Institut *für* Mathematik

Universität Augsburg
Institut für Mathematik
Prof. Dr. Dirk Hachenberger (Erstgutachter)
Prof. Dr. Dieter Jungnickel (Zweitgutachter)
Lehrstuhl für Diskrete Mathematik, Optimierung und Operations Research
Universitätsstraße 14 86159 Augsburg

Die Entwicklung eines neuen Kryptosystems am Beispiel NTRU

—

Probleme und Chancen

Diplomarbeit

von Felix von Eye

zur Erlangung des Grades eines
Diplom-Mathematikers
der Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Augsburg

6. August 2009

Vorwort

Wer die Vergangenheit nicht kennt,
kann die Gegenwart nicht verstehen.
Wer die Gegenwart nicht versteht,
kann die Zukunft nicht gestalten.

(Hans-Friedrich Bergmann)

Wer ein neues Kryptosystem entwickeln oder ein vorhandenes Kryptosystem untersuchen möchte, muss sich der Geschichte der Kryptologie bewusst werden. Denn in der Vergangenheit haben sich viele Verfahren als unsicher erwiesen, obwohl sie als *unknackbar* galten und nicht selten wurden erfolgreiche Angriffe erst möglich, indem man die zu untersuchenden Probleme von verschiedenen Seiten betrachtet hat. Somit ist diese Arbeit Rückblick und Ausblick auf die Welt der Kryptologie zugleich.

Das in dieser Arbeit untersuchte Kryptosystem NTRU (Number Theory Research Unit) hat möglicherweise viel Potential, vor allem da es neue Verfahren benutzt, die bis dahin in Kryptosystemen noch nicht angewandt wurden. Auf der anderen Seite birgt jede Neuerung die Gefahr, dass durch diese Schwachstellen entstehen und damit das Kryptosystem gebrochen werden kann. Eine abschließende Gegenüberstellung der Vorteile und Probleme von NTRU werde ich am Ende der Arbeit durchführen.

Danken möchte ich an dieser Stelle vor allem Herrn Prof. Dr. Hachenberger, der mir die Bearbeitung dieses interessanten Themas ermöglicht hat und jederzeit für Fragen offen war. Weiterhin möchte ich mich bei Robert Bertossi und Volker Rossipal für ihre fachlichen Disussionen bedanken. Für den nötigen nichtwissenschaftlichen Rückhalt bedanke ich mich bei meiner Frau, die während der Bearbeitungszeit manch schwere Stunde mit mir aushalten musste.

Inhaltsverzeichnis

1. Einleitung

Die Kryptologie hat eine sehr wechselhafte Geschichte hinter sich. Viele verschiedene Methoden wurden ausprobiert, um die *unknackbare Verschlüsselung* zu entwickeln. Der überwiegende Teil der verschlüsselten Nachrichten wurde jedoch nach und nach entziffert und nicht selten wurden Kryptosysteme, die als unknackbar galten, doch gebrochen. Ich werde in meiner Arbeit auf diesen Aspekt eingehen, indem ich einen Überblick über die verschiedenen Stationen in der Geschichte der Kryptologie gebe, verschiedene Methoden zur Entzifferung vorstelle und am Beispiel NTRU[1] die Entwicklung eines neuen Kryptosystems aufzeige.

Ziel dieser Arbeit ist, ein modernes Kryptosystem zu untersuchen und herauszustellen, welche Chancen sich bieten, wenn man ein neues Kryptosystem entwickelt, aber auch, welche Probleme auftreten können. Das hier betrachtete Kryptosystem ist NTRU, wobei ich mich bei der Analyse hauptsächlich auf die von NTRU abgeleiteten Signierverfahren NSS[2], R-NSS[3] und NTRUSign[4] beziehen werde.

NTRU gehört zu der Klasse der Gitterverschlüsselungsverfahren und weicht damit von den momentan vorherrschenden Verfahren ab, die auf Primfaktorzerlegung oder dem diskreten Logarithmus basieren, wie zum Beispiel RSA[5] oder ECC[6]. Der größte Vorteil von NTRU ist seine Effizienz. Jedoch ist das Verfahren noch relativ neu und daher noch nicht ausreichend untersucht, weswegen immer wieder größere und kleinere Sicherheitslücken in der aktuellen Forschung gefunden werden. Bisher konnten die Entwickler das Verfahren jedoch immer wieder verbessern, so dass die für die vorherige Version angepassten Angriffe ins Leere laufen.

Aufbau der Arbeit

Zum Einstieg wird in Kapitel 2 ein Motivationsbeispiel durchgespielt, anhand dessen die Grundzüge der Kryptologie vorgestellt werden. Weiterhin definiere ich an dieser Stelle einige Begriffe, die ich in der folgenden Arbeit immer wieder verwenden werde.

In Kapitel 3 stelle ich eine Reihe von Kryptosystemen in der langen Geschichte der Kryptologie vor. Ziel dieses Kapitels ist, zu sehen, welche die wichtigsten Methoden der klassischen und modernen Verschlüsselung sind und wie der Einsatz in der Praxis funktioniert:

[1] Number Theory Research Unit
[2] NTRU Signature Scheme
[3] Revised NTRU Signature Scheme
[4] NTRU Signature Algorithm
[5] Rivest Shamir Adleman
[6] Elliptic Curve Cryptography

- symmetrische Verschlüsselung
  - monoalphabetische Substitution
  - polyalphabetische Substitution
  - Transposition
  - Rotor-Chiffriermaschinen
- asymmetrische Verschlüsselung
  - Public-Key
- No-Key-Protokoll

Im Verlauf von Kapitel 4 stelle ich einige Verfahren der Kryptoanalyse vor, mit denen man die in Kapitel 3 vorgestellten Kryptosysteme brechen kann. Diese Auswahl der Methoden ist nicht vollständig, soll aber einen Überblick über die Möglichkeiten der Kryptoanalyse geben. Ich verzichte dabei meist darauf, zu erwähnen, welche Methode man zum Entziffern welches Kryptosystems benutzen kann, da die Kryptoanalyse häufig ein Zusammenspiel mehrerer Methoden ist.

In Kapitel 5 werde ich einige theoretische Aspekte der Kryptologie diskutieren. Dabei untersuche ich, wie die Sicherheit von Kryptosystemen gemessen werden kann und wie alle Kommunikationspartner sicherstellen können, dass die Nachrichten ungefälscht sind und dass es keinen Kommunikationspartner gibt, der eine falsche Identität vortäuscht.

Die auf NTRU basierenden Verfahren NTRUEncrypt[7], NSS, R-NSS und NTRUSign werden in Kapitel 6 eingeführt. Dafür stelle ich zuerst die grundlegenden Ideen vor, auf denen NTRU basiert. Anschließend erläutere ich verschiedene Anpassungen und diskutiere deren Vor- und Nachteile.

Die NTRU-basierenden Signaturverfahren NSS und R-NSS werden daraufhin in Kapitel 7 kryptoanalytisch untersucht, wobei ich auch darauf eingehen werde, wie Verbesserungen in den Verfahren vorher mögliche Angriffe verhindern.

Im Kapitel 8 versuche ich einen Ausblick auf die Kryptolandschaft der Zukunft zu werfen, der zugleich auch ein Rückblick auf die bewegte Geschichte ist.

Die mathematischen Grundlagen für das NTRU-Verfahren, Gittertheorie, werde ich in Anhang A kurz erläutert. Dies ist keine umfangreiche Ausarbeitung über Gittertheorie, erklärt aber die wichtigsten Grundlagen.

Anhang B ist noch einmal eine Auflistung aller in dieser Arbeit vorgestellten Kryptosysteme. Dabei wird sowohl die Chiffrierung als auch die Entschlüsselung mit Hilfe der vorgestellten Verfahren in Pseudocode beschrieben.

[7] NTRU Encryption Algorithm

2. Was ist Kryptologie?

Bevor ich damit beginne, eine formale Definition des Begriffs *Kryptologie* zu geben und um Unterschiede zu verwandten Methoden, wie der *Steganographie*, aufzuzeigen, betrachte ich als Motivationsbeispiel eine ganz alltägliche Kommunikation und erläutere an diesem Beispiel die grundsätzlichen Ideen der Kryptologie.

Beispiel 2.1. Person A steht an der Kasse im Supermarkt C und zahlt per Kreditkarte der Bank B seinen Einkauf. Um den Bezahlvorgang abzuschließen, muss A alle nötigen Überweisungsdaten an B übermitteln und die Überweisung mit Hilfe der PIN autorisieren.

