbasierter Vater- bzw. Kindknoten

4.6 Logische Baumstruktur

Die logische Baumstruktur zielt auf die Minimierung der beim Rekey betroffenen Knoten ab. Nur auf dem untersten Rang im Baum befinden sich die Clients. Da diese ein vorberechnetes Pairing und den *public key* jedes Knotens im untersten logischen Rang speichern, spielt die Höhe des logischen Schlüsselbaumes keine Rolle. Jeder zusätzliche Rang benötigt zusätzlichen Speicherplatz, weshalb die Implementierung auf dem minimalsten Baum mit drei Rängen (0 - 1 logisch) basiert. Hierbei kann die Anzahl der Knoten auf der untersten logischen Ebene (Rang 1) sowie die maximale Anzahl derer Kinder durch „*#define TREE_LEEFS*“ und „*#define TREE_LOGICAL_NODES*“ konfiguriert werden. In diesem Verfahren könnte die TTP durch einen logischen Knoten Nachrichten signieren und ist somit ein Mitglied im Baum. Abbildung 4.2 zeigt eine Baumstruktur mit drei Knoten auf der untersten logischen Ebene mit jeweils drei Blättern.
Die implementierten Berechnungsfunktionen für direkte Vater- und Kindknoten sind nur auf Bäume mit drei Rängen anwendbar.

```
int calculateParent(int tree_leafs, int tree_logical_nodes, int own_id){
  if(own_id > tree_logical_nodes){
    return(int)((own_id - tree_logical_nodes - 1) / tree_leafs + 1);
  }
  return 0;
}

int calculateChild(int tree_leafs, int tree_logical_nodes, int own_id, int k){
  if(own_id > 0){
    return (tree_logical_nodes + 1 + (own_id - 1) * tree_leafs + k);
  }
  return k;
}
```

Listing 4.3: Berechnung logischer Vater- bzw. Kindknoten

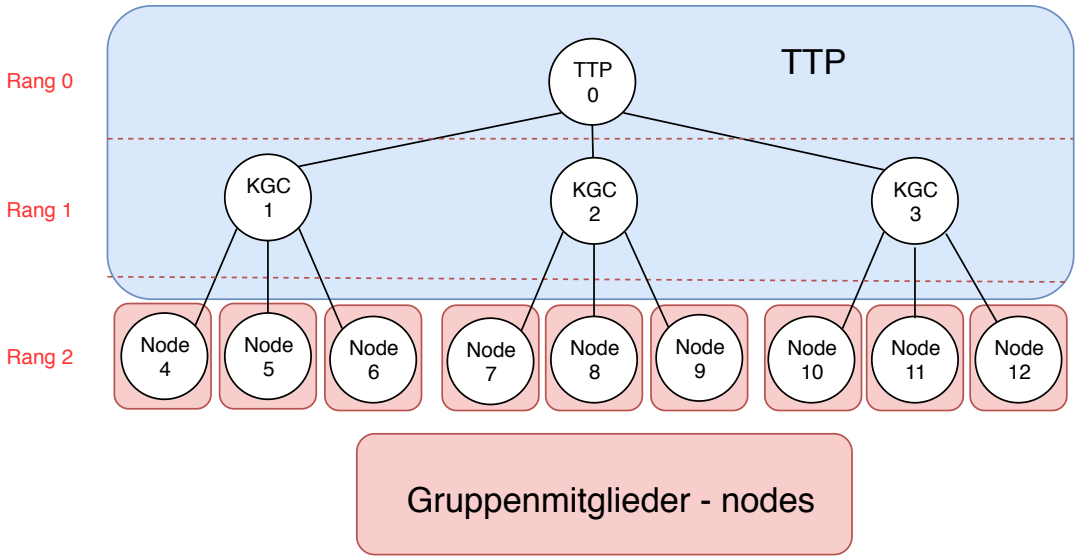

Abbildung 4.2: Logische 3-3 Baumstruktur

4.7 Besonderheiten der Implementierungen

Im Folgenden wird auf Besonderheiten und Schlüsselstellen der Implementierungen eingegangen. Sofern nicht anders angegeben wurden beide Protokolle strikt nach den Spezifizierungen in Kapitel 3 implementiert.

4.7.1 Gruppeneintritt

Initialisierung und Gruppeneintritt wurden analog zum Design aus Kapitel 3.2 bzw. 3.3 mit der Ausnahme des sicheren Kanals zum Schlüsselaustasch, implementiert. Der Prototyp zielt auf die Effizienz verschiedener Schlüsselbäume ab, weshalb der sichere Kanal zum Austausch des geheimen Kurvenpunktes außen vor gelassen wurde. Sollte der Prototyp weiterentwickelt werden, muss die sichere Verbindung, etwa über Diffie Hellman Schlüsselaustausch, realisiert werden.