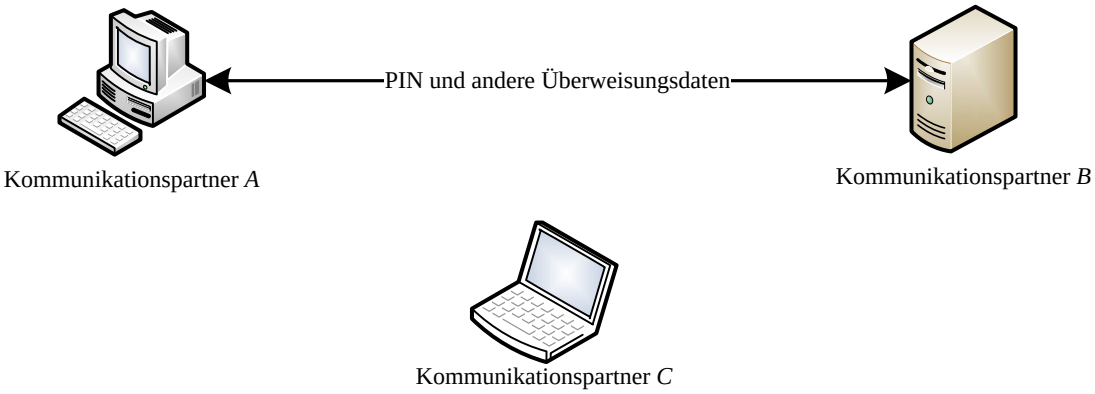

Abbildung 2.1.: Ungestörte aber öffentliche Kommunikation

Dieses alltägliche Beispiel einer Kommunikation zwischen zwei Parteien A und B im Beisein einer dritten Partei C, in Abbildung 2.1 zu sehen, offenbart die grundsätzliche Problematik, die bei einer vertraulichen Kommunikation auftreten kann. Insbesondere sind die Probleme:[1]

Die Geheimhaltung: *Wie kann ich verhindern, dass C, wie in Abbildung 2.2 skizziert, meine PIN-Eingabe mitlesen kann?*

Für die Geheimhaltung bieten sich folgende Verfahren an:

- **Organisatorische Maßnahmen:** Beispiele wären ein vertrauenswürdiger Bote oder die Einstufung als „Verschlusssache“.
- **Physikalische Maßnahmen:** Man verwahrt die Nachricht zum Beispiel in einem Tresor oder einem versiegeltem Brief. Man kann aber auch Methoden der

[1] vgl. [BSW98]

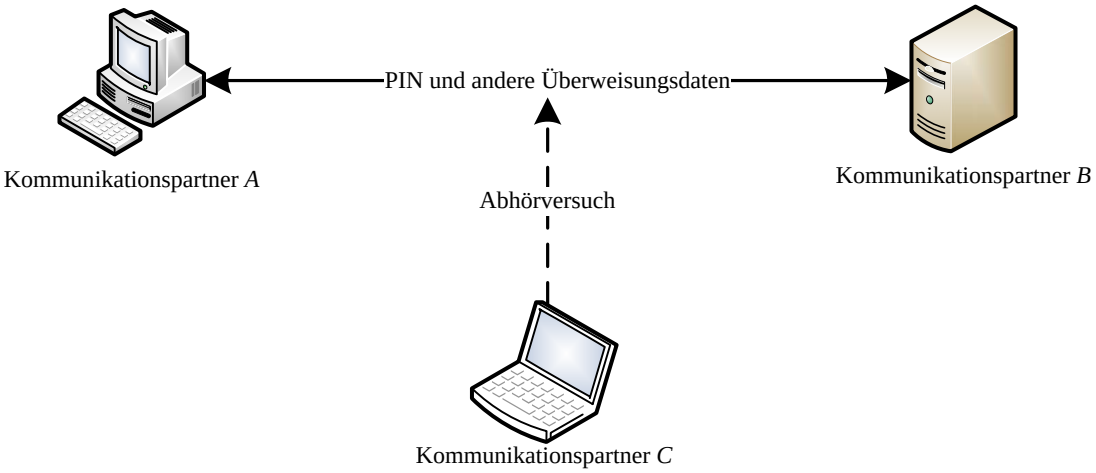

Abbildung 2.2.: Abhörversuch bei einer vertraulichen Kommunikation

Steganographie anwenden, das heißt, man verheimlicht die Existenz der Nachricht selbst, indem man zum Beispiel Geheimtinte benutzt oder die Nachricht in eine Zeichnung mit einfließen lässt.

Da in dem Beispiel die Daten per Internet verschickt werden und man aufgrund des im Internet verwendeten TCP/IP[2]-Protokolls keine Möglichkeit hat, zu beeinflussen, wie die Daten zu ihrem Ziel kommen, scheiden diese Möglichkeiten aus.

- **Kryptographische Maßnahmen:** Man versucht die Nachricht so zu verändern, dass zwar der Empfänger die Nachricht leicht lesen, aber ein Dritter, der die Nachricht abfängt, nichts damit anfangen kann.

Die Authentifizierung: *Wie kann B feststellen, dass die Überweisung von A kommt und nicht, wie in Abbildung 2.3 angedeutet, von C? Oder wie kann ich mich als A gegenüber B zweifelsfrei ausweisen?*

Es gibt zwei Arten von Authentifizierungen: Die *Teilnehmerauthentifikation*, bei der es um den Nachweis der Identität einer Person geht und die *Nachrichtenauthentifikation*, deren Ziel ist, den Ursprung einer Nachricht zu ermitteln und eventuelle Änderungen zu erkennen.

Teilnehmerauthentifikation: Das einfachste Mittel, um herauszufinden, dass die Nachricht von A stammt, ist, A nachweisen zu lassen, dass er etwas hat, was andere nicht haben, wie zum Beispiel seine PIN[3] oder ein *Zertifikat*[4]. Weitere Möglichkeiten wären, Fingerabdrücke, Stimmanalyse oder Irisscan zu verwenden. Es muss jedoch bei jedem Verfahren die Eindeutigkeit des Ergebnisses garantiert werden können.

[2] Transmission Control Protocol/Internet Protocol
[3] Persönliche Identifikationsnummer
[4] vgl. Kapitel 5.3.1

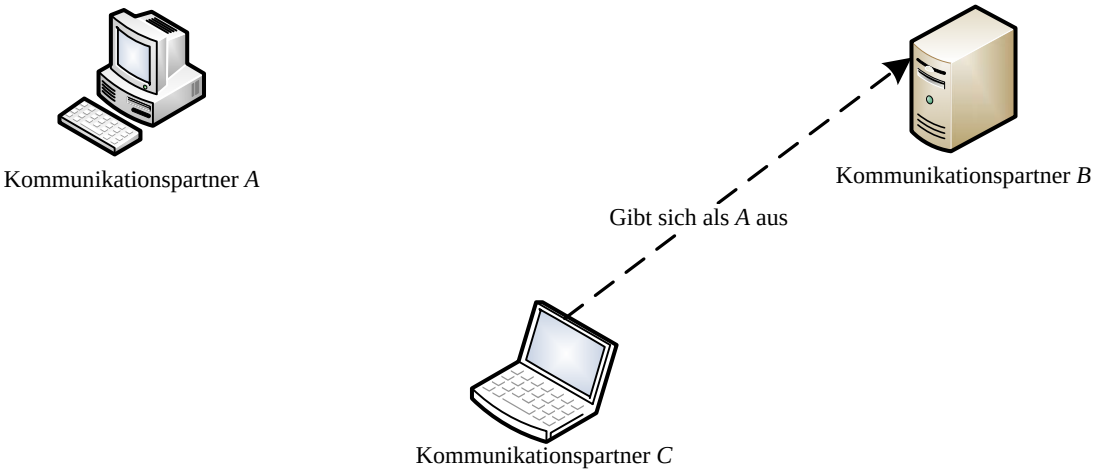

Abbildung 2.3.: Versuch, bei einer Kommunikation eine falsche Identität anzugeben

Nachrichtenauthentifikation: Da bei unserem Beispiel die Gefahr droht, dass C die Überweisungsdaten manipuliert und einen ganz anderen Betrag als den Verkaufspreis abbuchen will, muss B mögliche Manipulationen der Nachricht erkennen können. Dafür muss A die Nachricht mit einer Unterschrift oder *Signatur*[5] versehen, wobei es wichtig ist, dass man diese als ungültig erkennt, wenn die Nachricht im Nachhinein geändert wurde. Bei normal funktionierenden Kartenlesegeräten sollten üblicherweise passende Signaturen automatisch hinzugefügt werden.

Eine Lösung für die oben genannten Probleme bereit zu stellen, ist die Aufgabe der *Kryptographie*, die den ersten und älteren Zweig der *Kryptologie* bildet. Die in dem Beispiel 2.1 angeschnittenen Ziele werden in folgender Definition noch einmal zusammengefasst und erweitert:

Definition 2.2 (Ziele der Kryptographie)**.** [6] Die moderne Kryptographie hat vier Hauptziele zum Schutz von Informationen:

1. *Vertraulichkeit / Zugriffsschutz:* Nur dazu berechtigte Personen sollen in der Lage sein, die Daten oder die Nachricht zu lesen oder Informationen über ihren Inhalt zu erlangen.