Initial wählen sich TTP und Client eine Zufallszahl als *sk*. Diese multiplizieren sie mit

dem Generator der elliptischen Kurve und errechnen so ihren *pk*. Der geheime Kurvenpunkt der TTP ist das neutrale Element der elliptischen Kurve. Die Implementierung der Keygeneration von TTP und Clients wird in den Codebeispielen 4.4 und 4.5 verdeutlicht. Hat ein Client die Initialisierung abgeschlossen, wartet er auf die Nutzerinteraktion join. Die join Funktion kann mit oder ohne IP-Adresse aufgerufen werden. Sofern eine IP-Adresse angegeben wurde, wird der *join request* via Unicast an den Server versandt, anderenfalls sendet der Client den *join requeset* via Multicast an den Serverport. Letzteres sollte nur zu Testzwecken verwendet werden, da die IP-Adresse der TTP bekannt sein sollte. Anschließend wird der geheime Kurvenpunkt eines Clients von seinem KGC, mittels des geheimen Punktes und *ID-arrays* des KGC extrahiert. Das Codebeispiel 4.6 zeigt die Implementierung der Extraktion eines Kurvenpunktes von einem KGC. Der weitere Programmablauf unterscheidet sich in beiden Protokollen. Während das KGC bei dem rollenbasierten Verfahren dem neuen Knoten alle öffentlichen Schlüssel im Schlüsselbaum mitteilt, empfängt der Knoten im logischen Schlüsselbaum nur die vorberechneten Pairings und die öffentlichen Schlüssel der untersten logischen Ebene der TTP.

```
int cp_gs02_kgc_gen_root(bn_t msk,
    g1_t mpk, g2_t Sk0, g1_t P) {

  bn_t n;
  /* get order of the Group */
  g1_get_ord(n);

  /* master secret key */
  bn_rand_mod(msk, n);

  /* master public key = Qt[0] */
  g1_get_gen(P);
  g1_mul(mpk, P, msk);

  /* set Sk0 to infinity */
  g2_set_infty(Sk0);
  return STS_OK;
}
```

Listing 4.4: Root Keygeneration

```
int cp_gs02_kgc_gen_lower(bn_t lmsk
    , g1_t qt, g1_t P) {

  bn_t n;
  /* get order of the Group */
  g1_get_ord(n);

  /* user secret key */
  bn_rand_mod(lmsk, n);

  /* user public key qt[n] */
  g1_get_gen(P);
  g1_mul(qt, P, lmsk);
  return STS_OK;
}
```

Listing 4.5: Lower Level Keygeneration

```
int cp_gs02_kgc_extract_key(g2_t Sk_vorg, g2_t Sk, bn_t msk, uint8_t *
    identitys, int identity_len, int anz_IDs) {
  uint8_t *ident_ptr = identitys;

  // exclude 0. ID of KGC
  for(unsigned int i = 0; i < identity_len / sizeof(uint8_t); i++){
    ident_ptr++;
  }

  // Rk = H1(ID1,...,IDk)
  g2_map(hash, ident_ptr, (anz_IDs - 1)* identity_len);

  // tmp = Rk * msk
  g2_mul(tmp, hash ,msk);

  // S = Sk-1 + tmp
  g2_add(Sk, Sk_vorg, tmp);
  g2_norm(Sk, Sk);
  return STS_OK;
}
```

Listing 4.6: Extract

4.7.2 Kommunikation

Die Identifikation von Nachrichten erfolgt in beiden Protokollen über das erste Byte. Hierbei gibt es vier grundlegende Nachrichtentypen (EXTRACT_REQ, EXTRACT, SIGNED_MESSAGE und REKEY). Zusätzlich muss im rollenbasierten Verfahren nach der TTP mittels (TTP_REQUEST und TTP_REDIRECT) gefragt werden. Nach dem Eintritt neuer Knoten in den Baum, teilt dieser seien öffentlichen Schlüssel durch QT_NEW an alle anderen Knoten mit. Im logischen Baum werden die Nachrichten der vorberechneten Pairings und der

öffentlichen Schlüssel der letzten logischen Ebene (mittels HASH_GV und REFESH_KEY) identifiziert.