2. *Integrität / Änderungsschutz:* Der Empfänger soll in der Lage sein festzustellen, ob die Daten oder die Nachricht nach ihrer Erzeugung verändert wurden.

3. *Authentizität / Fälschungsschutz:* Der Urheber der Daten oder der Absender der Nachricht soll eindeutig identifizierbar sein, und seine Urheberschaft sollte nachprüfbar sein.

[5] vgl. Kapitel 5.3.2
[6] aus [Wik09a]

4. *Verbindlichkeit / Nichtabstreitbarkeit:* Der Urheber der Daten oder Absender einer Nachricht soll nicht in der Lage sein, seine Urheberschaft zu bestreiten, das heißt, sie sollte sich gegenüber Dritten nachweisen lassen.

Kryptographische Verfahren und Systeme dienen nicht notwendigerweise allen genannten Zielen.

Der zweite und jüngere Zweig der *Kryptologie* ist die *Kryptoanalyse*. Sie hat sich erst im Ersten Weltkrieg herauskristallisiert. Davor war die *Kryptoanalyse* ein Teilgebiet der *Kryptographie*. Um bei Beispiel 2.1 zu bleiben, kann man die Kommunikation aus Sicht des Dritten C betrachten. Unter der Annahme, dass C kriminelle Absichten hat, sieht das Problem wie folgt aus:

Brechen eines Kryptosystems: *Wie kann ich die Kommunikation zwischen A und B entschlüsseln oder wie kann ich die Nachrichten modifizieren?*

Wenn das verwendete Kryptosystem eine Schwachstelle hat, ist diese Frage einfach zu beantworten: Man nutzt diese Schwachstelle zum eigenen Vorteil aus. Das viel größere Problem ist jedoch, mögliche Schwachstellen zu finden. Dazu muss man erst einmal herausfinden, welches Kryptosystem verwendet wird und dann muss dieses analysiert werden.

Somit ergibt sich für die *Kryptoanalyse* folgende Definition:

Definition 2.3 (Ziele der Kryptoanalyse)**.** Die Kryptoanalyse hat zwei Hauptziele:

1. *Qualitätsanalyse:* Die Analyse und Bewertung kryptographischer Verfahren und Systeme, mit dem Ziel, Schwachstellen zu finden oder die Sicherheit nachzuweisen.
2. *Brechen geheimer Nachrichten:* Unbefugtes Entziffern einer verschlüsselten Nachricht ohne Kenntnis des verwendeten Schlüssels.
3. *Brechen eines Kryptosystems:* Entwickeln von Methoden, weitestgehend alle verschlüsselten Nachrichten eines kryptographischen Systems entziffern zu können. Oftmals ist dies beschränkt auf gewisse Sicherheitsstufen des Systems.

Im Folgenden werde ich noch ein paar kleinere Begriffsklarstellungen geben, auf die ich in der folgenden Arbeit immer wieder zurückgreifen werde.

Definition 2.4.

- Der *Klartext* ist die ursprüngliche unverschlüsselte Nachricht und der *Geheimtext* die Nachricht, die nach dem Durchlaufen einer kryptographischen Funktion entsteht.
- Der Klartext und der Geheimtext bestehen aus *Buchstaben* oder *Zeichen*. Ich werde diese beiden Begriffe synonym verwenden, da *Buchstaben* auch nur *Zeichen* eines speziellen Alphabets sind.

- Ein *Alphabet* ist eine geordnete Menge von Zeichen. Als *Alphabet* wird hier in der Arbeit zumeist das deutsche Standardalphabet mit 26 Buchstaben verwendet, Ausnahmen werden explizit kenntlich gemacht.

Definition 2.5. [7] Ein *Kryptosystem* ist ein Fünftupel $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, das die folgenden Bedingungen erfüllt:

1. $\mathcal{P}$ ist eine endliche Menge möglicher Klartexte.
2. $\mathcal{C}$ ist eine endliche Menge möglicher Geheimtexte.
3. $\mathcal{K}$, der *Schlüsselraum*, ist eine endliche Menge möglicher Schlüssel.
4. Für jedes $K \in \mathcal{K}$ gibt es eine Verschlüsselung $e_K \in \mathcal{E}$ und eine zugehörige Entschlüsselung $d_K \in \mathcal{D}$. Für alle Funktionen $e_K : \mathcal{P} \to \mathcal{C}$ und $d_K : \mathcal{C} \to \mathcal{P}$ und für jeden Klartext $x \in \mathcal{P}$ gilt: $d_K(e_K(x)) = x$.

Weiterhin definieren wir zwei Sicherheitsprinzipien, bei denen es um die Veröffentlichung des verwendeten Kryptosystems geht.

Definition 2.6. 1883 stellte Auguste Kerckhoffs in seiner Arbeit[8] den Grundsatz auf: „Die Sicherheit eines Kryptosystems darf nicht von der Geheimhaltung des Algorithmus abhängen. Die Sicherheit gründet sich nur auf die Geheimhaltung des Schlüssels." Diesen Grundsatz bezeichnet man als *Kerckhoffs'sches Prinzip*.

Definition 2.7. Wenn man einen möglichen Angreifer im Unklaren darüber lässt, welches Kryptosystem verwendet wird, erreicht man möglicherweise einen Sicherheitsgewinn. Dieses als *Security through obscurity* benannte Prinzip basiert auf der Annahme, dass der Angreifer, bevor er versuchen kann, die Nachricht zu entschlüsseln, erst einmal das verwendete Kryptosystem erraten muss und somit wertvolle Ressourcen gebunden werden.

Bemerkung 2.8. Auf den ersten Blick mögen diese beiden Prinzipien sehr widersprüchlich sein. Jedoch hat es sich bisher als äußerst hilfreich erwiesen, neue Kryptosysteme erst einmal nach dem *Kerckhoffs'schen Prinzip* der Öffentlichkeit vorzustellen und somit zu hoffen, dass möglichst viele Leute versuchen, mithilfe unterschiedlicher Methoden Schwachstellen aufzudecken, wie es beispielsweise auch bei dem hier in dieser Arbeit betrachteten NTRU-Verfahren war. *Security through obscurity* wiederum basiert darauf, dass beide Parteien, also derjenige, der verschlüsselt, und der, der entschlüsselt, erst einmal unter strengster Geheimhaltung ausmachen, welches Kryptosystem verwendet wird. Die großen in- und ausländischen Geheimdienste zum Beispiel veröffentlichen nur sehr wenig über die Arbeit ihrer Kryptoabteilungen, daher ist zu vermuten, dass diese *Security through obscurity* als eines ihrer Sicherheitsprinzipien betrachten. Aber außerhalb von solchen in sich geschlossenen Systemen ist es sehr schwierig bis unmöglich, vorher eine solche geheime Abmachung zu treffen.

[7] vgl. [Sti95]
[8] [Ker83]

Bemerkung 2.9. In allen Beispielen, in denen ich Kryptosysteme vorstelle, werde ich bewusst Groß- und Kleinschreibung verwenden und auch Leer- und Satzzeichen benutzen. In der Standardliteratur wird auf so etwas gerne verzichtet, da dies in der Praxis eine Entschlüsselung deutlich vereinfachen würde. Da hier jedoch nur die Prinzipien der einzelnen Verfahren vorgestellt werden sollen, lege ich mehr Wert auf die Übersichtlichkeit. In der Praxis wird daher bei den klassischen Kryptosystemen auf Groß- und Kleinschreibung verzichtet und Leerzeichen werden üblicherweise durch andere Zeichen ersetzt, wie zum Beispiel durch ein **x**. Bei modernen Kryptosystemen ist eine statistische Analyse praktisch kaum noch durchführbar. Deshalb bringt ein Verstecken oder Verändern bestimmter Zeichen vor der Verschlüsselung keinen zusätzlichen Sicherheitsgewinn und ist daher nicht mehr notwendig.

3. Geschichte der Kryptographie

Die Geschichte der Kryptographie kann man in drei Abschnitte einteilen:

1. Verschlüsselung per Hand (Antike bis kurz nach dem Ersten Weltkrieg)
2. Verschlüsselung mithilfe von Maschinen (bis etwa 1970)
3. Computergestützte Verschlüsselung (bis heute)

3.1. Verschlüsselung per Hand

Die Geschichte der Kryptographie begann spätestens um etwa 2000 vor Christus in Ägypten. Aus dieser Zeit fand man veränderte Hieroglyphen, die vermutlich dafür benutzt wurden, um das öffentliche Aussprechen von religiösen Inhalten zu unterbinden. Weitere veränderte Hieroglyphen wurden bei Grabinschriften von Königen gefunden, die deren Leben erzählen sollten. Scheinbar bestand hierbei jedoch nicht die Absicht, Informationen zu verschlüsseln, sondern man vermutet, dass diese Änderungen die Texte majestätischer oder würdevoller erscheinen lassen sollten. Bis heute ist der genaue Hintergund aber nicht vollständig erforscht.[1]

In den meisten anderen antiken Hochkulturen entsann man ebenfalls verschiedene Kryptosysteme. Diese waren oftmals auch eine Kombination aus kryptographischen Methoden und Methoden der Steganographie. So hat man nicht nur eines der folgenden Verschlüsselungsverfahren verwendet, sondern man versuchte zusätzlich die Nachricht selbst zu verbergen.