Signierung

Die Erstellung der Signatur einer Nachricht wurde äquivalent zu Kapitel 2.4.2 implementiert. Die Funktion unterscheidet sich in beiden Protokollen nicht. Signierte Nachrichten können von der Kommandozeile via Unicast mittels „message 'Text' 'IP-Adresse'“ oder per Multicast durch Weglassen der IP-Adresse versandt werden. Die Signatur wird zusammen mit dem *ID-array* des Senders sowie dem Nachrichtentext versandt.

```
int cp_gs02_user_sign(g2_t Sk, bn_t msk, uint8_t *identitys, int identity_len,
    int anz_IDs, uint8_t *message, int message_size, g2_t sigS){

  int buffer_size;
  uint8_t buffer[TREE_DEPTH * ID_SIZE + MAX_MESSAGE_SIZE];
  uint8_t *buf_ptr = buffer;
  uint8_t *ident_ptr = identitys;
  g2_t hash, tmp;

  // exclude 0. ID of KGC
  for(unsigned int i = 0; i < identity_len / sizeof(uint8_t); i++){
    ident_ptr++;
  }

  // Buffersize = msg_len + identy_len
  buffer_size = (anz_IDs - 1) * identity_len + message_size + 1;

  // Buffer contains IDs and message
  memcpy(buf_ptr, ident_ptr, identity_len * anz_IDs - identity_len);
  buf_ptr += identity_len * anz_IDs - identity_len;
  memcpy(buf_ptr, message, message_size);

  // Tm = H3(Id1,...,IDk,M)
  g2_map(hash, buffer, buffer_size);
  buf_ptr = NULL; ident_ptr = NULL;

  // tmp = Tm * msk
  g2_mul(tmp, hash ,msk);

  // S = Sk + Tm * msk
  g2_add(sigS, Sk, tmp);
  g2_norm(sigS, sigS);
  return STS_OK;
}
```

Listing 4.7: Signierung

Verifikation

Der wesentliche Unterschied bei der Verifikation in beiden Protokollen liegt im Rechenaufwand. Im logischen Verfahren müssen nur zwei Pairings und Hashes gebildet werden, da die TTP die Pairings von der Wurzel bis zum untersten logischen Rang schon vorberechnet

hat. Im rollenbasierten Verfahren hingegen muss sich der Empfänger erst ein temporäres *array* mit dem Schlüsselast des Senders erstellen, und anschließend die Pairings mit dem mitgesandten *ID-array* berechnen. Der Ast lässt sich zwar effektiv rekursiv erstellen, doch bei längeren Ästen sind die anfallenden Pairings mit erheblichem Rechenaufwand verbunden. Die Funktion „cp_gs02_user_verify“ vergleicht die mitgesandte Signatur mit den selbst berechneten Pairings. Stimmen beide überein, so ist die Signatur *valid* anderenfalls *invalid*.

```
uint8_t rekcalcQtArrayWorker(g1_t *Qt, g1_t *QtOut, int k_tree_children, int
    own_id, int *iter){
  if(own_id == 0){
    g1_copy(QtOut[0] , Qt[0]);
    return 1;
  }
  g1_copy(QtOut[rekcalcQtArrayWorker(Qt, QtOut, k_tree_children,
    calculateParent(k_tree_children, own_id), iter)] , Qt[own_id]);

  *iter += 1;
  return (*iter);
}

void rekcalcQtArray(g1_t *Qt, g1_t *QtOut, int k_tree_children, int own_id){
  int iter = 1;
  rekcalcQtArrayWorker(Qt, QtOut, k_tree_children, own_id, &iter);
}

...
rekcalcQtArray(Qt, QtOut, k_tree_children, sender_tree_id);
valid = cp_gs02_user_verify(users_id, user_id_size, anz_ids, message, *
    message_size, sigS, QtOut, P);
...
```

Listing 4.8: Rollenbasierte Verifikation

```
...
 valid = cp_gs02_user_verify(users_id, user_id_size, anz_ids, message, *
    message_size, sigS, Qtreceived, parentQt[calculateParent(tree_leafs,
    tree_logical_nodes, sender_tree_id) - 2], P, hashGV[calculateParent(
    tree_leafs, tree_logical_nodes, sender_tree_id) - 2]);
...
```

Listing 4.9: Logische Verifikation

4.7.3 Gruppenaustritt

In beiden Implementierungen wird der Gruppenaustritt immer von der TTP initiiert. Diese entfernt einen Knoten durch den Befehl „kick Baum-ID“ aus der Gruppe. Eine Clientseitige Gruppenaustrittsanfrage wurde nicht implementiert. Stürzt ein Client ab oder startet neu, so wird kein *Rekey* durchgeführt, sondern er behält seine ID im Baum. Im rollenbasierten Verfahren sendet ein Knoten, der einen *Rekey* durchgeführt hat oder neugestartet wurde, seinen öffentlichen Schlüssel an alle anderen Gruppenteilnehmer via *multicast*. Alle Knoten prüfen, wie im Codebeispiel 4.10 beschrieben, ob der Sender des Schlüssels sein rekursiver Vater ist. Trifft dies zu, führt der Knoten einen *Rekey* durch, andernfalls fügt er den emp-

fangenen öffentlichen Schlüssel in seinen lokalen Schlüsselbaum ein. Im logischen Verfahren betrifft der Neustart eines Knotens keinen anderen Gruppenteilnehmer. Im Falle des Ausschlusses eines Gruppenmitgliedes, berechnet die TTP ein neues Schlüsselpaar für seinen logischen Vater, und verteilt den neuen öffentlichen Schlüssel. Des Weiteren werden dessen weitere Kinder angewiesen einen *Rekey* durchzuführen.

```
...
tmp = calculateParent(k_tree_children, own_number);

while(tmp >= new_public_key_id){
  if(tmp == new_public_key_id){
    return(REKEY);
  }
  tmp = calculateParent(k_tree_children, tmp);
}
...
```

Listing 4.10: Erkennung ob der empfangene öffentliche Schlüssel sein rekursiver Vater ist

Rekey

Ein *Rekey* beschreibt die Schlüsselneugenerierung eines Knotens, falls dieser von der TTP darauf angewiesen wurde, oder dessen Vater ein neues Schlüsselpaar erzeugt hat. Dadurch wird die Signatur von, aus der Gruppe ausgeschlossenen Knotenen, mathematisch falsifiziert. Der größte Unterschied beider Protokolle beim *Rekey* liegt in der Effektivität. Im rollenbasierten Verfahren ändert sich beim *Rekey* die Position im Baum. Deshalb erfrägt der Knoten, mittels eines TTP_REQUEST, seine aktuelle Baum-ID und die IP seines KGC von der TTP. In der ursprünglichen Implementierung, bei der jeder Knoten seinen Schlüsselbaum behielt und nur der geänderte Schlüssel aktualisiert wurde, kam es zu folgenden Problemen. Meist mussten mehrere Knoten gleichzeitig einen *Rekey* durchführen, dies resultierte in Unstimmigkeiten in deren Schlüsselbäumen. Daher wurde in diesem Verfahren der *Rekey* quasi als Neustart des Knotens implementiert, bei dem er den kompletten Schlüsselbaum seines neuen Vaters kopiert.
In der logischen Baumstruktur hingegen wird lediglich der öffentliche Schlüssel, des logischen Knotens der einen *Rekey* ausführt, via *multicast* an alle Gruppenmitglieder verteilt. Falls Blätter ein neues Schlüsselpaar generieren, fällt keine Gruppenkommunikation an.

4.8 Evaluation

Beide Implementierungen wurden gegeneinander und gegen die Anforderungen an ein HIBS aus Kapitel 3 getestet.

4.8.1 Leistung und Speicherverbrauch

Zur Leistung auf einem eingebetteten System kann in dieser Arbeit keine Aussage gemacht werden, da die Prototypen auf einem nativen RIOT in einem Linux Hostsystem getestet wurden. Mit der Rechenleistung eines 3 GHz Prozessors gab es keine spürbare Verzögerung bei der Berechnung der Pairings und der Hashfunktionen. Es kann lediglich eine Aussage

über die Effizienz der Berechnung der Pairings getroffen werden. Die ursprüngliche Implementierung basierte auf symmetrischen Pairings, welche auf der einzigen supersingulären elliptischen Kurve „SS_P1536“ operieren, die in Relic implementiert ist. Auf dieser Kurve benötigte der 3 GHz Prozessor pro Pairing etwa eine Sekunde. Durch den Umstieg auf asymmetrischen Pairings auf einer wesentlich kleineren Kurve, konnte eine erhebliche Leistungssteigerung beobachtet werden. In der modernen Pairing basierenden Kryptografie wird bevorzugt zu asymmetrischen Pairings gegriffen, da diese wesentlich effektiver berechenbar sind als symmetrische [BCM+15].
Da Pairings und Hashfunktionen den Großteil des Rechenaufwands ausmachen, sind sie ausschlaggebend für die Effizienz der zugrundeliegenden Baumstruktur. Im GS02 Verfahren werden Pairings nur bei der Nachrichtenverifikation berechnet, wobei deren Anzahl dem Rang des Senders im Baum entspricht. Daher ist für die Effektivität der Kommunikation ein möglichst flacher Baum mit vielen Kindern pro Knoten ideal. Mit einer hohen Anzahl an Kindern geht jedoch der Nachteil einher, dass im Falle eines Gruppenaustrittes eines Knotens alle anderen Kinder des selben Vaters einen *Rekey* durchführen müssen. Folglich ist es vorteilhaft ständige Gruppenmitglieder nahe der Wurzel, und häufig wechselnde an den Blättern anzusiedeln.

Der gesamte Speicherverbrauch hängt im Wesentlichen von der eingesetzten Baumstruktur und der Anzahl der Kinder pro Knoten ab. Der generelle Speicherverbrauch eines einzelnen Schlüssels unter Verwendung der Kurve „BN_P254“ aus Relic, wird in Tabelle 4.2 veranschaulicht.