Beispiel 3.1. In Sparta verwendete man die *Skytale*[2], um Nachrichten zu verschlüsseln. Die verschlüsselte Nachricht selbst war dann teilweise auf einem langen Streifen Leder geschrieben, den der Bote ohne Probleme so als Gürtel verwenden konnte, dass die Schrift nach innen zeigt. Wurde der Bote dann von einem Angreifer abgefangen, so konnte dieser die Nachricht nicht leicht auffinden, so dass der Bote unter Umständen weiterziehen konnte.

- In Indien setzte man einfache Ersetzungsalgorithmen ein, die ähnlich zu *Pig Latin* sind. Bei diesem Verfahren wird bei jedem Wort der möglicherweise vorhandenen Initialkonsonant an das Ende des Wortes gestellt. Sind mehrere Initiakonstanten vorhanden, so werden all diese auch an das Ende des Wortes gestellt. Anscließend ergänzt man die Buchstaben *ay*.

[1] vgl. [Coh87]
[2] vgl. Seite 10

Klartext	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Geheimtext	z	y	x	w	v	u	t	s	r	q	p	o	n	m	l	k	j	i	h	g	f	e	d	c	b	a

Klartext	ת	ש	ר	ק	צ ץ	פ ף	ע	ס	נ ן	מ ם	ל	כ ך	י	ט	ח	ז	ו	ה	ד	ג	ב	א
Geheimtext	א	ב	ג	ד	ה	ו	ז	ח	ט	י	כ ך	ל	מ ם	נ ן	ס	ע	פ ף	צ ץ	ק	ר	ש	ת

Abbildung 3.1.: Atbash-Tabellen für das deutsche und das hebräische Alphabet

Beispiel 3.2.

Klartext: Sicherheit ist das Hauptziel der Kryptologie!

Geheimtext: icherheitSay istay asday auptzielHay erday ologieKryptay!

Um den Geheimtext entschlüsseln zu können, muss man zuerst die ergänzten Endungen *ay* bei allen Wörtern entfernen. Anschließend muss man Wort für Wort durchgehen und schauen, ob am Ende des Wortes Koeffizienten stehen. Ist dies der Fall, muss man entscheiden, ob diese, möglicherweise nur teilweise, an den Anfang des Wortes geschrieben werden müssen, um die Nachricht korrekt zu dechiffrieren. Das Verfahren ist jedoch nicht immer eindeutig, da beispielsweise für den Geheimtext *amenay* die Klartextwörter *Amen* und *Name* möglich sind.

- Die Hebräer ersetzten den ersten Buchstaben des Alphabets mit dem letzten, den zweiten mit dem vorletzten und so weiter. Diese Methode ist auch unter dem Namen *Atbash* bekannt[3] und findet auch in der Bibel im Alten Testament Verwendung.[4] Aus welchem Grund einzelne Worte in der Bibel veschlüsselt wurden, ist aber bis heute nicht geklärt. Abbildung 3.1 zeigt beispielhaft sowohl die Atbash-Tabelle für das deutsche als auch für das hebräische Alphabet.

  Beispiel 3.3.

  Klartext: Sicherheit ist das Hauptziel der Kryptologie!

  Geheimtext: Hrxsvisvrg rhg wzh Szfkgarvo wvi Pibkgloltrv!

  Um den Text zu entschlüsseln, muss man einfach ein weiteres Mal die Atbash-Verschlüsselung anwenden.

- In Sparta benutzte man, um Nachrichten geheim zu halten, eine *Skytale*, wie sie in Abbildung 3.2 zu sehen ist. Dazu nahm man einen Holzstab mit einem bestimmten Durchmesser, um den ein dünner Streifen Papyrus oder Leder gewickelt war. Nachdem die Nachricht horizontal auf das Papyrus oder Leder geschrieben wurde, nahm man den Streifen wieder vom Stab ab, so dass darauf nur noch ein Buchstabensalat zu lesen war. Um die Botschaft wieder entziffern zu können, brauchte man ebenfalls einen Stab mit demselben Durchmesser. Wickelte man den Nachrichtenstreifen auf den Stab, so konnte man die Nachricht genauso lesen, wie sie auch geschrieben wurde.

[3] vgl. [Sin06]

[4] vgl. [DKSW04]: In Jer. 25,26 und Jer. 51,41 ist jeweils von *Scheschach* die Rede, was unverschlüsselt *Babel* beziehungsweise *Babylon* bedeutet.

Abbildung 3.2.: Eine unbeschriebene Skytale

Beispiel 3.4.

Klartext: Sicherheit ist das Hauptziel der Kryptologie!

Geheimtext: Siaz oitsiKgc erihiHlyeesa p!rtudt h peo edtrl

Da ein potentieller Angreifer nicht weiß, wie dick die verwendete Skytale ist, müsste er jetzt alle Möglichkeiten durchprobieren. Da dies jedoch bei längeren Nachrichten fast unmöglich ist, muss er Kryptoanalyse betreiben, die im Abschnitt 4 vorgestellt wird. Eine Möglichkeit, die Geheimtextnachricht übersichtlicher aufzuschreiben, ist, den Geheimtext in Spalten zu schreiben, wobei die Zeilenanzahl der Dicke der Skytale entspricht, um den Effekt der Skytale zu simulieren. Ein einfaches nochmaliges Verschlüsseln erzielt im Gegensatz zur Atbash-Verschlüsselung (vgl. Beispiel 3.3) nicht den gewünschten Effekt.

S		s	c	r	H	e	p	u	h	o	t
i	o	i		i	l	s	!	d			r
a	i	K		h	y	a	r	t	p	e	l
z	t	g	e	i	e		t		e	d	

Versuchte Entschlüsselung mit 4 Zeilen

S	i	c	h	e	r	h	e
i	t		i	s	t		d
a	s		H	a	u	p	t
z	i	e	l		d	e	r
	K	r	y	p	t	o	l
o	g	i	e	!			

Richtige Entschlüsselung mit 6 Zeilen

- In Rom wurde unter anderem die *Caesar-Verschlüsselung* verwendet. Bei dieser wird das Alphabet um eine gewisse Anzahl von Buchstaben verschoben. Für das deutsche Alphabet gibt es dafür 26 Möglichkeiten, die alle in Abbildung 3.3 aufgezeigt sind. Dabei entspricht bei einer Verschiebung um Null Buchstaben der Klartext dem Geheimtext, ebenso bei der Verschiebung um 26 Buchstaben. Somit muss

Klartext	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Geheimtext	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a
	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b
	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d
	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g
	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h
	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j
	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m
	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x
	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

Abbildung 3.3.: Alle möglichen Verschiebungen des deutschen Alphabets für die Caesar-Verschlüsselung

die Verschiebung modulo 26 betrachtet werden.

Beispiel 3.5.

Klartext: Sicherheit ist das Hauptziel der Kryptologie!

Verschiebung: $u = 14$

Geheimtext: Gwqvsfvswh wgh rog Voidhnwsz rsf Yfmdhczcuws!

Ähnlich wie auch schon bei der Skytale[5] muss ein potentieller Angreifer wissen, um wie viele Buchstaben das Alphabet verschoben wurde. Im Gegensatz zur Skytale ist hier aber die Anzahl der Möglichkeiten überschaubar, da sie nur von der Länge des Alphabets und nicht von der Länge der Nachricht abhängt. Der Empfänger aber weiß die Anzahl u und muss nun den Caesar-Algorithmus noch einmal mit der Verschiebung um $26-u \pmod{26}$ durchführen, wenn das verwendete Alphabet

[5] vgl. Beispiel 3.4

26 Buchstaben hat. In diesem Fall ist die Verschiebungsanzahl $u = 14$ und zum Entschlüsseln muss man den Geheimtext ein weiteres Mal mit $u' = 12$ verschlüsseln.

Wie man unschwer erkennen kann, waren sowohl das Caesar-Verfahren als auch das Skytale-Verfahren in der Sicherheit sehr weit fortgeschritten. Einem potentiellen Angreifer nutzte das alleinige Wissen um das verwendete Kryptosystem nicht viel, wenn er nicht auch den verwendeten Schlüssel kannte. Somit war auch in vielen Fällen eine sichere Identifizierung des Absenders möglich.

Beispiel 3.6 (Standardbeispiel des Feldherrn in der Schlacht). Der Feldherr und einer seiner Zenturionen beschließen vor einer Schlacht, das Caesar-Verfahren mit dem Geheimschlüssel $u = 6$ zu verwenden. Während der Schlacht treffen nun beim Zenturio zwei Nachrichten ein:

1. **Gtmxoll yzgxzkt!**
2. **Rexizww jkfggve!**

Da beide Boten nicht mehr aufzufinden sind, weiß er nicht, welche Nachricht von seinem Feldherrn kommt und welche eventuell vom Feind. Da aber das Caesar-Verfahren sowohl Verschlüsselung als auch Signierung ist, braucht der Zenturio nur die Nachrichten mit dem richtigen Geheimschlüssel $u' = 26 - u = 26 - 6 = 20$ zu entschlüsseln. Wenn die Nachricht vom richtigen Absender kommt, wird sie dadurch lesbar. Nach dem Entschlüsseln kommen folgende Nachrichten zum Vorschein:

1. **Angriff starten!**
2. **Lyrctqq dezaapy!**

Somit ist die vom Feind abgeschickte (zweite) Nachricht **Angriff stoppen!** als falsch identifiziert worden, da als Geheimschlüssel $u_0 = 17$ verwendete wurde.

Die oben genannten Verfahren sind die wichtigsten in der Antike benutzten Kryptosysteme. Sie lassen sich in zwei verschiedene Verschlüsselungsverfahren, die *monoalphabetische Substitution* und die *Transposition*, einteilen.

Definition 3.7. Eine *monoalphabetische Substitution* oder *einfache Substitution* ist eine injektive Abbildung $g : V \to W$, wobei V die Menge der Klartextzeichen und W die Menge der Geheimzeichen ist. Dabei müssen V und W nicht notwendigerweise gleiche Mächtigkeit haben, die Mächtigkeit von W darf aber nie kleiner sein als die Mächtigkeit von V, damit eine eindeutige Entschlüsselung möglich wird. Wird jedes Zeichen aus V durch genau ein Zeichen in W ersetzt, so spricht man von einer *monopartiten einfachen Substitution*. Ebenso kann man auch von einer *bipartiten* (bei zwei Zeichen aus W), *tripartiten* (bei drei Zeichen aus W) usw. *einfachen Substitution* sprechen.

Die Sicherheit der *monoalphabetischen Substitution* kommt auf die Mächtigkeit der verwendeten Mengen V und W an. Ist beispielsweise $V = W$ gleich dem deutschen Alphabet, so ist $|V| = |W| = 26$. Dann gibt es für die Substitution des ersten Buchstabens **a** maximal 26 Möglichkeiten, für den zweiten Buchstaben **b** noch 25, und so weiter,

so dass am Ende die maximale Anzahl der verschiedenen Möglichkeiten $26! \approx 4 \cdot 10^{26}$ entspricht. Damit ist ein reines Ausprobieren, welcher Buchstabe welchem Zeichen entspricht, praktisch unmöglich. Im Abschnitt 4 werden aber Verfahren vorgestellt, die diese große Anzahl von Möglichkeiten unter Kontrolle bekommen.[6]

Bemerkung 3.8. Das Caesar-Verfahren und das Atbash-Verfahren verwenden monoalphabetische Substitution. Da beim Atbash-Verfahren die Zuordnung fest vorgeschrieben ist, bleibt von den 26! Möglichkeiten nur eine übrig. Das Caesar-Verfahren ist dabei schon sicherer. Es bietet 26 Möglichkeiten einer Verschlüsselung, wovon aber nur 25 sinnvoll sind, da bei einer Verschiebung mit dem Schlüssel $u = 0$ der Geheim- und der Klartext identisch wären.

Definition 3.9. Unter *Transposition* versteht man in der Kryptologie ein Verfahren, dass bei einem Klartext die Zeichen zwar bestehen lässt, dafür aber die Position verändert. Also eine Abbildung $g : X_n \to X_n$ mit $g(x_1, x_2, \ldots, x_n) = (x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$ für eine Permutation $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$.

Die Sicherheit einer *Transposition* steigt mit Länge des Klartextes an. Wenn n die Anzahl der Zeichen im Klartext ist, so existieren maximal $n!$ Möglichkeiten für die *Transposition.*

Bemerkung 3.10. Das Skytale-Verfahren verwendet Transposition, wobei die Permutationsvorschrift vom Durchmesser des verwendeten Stabes abhängt.

Bemerkung 3.11. Hat man einen mit einer beliebigen Transposition erstellten Geheimtext, bei dem beispielsweise die Buchstaben des Klartextes einfach in alphabetischer Reihenfolge sortiert wurden, so ist es fast unmöglich, den Klartext daraus zu extrahieren, da das Ergebnis im Allgemeinen nicht eindeutig wäre. Somit war man in der Lage, den Geheimtext zu veröffentlichen, ohne Gefahr zu laufen, dass jemand die wirkliche Botschaft missbraucht. Zu gegebener Zeit kann man dann die Botschaft verbreiten und durch nochmalige Anwendung der Transposition beweisen, dass man der ursprüngliche Urheber war.[7]

Historische Beispiele dafür sind:[8]

- Galileo Galilei veröffentlichte seine wissenschaftliche Erkenntnis *Cynthiae figuras aemulatur Mater Amorum* unter: **haec immatura a me jam frustra leguntur oy**.
- Christiaan Huygens veröffentlichte statt des ursprünglichen Satzes *Annulo cingitur, tenui plano, nusquam cohaerente, ad eclipticam inclinato* nur die sortierten Buchstabenreihe: **aaaaaaa ccccc d eeeee g h iiiiiii llll mm nnnnnnnnn oooo pp q rr s ttttt uuuuu**.

[6] vgl. [Bau95]
[7] vgl. Fragestellungen zu Beispiel 2.1
[8] vgl. [Kip99]

Abbildung 3.4.: Chiffrierscheibe

Definition 3.12. [9] Eine *Einwegfunktion* ist in der Kryptographie eine Funktion $f : X \to Y$, bei der $f(x)$ für alle $x \in X$ „einfach“ berechnet werden kann, aber für „weitestgehend alle“ $y \in \mathrm{Im}(X)$ es praktisch unmöglich sein soll, ein $x \in X$ zu finden, so dass $f(x) = y$.

Bemerkung 3.13. Wenn man die Definition einer Einwegfunktion ansieht und sie mit der Definition eines Kryptosystems[10] vergleicht, so fällt auf, dass die Einwegfunktion die vierte Bedingung nicht erfüllt. Somit ist sie strenggenommen *kein* Kryptosystem. In der Praxis wird sie aber gerne für die Authentifizierung[11] verwendet, zum Beispiel, wenn man bei einer Passwortabfrage nicht sein richtiges Passwort angeben will, sondern nur beweisen will, dass man es kennt.

Nach dem Untergang des römischen Reiches gab es im europäischen Kulturkreis nur wenige kryptographische Neuerungen.[12] Die einzige Abhandlung über Kryptographie, die aus dieser Zeit überliefert ist, wurde vom englischen Mönch und Universalgelehrten Roger Bacon verfasst. Darin zählte er sieben Verschlüsselungsmethoden auf, darunter das Weglassen von Vokalen und die Verwendung eines unbekannten Alphabets.

Wirklich entscheidende Neuerungen brachten erst die Entwicklung von Chiffriermaschinen, wie die Chiffrierscheibe, die Leon Battista Alberti um 1470 entwickelte. Diese bestand aus einer äußeren Scheibe, auf der die Geheimtextbuchstaben aufgedruckt waren, und einer beweglichen inneren Scheibe, deren Aufdruck aus dem Klartextalphabet bestand. Stellte man nun die Scheibe einmal auf ein bestimmtes Schlüsselpaar ein, zum Beispiel $g \to A$, so kann man leicht einen Klartext mithilfe einer monoalphabetischen Verschlüsselung chiffrieren oder einen Geheimtext entschlüsseln. Zusätzlich kann man die Stellung der Scheiben zueinander während der Verschlüsselung verändern, so dass sogar

[9] vgl. [MVOV97]
[10] vgl. Definition 2.5
[11] vgl. Beispiel 2.1
[12] vgl. [Kah96]

eine *polyalphabetische Verschlüsselung* möglich wird.

Definition 3.14. Eine *polyalphabetische Verschlüsselung* oder *polyalphabetische Substitution* verschlüsselt einen Klartext in einen Geheimtext unter Zuhilfenahme mehrerer Alphabete. Somit gibt es, im Gegensatz zur monoalphabetischen Verschlüsselung, keine eindeutige Entsprechung von Geheimtextzeichen und Klartextzeichen.

Beispiel 3.15. Man wendet den *Caesar-Algorithmus* auf den Klartext **Sicherheit ist das Hauptziel der Kryptologie!** so an, dass jeder Buchstabe mit einem anderen Schlüssel verschlüsselt wird. Dafür kann man mit dem Schlüssel $u = 0$ beginnen und nach jedem Buchstaben den Schlüssel um Eins erhöhen. Dann erhält man den Geheimtext **Sjekiwnlqc sdf qoh Xrminuebj ces Mucuzvtxqtq!**. Bei diesem ist offensichtlich, dass kein Buchstabe des Geheimtextes sich eindeutig einem Buchstaben des Klartextes zuordnen lässt, wie zum Beispiel der Geheimtextbuchstabe *q*, der im Klartext den Buchstaben *i,d,g* und *e* entspricht oder der Klartextbuchstabe *e*, der im Geheimtext den Buchstaben *i*, *l*, *b*, *e* und *q* entspricht.