Datentyp	*Speicherverbrauch in Byte*
Öffentlicher Schlüssel	100
Geheime Zufallszahl	276
Geheimer KurvenPunkt	196
Summe privater Schlüssel	472
Vorberechnete Pairings	384

Tabelle 4.2: Speicherverbrauch von Datentypen

Im rollenbasierten Verfahren, bei dem jeder Gruppenteilnehmer alle öffentlichen Schlüssel kennen muss, wird auf jedem Knoten Speicherplatz für sein eigenes Schlüsselpaar (572 Byte), sowie jeweils 100 Byte für jeden weiteren Gruppenteilnehmer inklusive TTP benötigt. Der Speicherverbrauch ist daher unabhängig von der gewählten Anzahl an Kinder pro Knoten. Somit skaliert der Speicherverbrauch in diesem Verfahren linear zur Anzahl der Gruppenmitglieder. In der logischen Baumstruktur besteht der Speicherverbrauch auf jedem Knoten lediglich aus dem eigenen Schlüsselpaar (572 Byte), sowie jeweils aus 484 Byte für das vorberechnete Pairing und den öffentlichen Schlüssel für jeden Knoten auf der untersten logischen Ebene. Die maximale Speichereffektivität wird durch möglichst wenige Knoten auf der untersten logischen Ebene mit vielen Kindern pro Knoten erzielt.

In einer Desktop-PC-Umgebung wäre aufgrund des riesigen Arbeitsspeichers eine logische Baumstruktur mit vielen Knoten auf der untersten logischen Ebene das Mittel der Wahl. So sind bei einem Gruppenaustritt nur wenige andere Teilnehmer betroffen. Für die eingebetteten Systeme im *Internet of Things* hingegen, gibt es aufgrund der beschränkten Ressourcen keinen wirklichen Königsweg. Da Energie im IoT besonders kostbar ist, hat das logische Verfahren gegenüber dem rollenbasierten den Vorteil, dass beim Gruppeneintritt

eines Knotens, dessen öffentlicher Schlüssel nicht an die anderen Gruppenteilnehmer verteilt werden muss. Des Weiteren ist der Rechenaufwand bei der Signierung und Verifikation der Nachrichten ebenfalls geringer. Falls die Hardwareausstattung der TTP deutlich besser ist als die der Knoten, so ist auch hier das logische Verfahren zu bevorzugen. Im Bezug auf die optimale Anzahl an Kindern pro Knoten kann pauschal keine konkrete Empfehlung gegeben werden, denn es kommt vor allem auf die Abwägung zwischen der Speicherausstattung und der Häufigkeit von Gruppenein- und austritten an.

Die Stärke des rollenbasierten Verfahrens liegt in der Entlastung der TTP. Daher muss eine TTP mit schwacher Hardwareausstattung, die ganze Rechenlast der Schlüsselerzeugung nicht alleine stemmen. Ein weiteres sinnvolles Einsatzszenario wäre eine Gruppe, bei der keine Gruppenmitglieder ausgeschlossen werden. In diesem Fall müssen die öffentlichen Schlüssel nicht an jeden Knoten verteilt werden, sondern ein Sender kann seinen eigenen Schlüsselast an eine signierte Nachricht anhängen.

4.8.2 Sicherheit und Funktionalität

Alle in Kapitel 3 geforderten Funktionalitäten (Gruppeneintritt, Gruppenkommunikation, Gruppenaustritt) an ein HIBS-Protokoll wurden in den zwei Prototypen voll funktionsfähig implementiert und getestet. Die kryptografische Sicherheit des GS02 Verfahrens wurde nicht erneut mathematisch überprüft, sondern es wurde auf die Abhandlung von Alice Silverberg und Craig Gentry [GS02] vertraut. In den Implementierungen wurde lediglich auf den Aufbau eines sicheren Kanals zum Schlüsselaustausch verzichtet. Daher sind beide Verfahren durch Abhören des Gruppeneintritts angreifbar. Im rollenbasierten Verfahren könnte ein Vaterknoten geheime Schlüssel für all seine Kinder generieren, und somit die Authentität und Verbindlichkeit gefährden. Dadurch ist es in diesem Verfahren unabdingbar, dass jedes Gruppenmitglied alle öffentlichen Schlüssel kennt. Denn diese hängen von den selbstgenerierten Geheimnissen jedes Gruppenmitgliedes ab und könnnen vom KGC nicht berechnet werden.