Das Prinzip der polyalphabetischen Verschlüsselung machte sich auch die *Vigenère-Verschlüsselung* zunutze.[13] Dabei wird anfänglich ein Schlüsselwort gewählt. Hat der Klartext nun k Zeichen, so muss man für den Schlüssel das Schlüsselwort auch auf k Zeichen kürzen oder verlängern. Nachdem das Kürzen des Schlüsselworts trivial ist, betrachte ich hier nur den Fall des Verlängerns. Dafür gibt es zwei Möglichkeiten:

Anreihung des Schlüssels: Das Schlüsselwort wird so oft wiederholt, bis der Schlüssel k Zeichen enthält.

Beispiel. Der Klartext lautet **Geheimnis** und das Schlüsselwort **key**. Dann ist der zu verwendende Schlüssel **keykeykey**

Autokey-Vigenère-Verschlüsselung: Der Schlüssel besteht in den ersten Zeichen aus dem Schlüsselwort. Danach wird der Klartext angehängt.

Beispiel. Der Klartext lautet **Geheimnis** und das Schlüsselwort **key**. Dann ist der zu verwendende Schlüssel **keygeheim**

Dann benutzt man das *Vigenère-Quadrat*, mit dessen Hilfe man den Klartext verschlüsselt. Dazu geht man Zeichen für Zeichen vor und ersetzt den jeweiligen Klartextbuchstaben durch das Zeichen, dass durch die Verschiebung mit dem Schlüsselzeichen entsteht. Ist zum Beispiel der Klartext **Geheimnis** und der Schlüssel **keykeykey**, dann wird der Buchstabe **G** durch den Buchstaben **R** ersetzt, da die Verschlüsselung mit dem Buchstaben **K** einer Verschiebung um elf Zeichen entspricht oder im *Vigenère-Quadrat*: Man sucht die Position des Buchstabens **K** im Alphabet (11. Buchstabe), wobei diese Zahl der Verschiebung des Klartextalphabets entspricht. Dann sucht man in der ersten Zeile nach dem Buchstaben **G** (7. Spalte). Das Zeichen für den Geheimtext findet man nun in der siebten Spalte in der Zeile, die für die Verschiebung um elf Buchstaben steht.

Bemerkung 3.16. Somit entspricht die Vigenère-Verschlüsselung in jedem Zeichen dem Anwenden des Caesar-Verfahrens.

[13] vgl. [Kri08]

Verschiebung																									
0	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
1	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a
2	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b
3	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
4	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d
5	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
6	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
7	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g
8	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h
9	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
10	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j
11	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
12	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
13	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m
14	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
15	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
16	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
17	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
18	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
19	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
20	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
21	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
22	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
23	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
24	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x
25	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

Abbildung 3.5.: Vigenère-Quadrat für das deutsche Alphabet

Diese Verschlüsselung war etwa 300 Jahre ungebrochen und wurde deshalb (fast) zu Recht als „*Le Chiffre indéchiffrable*“ angesehen. Eine Variation davon ist jedoch selbst heute unentzifferbar: der *One-Time-Pad*. Da die Sicherheit der Vigenère-Verschlüsselung von der Länge und der Beschaffenheit des verwendeten Schlüsselwortes abhängt, ist eine perfekte Sicherheit[14] gegeben, wenn das Schlüsselwort genau so lange ist, wie der Klartext, wenn das Schlüsselwort aus zufälligen Zeichen besteht und wenn für jede Nachricht ein neuer zufälliger Schlüssel verwendet wird. Durch seine Sicherheit wird der One-Time-Pad deshalb gerne von Spionen verwendet, wie in Abbildung 3.6 illustriert, oder ist Teil der Implementation des „heißen Drahtes“ zwischen Moskau und Washington.[15]

[14] laut [Sha49]
[15] vgl. [KNM03]

Abbildung 3.6.: One-Time-Pad-Schablone in einer Nussschale

3.2. Verschlüsselung mit Hilfe von Maschinen

Nach dem Ersten Weltkrieg und mit der Entwicklung elektrischer Schreibmaschinen wurden an vier Stellen unabhängig voneinander[16] *Rotor-Chiffriermaschinen* entwickelt, die eine maschinelle Ver- und Entschlüsselung mit Hilfe von Rotoren oder Walzen ermöglichten und somit eine polyalphabetische Verschlüsselung vereinfachten. Die bekannteste davon ist die vom Deutschen Reich während des Zweiten Weltkriegs verwendete und 1918 von Arthur Scherbius zum Patent angemeldete und in Abbildung 3.7 zu sehende *ENIGMA*. Diese besteht aus einer Tastatur, einem Walzensatz und einem Lampenfeld zur Anzeige.

Der *Walzensatz* ist das Herzstück der Verschlüsselung. Die darin liegenden Walzen sind drehbar angeordnet und weisen für jeden Buchstaben des Alphabets je einen elektrischen Kontakt auf jeder Seite auf, die durch Drähte im Inneren der Walze paarweise und unregelmäßig miteinander verbunden sind. Drückt man nun eine Buchstabentaste, so fließt elektrischer Strom von der gedrückten Taste durch den Walzensatz und lässt eine Anzeigelampe aufleuchten. Eine Besonderheit bei der ENIGMA ist die 1926 eingeführte *Umkehrwalze*. Diese ermöglichte, dass man mit der Enigma sowohl ver- als auch entschlüsseln kann, ohne dass man irgendetwas an der Maschine verändern muss.[17] Weiterhin besitzt die ENIGMA ein *Steckerbrett* mit doppelpoligen Steckbuchsen für jeden der 26 Buchstaben. Werden nun zwei Buchstaben miteinander verdrahtet, zum Beispiel **a** mit **k**, so bewirkt das, dass beim Drücken der Taste **a** der Strom erst einmal über den Buchstaben **k** umgeleitet und erst dann in den Walzensatz geleitet wird.

In Abb. 3.8 ist symbolisch eine ENIGMA C ohne Steckerbrett abgebildet. Diese besteht aus drei Rotoren R_L, R_M und R_N und der Umkehrwalze U. Drückt man nun den Buchstaben **e**, so wird ein Stromkreis über den Stator durch die drei Rotoren und die Umkehrwalze so geschlossen, dass am Ende die Lampe beim Buchstaben **m** anfängt zu leuchten.

Damit die Verschlüsselung auch polyalphabetisch wird und nicht während der Eingabe einer Nachricht irgendetwas an der Maschine geändert werden muss, ist die ENIGMA so konzipiert, dass nach dem Loslassen einer beliebigen Buchstabentaste ein automatisches Weiterdrehen der Walzen stattfindet. Dabei wird, bei der ENIGMA mit drei Walzen,

[16] vgl. [Bau95]
[17] vgl. die Atbash-Verschlüsselung

Abbildung 3.7.: ENIGMA

zuerst die rechte Walze weitergedreht, bis eine spezielle Stellung erreicht ist. Wird in dieser Position noch eine Taste gedrückt, dann dreht zusätzlich die mittlere Walze um eine Position weiter und danach erstmal wieder nur die rechte. Diese Position wird auch die *Nutstellung* genannt. Allgemein folgt die Weiterdrehung folgender Regel:[18]

- Ist weder in der Mitte noch rechts die Nutstelle erreicht, so wird nur die rechte Walze weitergedreht.
- Ist die rechte Walze in Nutstellung, die mittlere aber nicht, so wird die mittlere und rechte Walze weitergedreht.
- Ist die mittlere Walze in Nutstellung, so werden alle drei Walzen weitergedreht.

Weiterhin sind die Walzen austauschbar und auch deren Startposition ist beliebig wählbar, so dass ein gültiger Schlüssel immer aus folgenden Angaben bestand:

- welche Umkehrwalze verwendet wird,
- welche rotierenden Walzen verwendet werden und in welcher Reihenfolge,

[18] aus [Rup01]

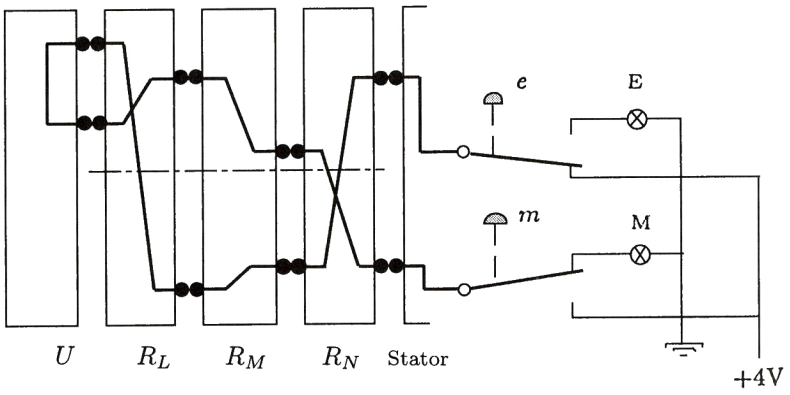

Abbildung 3.8.: Stromlauf in einer ENIGMA

- die *Ringstellung* für die rotierenden Walzen, die die Abweichung zwischen der inneren Verdrahtung der Walzen und der Nutstellung bestimmt,
- die nötigen Steckerverbindungen und
- die Grundstellung der Walzen.

Die ersten vier Angaben wurden, wie in Abbildung 3.9 zu sehen, zusammengefasst zu einem *Tagesschlüssel*, der täglich verändert wurde und für alle Nachrichten gleich war. Die letzte Angabe aber wurde von jedem Sender zufällig ausgewählt, so dass die Analyse der Geheimtexte auch bei größerer Nachrichtenzahl erschwert wird.

Der Schlüsselraum der ENIGMA ergibt sich aus den folgenden vier Faktoren:

Walzenlage Von fünf Walzen werden drei rotierende ausgewählt, sowie von zwei Umkehrwalzen eine. Dies ergibt $5! = 120$ mögliche Walzenlagen.

Ringstellung Es gibt für die drei rotierenden Walzen jeweils 26 verschiedene Ringstellungen und somit 26^3 Möglichkeiten. Da eine Weiterschaltung einer noch weiter links stehenden Walze nicht existiert, ist die Nutstellung der linken Walze irrelevant und damit die Ringstellung der linken Walze kryptographisch ohne Bedeutung. Somit bleiben $26^2 = 676$ relevante Ringstellungen.

Die Grundstellung Es gibt für jede der drei rotierende Walzen insgesamt 26 unterschiedliche Grundstellungen, die Umkehrwalze ist fest. Insgesamt sind somit $26^3 = 17.576$ Grundstellungen verfügbar.

Steckerverbindungen Es können bis zu maximal 13 Steckerverbindungen zwischen den 26 Buchstaben hergestellt werden. Für $n \leq 13$, was der Anzahl der Steckerverbindungen entspricht, gibt es

$$\frac{1}{n!}\prod_{i=1}^{n}\frac{(26-2i+2)(26-2i+1)}{2} = \frac{26!}{2^n \cdot n! \cdot (26-2n)!}$$

Möglichkeiten.

VIII. Beiſpiel.

17. Gültiger Tagesſchlüſſel:

(Ausſchnitt aus der für die Verſchlüſſelung des Klartextes in Betracht kommenden Schlüſſeltafel, z. B. ».....« Maſchinenſchlüſſel für Monat Mai«)

Datum	Walzenlage	Ringſtellung	Grundſtellung
4.	I III II	16 11 13	01 12 22

Steckerverbindung	Kenngruppen-Einſatzſtelle ..... Gruppe	Kenngruppen
CO DI FR HU JW LS TX	2	adq nuz opw vxz

Abbildung 3.9.: Tagesschlüssel der ENIGMA

Ab 1939 gab es innerhalb der Wehrmacht die Anweisung, immer genau 10 Steckverbindungen zu verwenden, so dass die Größe des Schlüsselraums der ENIGMA während des Krieges gleich dem Produkt aus 120 Walzenlagen, 676 Ringstellungen, 17.576 Grundstellungen und 150.738.274.937.250 Steckermöglichkeiten war:[19]

$$120 \cdot 676 \cdot 17.576 \cdot 150.738.274.937.250 = 214.917.374.654.501.238.720.000 \approx 2 \cdot 10^{23}$$

3.3. Computergestützte Verschlüsselung

Doch trotz dieses riesigen Schlüsselraums unter der Verwendung von polyalphabetischer Verschlüsselung wurde die ENIGMA sowohl vor als auch während des Krieges gebrochen. Deshalb versuchte man nach dem Krieg, die Kryptographie auf eine neue Grundlage zu stellen: *asymmetrische Verschlüsselung* im Gegensatz zur (klassischen) *symmetrischen Verschlüsselung*.

Definition 3.17.

- Die (klassische) *symmetrische Verschlüsselung* baut auf dem Prinzip auf, dass beim Verschlüsseln derselbe Schlüssel verwendet wird wie beim Entschlüsseln, das heißt,

[19] vgl. [Wik09b]

beide Parteien müssen sich vor der Kommunikation auf einen Schlüssel einigen oder der Schlüssel muss während einer Kommunikation mit übertragen werden.

- Bei der *asymmetrischen Verschlüsselung* wird für die Verschlüsselung ein anderer Schlüssel verwendet als für die Entschlüsselung. Diese Schlüssel heißen *Public Key* für die Verschlüsselung und *Private Key* für die Entschlüsselung. Wichtig ist dabei, dass aus dem *Public Key* der *Private Key* möglichst nicht erraten werden können sollte.

Bemerkung 3.18. Wie der Name Private Key schon sagt, wird dieser geheim gehalten. Er dient dem Zweck, ankommende verschlüsselte Nachrichten zu dechiffrieren und in vielen Verfahren benutzt man den Private Key auch dafür, den Public Key zu berechnen. Der Public Key wiederum wird veröffentlicht und ist daher für jedermann zu lesen und wird für die Verschlüsselung von Daten, zur Prüfung einer digitalen Signatur oder zur Authentifikation verwendet. Dabei muss immer der Public Key desjenigen verwendet werden, der auch die Daten dechiffrieren oder dessen Identität überprüft werden soll.[20]

Bemerkung 3.19. Das erste Kryptosystem, dass nach dem Prinzip der asymmetrischen Verschlüsselung funktioniert ist das 1977 von Ronald L. Rivest, Adi Shamir und Leonard Adleman am MIT[21] entwickelte *RSA-Verfahren*. Dieses macht sich zunutze, dass es, vor allem mit der modernen Computertechnik, einfach ist, zwei Zahlen zu multiplizieren, es aber wesentlich schwerer ist, große Zahlen zu faktorisieren.

Für das Verfahren sucht man sich zwei große Primzahlen p und q und berechnet ihr Produkt $N = pq$ sowie die *Eulersche Funktion* $\varphi(N) = (p-1) \cdot (q-1)$. Dann werden zwei Zahlen d und e gesucht, wobei gelten muss, dass $\mathrm{ggT}(e, \varphi(N)) = 1$ und $e \cdot d \equiv 1 \pmod{\varphi(N)}$ ist. Der Private Key entspricht nun den Zahlen (d, N) und der Public Key besteht aus den Zahlen (e, N), die veröffentlicht werden können. Zum Verschlüsseln der Nachricht x wird nun $r = x^e \pmod{N}$ berechnet, wobei $x < N$ sein muss, sonst muss x aufgespalten werden. Zum Entschlüsseln berechnet man $r^d = x^{ed} \pmod{N} = x$ und bekommt somit die ursprüngliche Nachricht.[22]

Bemerkung 3.20. Da die Sicherheit des RSA-Verfahrens davon abhängt, wie groß die gewählten Zahlen sind, ist die Verschlüsselung gerade von großen Datenmengen sehr rechenintensiv. Daher wird in der Praxis häufig folgendes Verfahren angewendet:

Definition 3.21. Unter einem *hybriden Verfahren* versteht man in der Kryptologie ein Verfahren, bei dem für die eigentliche Verschlüsselung einer Nachricht ein symmetrisches Verfahren benutzt wird, der Austausch des Schlüssels aber über ein asymmetrisches Verfahren abläuft.

Gerade im Internet oder bei internetbasierten Anwendungen ist es wichtig, dass jeder Kommunikationspartner jederzeit unbegrenzt mit einem anderen vertraulich kommunizieren kann, ohne dass umfangreiche Codebücher[23] ausgetauscht werden müssen. Daher

[20] vgl. [LPV05]
[21] Massachusetts Institute of Technology
[22] vgl. [Sch96] und [RSA78]
[23] vgl. Kapitel 5.2

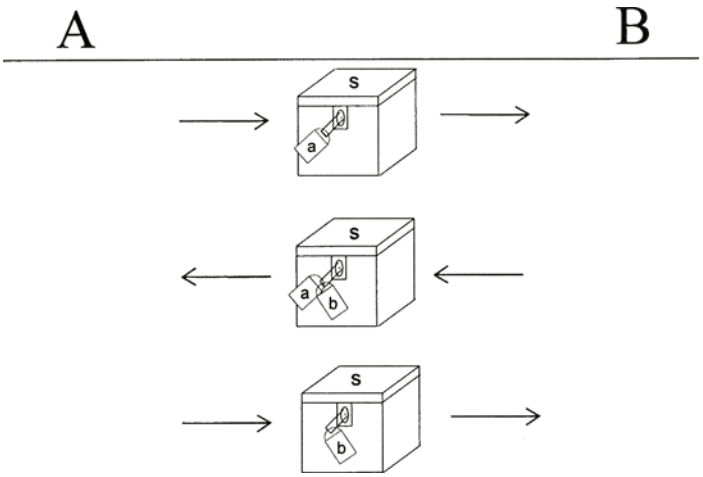

Abbildung 3.10.: Shamir's No-Key-Verfahren

ist eine reine symmetrische Verschlüsselung ohne ein hybrides Verfahren im Internet nicht durchführbar, da das Problem des sicheren und jederzeit verfügbaren Schlüsselaustausches quasi unlösbar ist.