5 Zusammenfassung und Ausblick

Das Ziel der Arbeit, die Implementierung und Evaluierung zweier hierarchischer identitätsbasierter Signaturverfahren zur Gruppenkommunikation im IoT, wurde im Zuge dieser Bachelorarbeit erreicht. Beide beruhen auf dem GS02 HIBS-Verfahren (Kapitel 2.4.2) und umfassen den Gruppeneintritt mit Schlüsselgeneration und Extraktion, die Gruppenkommunikation mit Signierung und Verifikation, sowie den Gruppenaustritt mit Schlüsselerneuerung (*Rekey*). GS02 bietet sich besonders für Baumstrukturen an, da es einen *Rekey* erlaubt ohne das zentrale Geheimnis in der Wurzel zu ändern. In dieser Bachelorarbeit wurden zwei verschiedene Schlüsselbäume entwickelt und implementiert, sowie die Anzahl der betroffenen Knoten beim *Rekey* untersucht. Eine Implementierung beruht auf einer logischen Baumstruktur, bei der nur die Blätter Gruppenteilnehmer darstellen, und Schlüssel ausschließlich von der TTP generiert werden. Die Andere beruht auf einem rollenbasierten Baum bei dem die TTP ihre schlüsselgenerierende Aufgabe an die Knoten im Baum verteilt. Für beide Protokolle wurde jeweils eine TTP als Server und Knoten als Clients implementiert. Hierbei fungiert die TTP zusätzlich als Manager der Baumstruktur, indem sie jedem Knoten seine Position im Baum zuweist und ihn auch ggfs. wieder entfernt.

Tritt ein Knoten in die Gruppe ein, kontaktiert er die TTP und bekommt einen Vaterknoten zugewiesen. Dieser generiert den geheimen Schlüssel aus dem eigenen Geheimnis und der *ID* des neuen Gruppenmitgliedes. Beides wird gemeinsam mit dem *ID*- und *pk*-Ast von der Wurzel bis zum neuen Gruppenmitglied zurückgesandt. Der Knoten ist nun der Gruppe beigetreten und kann, wie in Kapitel 3 beschrieben, Nachrichten anhand seines *ID*-Astes und seines geheimen Schlüssels signieren, und durch den *ID*- und *pbk*-Pfads des Nachrichtenverfassers verifizieren. Um einen Knoten aus der Gruppe auszuschließen, muss sein geheimer Schlüssel mathematisch invalidiert werden. Dazu führt sein Vaterknoten einen *Rekey* durch. Dadurch bedingt werden auch die Schlüssel all seiner Kinder ungültig, sodass diese ebenfalls mit neuen Schlüsselpaaren ausgestattet werden müssen.

Der Vorteil von HIBS gegenüber herkömmlichen IBS ist ein effektiverer *Rekey*. Die Anzahl der Betroffenen Clients hängt vor allem von der zugrunde liegenden Baumstruktur ab. Bäume mit vielen Kindern Pro Knoten minimieren den Speicher- und Rechenaufwand bei der Kommunikation, doch beim Gruppenaustritt müssen viele Kinder einen *Rekey* durchführen. In Bäumen mit wenigen Kindern Pro Knoten hingegen sind sehr speicherintensiv aber dafür sind nur wenige Knoten von einem *Rekey* betroffen. Je geringer der Abstand eines Knotens zur Wurzel, desto weniger Hasehes und Pairings müssen bei der Signierung und Verifikation der Nachricht berechnet werden. Daher sind Signaturen wurzelnaher Baumknoten effektiver berechenbar als die der Blätter.

Die Entwicklung Vernetzung der Dinge hat erst begonnen, doch Kriminelle haben die Sicherheitslücken in den leistungsschwachen Geräten des IoT schon längst erkannt und ausgenutzt. In der Zukunft werden die eingebetteten Systeme immer näher an den Menschen

heranrücken, um ihm den Alltag zu erleichtern. Hierarchische identitätsbasierte Signaturen bieten einen sicheren und effizienten Weg den Versender einer Nachricht zu identifizieren, und die übertragenen Daten vor Veränderung zu schützen. Die in dieser Arbeit entwickelten Prototypen können in zukünftigen Arbeiten weiterentwickelt und optimiert werden. Beide Protokolle unterstützen im Moment nur *ID*s mit vordefinierter Länge. Wird eines der Protokolle zum produktiven Einsatz weiterentwickelt, ist es sinnvoll eine E-Mail- oder MAC-Adresse als *ID* zu verwenden. Des Weiteren wird der geheime Kurvenpunkt bei der Schlüsselextraktion noch im Klartext übertragen. Um das Abhörrisiko zu senken, ist es sinnvoll einen sicheren Kanal zum Schlüsselaustausch aufzubauen. In den Implementierungen wird der Schlüsselbaum immer von links nach rechts aufgefüllt. Daher kommt es dazu, dass einige Knoten im vorletzten Rang vollbesetzt und andere wiederum leer sind. Ein effektiveres *Rekeyverhalten* würde ein Ansatz bieten, bei dem einen neuer Knoten immer an den Vaterknoten mit den wenigsten Kindern im vorletzten Rang gehängt wird. Dadurch kann die Anzahl der betroffenen Knoten im Falle eines Gruppenaustrittes minimiert werden.

Abbildungsverzeichnis

Literaturverzeichnis

[AG] Aranha, D. F. ; Gouvêa, C. P. L.: *RELIC is an Efficient Library for Cryptography*. https://github.com/relic-toolkit/relic, . – abgerufen am 10. April 2018

[ALYW06] Au, Man H. ; Liu, Joseph K. ; Yuen, Tsz H. ; Wong, Duncan S.: *Practical Hierarchical Identity Based Encryption and Signature schemes Without Random Oracles*. https://eprint.iacr.org/2006/368.pdf. Version: 2006

[AUN] *ARDUINO UNO REV3*. https://store.arduino.cc/arduino-uno-rev3, . – abgerufen am 15. Mai 2018

[BCM+15] Barreto, Paulo S. L. M. ; Costello, Craig ; Misoczki, Rafael ; Naehrig, Michael ; Pereira, Geovandro C. C. F. ; Zanon, Gustavo: Attribute-Based Signatures. Version: 2015. http://dx.doi.org/10.1007/978-3-319-22174-8. In: *Progress in Cryptology – LATINCRYPT 2015*. Cham : Springer Cham, 2015. – DOI 10.1007/978–3–319–22174–8. – ISBN 978–3–319–22174–8

[BHW+12] Baccelli, Emmanuel ; Hahm, Oliver ; Wählisch, Matthias ; Günes, Mesut ; Schmidt, Thomas: RIOT: One OS to Rule Them All in the IoT. In: *Research Report RR-8176*, 2012

[BLM05] Barreto, Paulo S. ; Libert, Benoit ; McCullagh, Noel: Efficient and Provably-Secure Identity-Based Signatures and Signcryption from Bilinear Maps. Version: 2005. http://dx.doi.org/10.1007/11593447. In: Roy, Bimal (Hrsg.): *Advances in Cryptology - ASIACRYPT 2005*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2005. – DOI 10.1007/11593447. – ISBN 978–3–540–32267–2, 515–532

[BLS04] Boneh, Dan ; Lynn, Ben ; Shacham, Hovav: Short Signatures from the Weil Pairing. Version: 2004. http://dx.doi.org/10.1007/s00145-004-0314-9. In: *Journal of Cryptology*. Springer-Verlag, 2004. – DOI 10.1007/s00145–004–0314–9. – ISSN 1432–1378, 297-319

[BR15] Barker, Elaine ; Roginsky, Allen: Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths / National Institute of Standards and Technology. Version: 2015. http://dx.doi.org/10.6028/NIST.SP.800-131Ar1. 2015. – Forschungsbericht

[Che06] Cheon, Jung H.: Security Analysis of the Strong Diffie-Hellman Problem. Version: 2006. https://doi.org/10.1007/11761679_1. In: *Advances in Cryptology - EUROCRYPT 2006*. Springer, 2006. – ISBN 978–3–540–34547–3

[Cou08] Council, National I.: Disruptive Civil Technologies. In: *Six Technologies With Potential Impacts on US Intrests Out to 2025*, 2008

[DGV04] Dunkels, A. ; Gronvall, B. ; Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: *29th Annual IEEE International Conference on Local Computer Networks*, 2004. – ISSN 0742–1303, S. 455–462

[FGCG18] Felde, Nils gentschen ; Grundner-Culemann, Sophia ; Guggemos, Tobias: *Using identity-based signatures for authenticated group communication.* 2018

[GC17] Grundner-Culemann, Sophia: *Identity-based source authentication in constrained networks*, Ludwig Maximilians Universität München, Diplomarbeit, 2017

[GS02] Gentry, Craig ; Silverberg, Alice: Hierarchical ID-Based Cryptography. Version: 2002. `http://dx.doi.org/10.1007/3-540-36178-2`. In: Yuliang (Hrsg.) ; Zheng (Hrsg.): *Advances in Cryptology - ASIACRYPT 2002*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2002. – DOI 10.1007/3–540–36178–2. – ISBN 978–3–540–36178–7, 548–566

[Inf12] Informatik, Bundesamt für Sicherheit in d.: *Technical Guideline TR-0311 Elliptic Curve Cryptography.* `https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr03111/index_htm.html`. Version: Version 2.0, 2012. – abgerufen am 10. April 2018

[Inf18] Informatik, Bundesamt für Sicherheit in d.: *BSI - Technische Richtline.* `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile`. Version: Januar 2018. – abgerufen am 10. April 2018

[Jou04] Joux, Antoine: A One Round Protocol for Tripartite Diffie-Hellman. Version: 2004. `http://dx.doi.org/10.1007/s00145-004-0312-y`. In: *Journal of Cryptology*. Springer-Verlag, 2004. – DOI 10.1007/s00145–004–0312–y. – ISSN 1432–1378, 263–276

[Kob87] Koblitz, Neal: Elliptic Curve Cryptosystems. In: *American Mathematical Society* 48 (1987), Nr. 177, S. 203 – 209

[KSKK16] Kato, Akihiro ; Scott, Michael ; Kobayashi, Tetsutaro ; Kawahara, Yuto: Barreto-Naehrig Curves / Internet Engineering Task Force. 2016. – Forschungsbericht

[KU16] Kiraz, Mehmet S. ; Uzunkol, Osmanbey: *Still Wrong Use of Pairings in Cryptography.* November 2016

[LMP+05] Levis, P. ; Madden, S. ; Polastre, J. ; Szewczyk, R. ; Whitehouse, K. ; Woo, A. ; Gay, D. ; Hill, J. ; Welsh, M. ; Brewer, E. ; Culler, D.: TinyOS: An Operating System for Sensor Networks. Version: 2005. `http://dx.doi.org/10.1007/3-540-27139-2-7`. In: *Ambient Intelligence*. 2005. – DOI 10.1007/3–540–27139–2–7. – ISBN 978–3–540–27139–0, S. 115–145

[McC90] McCurley, Kevin S.: The Discrete Logartithm Problem. In: *Proceedings of Symposia Applied Mathematics* 42 (1990), S. 49 – 74

[Mel18] Melnikov, Andrian: *A Testbed for Evaluating ID-based Authentication in Constrained Networks.* 2018

[Mil85] Miller, Victor S.: *Use of Elliptic Curves in Cryptography*. Springer Berlin Heidelberg, 1985. http://dx.doi.org/10.1007/3-540-39799-X_31. http://dx.doi.org/10.1007/3-540-39799-X_31. – ISBN 978–3–540–16463–0

[MPR11] Maji, Hemanta K. ; Prabhakaran, Manoj ; Rosulek, Mike: Attribute-Based Signatures. Version: 2011. http://dx.doi.org/10.1007/978-3-642-19074-2. In: Preneel, Bart (Hrsg.): *Topics in Cryptology - CT-RSA 2011*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011. – DOI 10.1007/978–3–642–19074–2. – ISBN 978–3–642–19073–5, 376–392

[MSP11] Müller-Stach, Stefan ; Piontkowski, Jens: *Elementare und algebraische Zahlentheorie*. 2. Auflage. Wiesbaden 2011 : Vieweg+Teubner Verlag — Springer Fachmedien Wiesbaden GmbH, 2011. http://dx.doi.org/10.1007/978-3-8348-8263-9. http://dx.doi.org/10.1007/978-3-8348-8263-9. – ISBN 978–3–8348–1256–8

[PS12] Peng, SheQiang ; Shen, HongBing: Security Technology Analysis of IOT. Version: 2012. http://dx.doi.org/10.1007/978-3-642-32427-7. In: Wang, Yongheng (Hrsg.) ; Zhang, Xiaoming (Hrsg.): *Internet of Things*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012. – DOI 10.1007/978–3–642–32427–7. – ISBN 978–3–642–32426–0, 401–408

[Rie16] Ries, Uli: *Hacker-Spaß mit Hue-Leuchten*. https://www.heise.de/security/meldung/Hacker-Spass-mit-Hue-Leuchten-3289481.html. Version: 2016. – abgerufen am 6. April 2018

[SB09] Shelby, Zach ; Bormann, Carsten: *6LoWPAN: The Wireless Embedded Internet*. John Wiley and Sons Ltd, 2009

[Sha84] Shamir, Adi: Identity-Based Cryptosystems and Signature Schemes. Version: 1984. http://dx.doi.org/doi.org/10.1007/3-540-39568-7-5. In: *Advances in Cryptology - Proceedings of CRYPTO 84*. 1984. – DOI doi.org/10.1007/3–540–39568–7–5. – ISBN 978–3–540–39568–3, S. 47–53

[SSN09] Shahandashti, Siamak F. ; Safavi-Naini, Reihaneh: Efficient and Provably-Secure Identity-Based Signatures and Signcryption from Bilinear Maps. Version: 2009. http://dx.doi.org/10.1007/978-3-642-02384-2. In: Preneel, Bart (Hrsg.): *Progress in Cryptology - AFRICACRYPT 2009*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009. – DOI 10.1007/978–3–642–02384–2. – ISBN 978–3–642–02384–2, 198–216

[YD08] Yanli, REN ; Dawu, GU: Efficient Hierarchical Identity Based Signature Scheme in the Standard Model. Version: 2008. http://dx.doi.org/10.1007/s11859-008-0606-2. In: *Wuhan Iniversity Journal of Natural Sciences*. Wuhan University, 2008. – DOI 10.1007/s11859–008–0606–2. – ISSN 1993–4998, 665-669