Die einzige verbliebene Alternative zur asymmetrischen Verschlüsselung wäre das in Abbildung 3.10 skizzierte *No-Key-Protokoll* von Shamir.[24] Bei diesem braucht keiner der Kommunikationspartner A und B dem anderen seinen Schlüssel verraten, es wird nur eine genügend große Primzahl p, unter Umständen auch öffentlich, ausgemacht. Dabei muss nur sichergestellt werden, dass $\mathrm{ggT}(m, p) = 1$ gilt, mit m der zu verschlüsselnden Nachricht. Dann suchen sich A und B je eine Zahl und deren Inverse modulo $p-1$, so dass nun A die Zahlen p, a und a', mit $a \cdot a' \equiv 1 \pmod{p-1}$, besitzt und B die Zahlen p, b und b', mit $b \cdot b' \equiv 1 \pmod{p-1}$. Jetzt fängt A an und berechnet $x \equiv m^a \pmod{p}$ und schickt B die Nachricht x. B berechnet nun daraus $y \equiv x^b \pmod{p}$ und schickt A die Nachricht y zurück. Dann berechnet A den Term $z \equiv y^{a'} \pmod{p}$ und schickt diese Nachricht an B. Dieser entschlüsselt nun endgültig die Nachricht, indem er $w \equiv z^{b'} \pmod{p}$ berechnet.

Nach dem Satz von Euler gilt für alle $m < p$ und für alle $k \in \mathbb{N}_0$: $m^{k\varphi(p)+1} \equiv m \pmod{p}$. Da p ein Primzahl ist, gilt $\varphi(p) = p - 1$, und damit ist $aa' \equiv k\varphi(p) + 1$; entsprechendes gilt für bb'. Somit gilt, modulo p gerechnet,

$$w \equiv z^{b'} \equiv y^{a'b'} \equiv x^{ba'b'} \equiv m^{aba'b'} \equiv m^{aa'bb'} \equiv m,$$

das heißt, die Nachricht m kommt beim Empfänger B an, ohne dass irgendein Schlüssel geheim ausgetauscht werden musste.

Bemerkung 3.22. Das *No-Key-Protokoll* basiert im Gegensatz zum RSA-Verfahren auf dem Problem des diskreten Logarithmus. Bisher ist es weder beim diskreten Logarithmus, noch beim Problem der Faktorisierung irgendjemandem gelungen, einen polynomialen

[24] vgl. [BSW98]

Algorithmus ohne den Einsatz von Quantencomputern zur Berechnung zu finden oder zu beweisen, dass dies unmöglich ist. Solange dieser Beweis jedoch ausbleibt, sind alle Kryptosysteme, die auf diesen Problemen basieren, potentiell unsicher. Ebenso werden alle diese Kryptosysteme unsicher, sollte es eines Tages gelingen, Quantencomputer in großem Umfang herzustellen.

Bemerkung 3.23. Im Internet haben sich bisher Public Key-Kryptosysteme eher durchgesetzt als zum Beispiel das No-Key-Protokoll, das ganz ohne Schlüsselaustausch funktioniert. Womöglich liegt dies an drei Gründen:

- Das No-Key-Protokoll wurde von Shamir nicht[25] oder erst sehr spät veröffentlicht, also zu einem Zeitpunkt, zu dem Standardsicherheitsprotokolle wie IPsec oder SSL längst im Einsatz waren.
- Durch das dreimalige Verschicken der verschlüsselten Nachricht wären die analogen Internetanschlüsse, die nicht für große Datenmengen ausgelegt sind, überlastet gewesen. Weiterhin muss bei dem Verfahren jede Nachricht viermal verschlüsselt werden, was zusätzliche Rechenkapazitäten benötigt. Daher lässt sich ein Kryptosystem, das für viel Datenoverhead sorgt und zusätzlich sehr rechenintensiv ist, nur schwer integrieren. Dahingegen muss bei einem Public Key-Kryptosystem die Nachricht nur einmal verschickt werden und zusätzlich der Schlüssel, der aber im Umfang normalerweise deutlich kleiner ist. Der Rechenaufwand kann dann zusätzlich noch durch Verwendung hybrider Verfahren verringert werden.
- Schließlich ist die Sicherheit beim No-Key-Protokoll mit einem auf dem gleichen Problem basierenden Public Key-Kryptosystem vergleichbar, so dass kein wesentlicher Sicherheitsgewinn dazukommt, da sich die Sicherheit nur darauf gründet, dass es derzeit noch nicht möglich ist, den diskreten Logarithmus in Polynomialzeit zu berechnen, um damit die Nachricht im Klartext entziffern zu können.

[25] laut [BSW98]

4. Ausgewählte Methoden der Kryptoanalyse

Der wichtigste Ansatz der Kryptoanalyse ist, alle verfügbaren Informationen über das zu untersuchende Verfahren zu erlangen. Der einfachste Fall liegt vor, wenn man das verwendete Kryptosystem kennt und etwa durch gezielte Spionage den verwendeten Schlüssel in Erfahrung bringen konnte. Doch ist dieses Szenario im Allgemeinen nur Wunschdenken der Kryptoanalytiker. Im Normalfall muss man durch geschickte Analyse der abgefangenen oder mitgehörten Nachrichten Rückschlüsse auf die fehlenden Informationen ziehen.

In der klassischen Kryptoanalyse war Statistik die stärkste Waffe zur Entzifferung von Geheimtexten, da die verwendeten Algorithmen einfach genug bleiben mussten, so dass sowohl eine Chiffrierung als auch eine Entschlüsselung per Hand in vertretbarer Zeit fehlerfrei umgesetzt werden konnte. Mithilfe der Statistik werden die Häufigkeiten bestimmter Zeichen und Zeichenfolgen ermittelt, da mit dem Wissen über die Gesetzmäßigkeiten einer Sprache Buchstaben und Wörter zugeordnet werden können und der Klartext rekonstruiert werden kann.

Seitdem Computer Verschlüsselungen durchführen und diese durch ihre Geschwindigkeit und Präzision die statistischen Bindungen in einem verschlüsselten Text auf fast Null reduzieren können, steht die Kryptoanalyse vor neuen Herausforderungen. So muss man jetzt versuchen, mithilfe neuer Analysetechniken den Verschlüsselungsalgorithmus aufzudecken, eine Schwachstelle in diesem auszunutzen, wie auch schon die Statistik Schwachstellen in den klassischen Kryptosystemen nutzte, und den Schlüssel zu rekonstruieren, mit dem die Nachricht verschlüsselt wurde. Dazu versucht man häufig in verschiedenen anderen mathematischen Teilgebieten mögliche Lösungen zu finden, um komplizierte Probleme auf schon bekannte einfache Problemstellungen zu reduzieren.[1]

Im Folgenden werde ich einige Verfahren der Kryptoanalyse vorstellen.

4.1. Brute Force-Angriff

Der *Brute Force-Angriff* ist eine Möglichkeit, einen unbekannten Schlüssel zu erraten. Dafür schaut man zuerst, welcher Schlüsselraum bei dem angewandten Kryptosystem verwendet wird und probiert dann alle möglichen Schlüssel aus. Im Worst Case muss man mit dieser Methode alle Schlüssel des Schlüsselraumes durchprobieren und daher ist Brute Force im Gegensatz zu allen anderen hier vorgestellten Kryptoanalysemethoden die langsamste und scheinbar am wenigsten Erfolg versprechende. Es gibt jedoch nur ein

[1] vgl. [Wik09c]

einziges Kryptosystem, dass *nicht* mit Hilfe von Brute Force entschlüsselt werden kann: Der One-Time-Pad.[2]

Gerade in der heutigen Zeit kann man durch das Internet und Cluster-Computer Rechenkapazitäten in großer Menge sehr günstig beschaffen. Dazu verwendet man entweder einen der zahlreichen und immer besser werdenden Supercomputer[3] oder man mietet sich ein Botnetz, das zum Teil gigantische Dimensionen[4] haben kann. Hat man sich erst einmal die gewünschte Rechenkapazität gesichert, so kann man einen Brute Force-Angriff größtenteils automatisiert ablaufen lassen.

Brute Force-Angriffe auf moderne Verschlüsselungsalgorithmen bei Verwendung ausreichend langer Schlüssel sind in der Praxis aber trotzdem aussichtslos, da der erforderliche Rechenaufwand und damit Zeit- und Kostenaufwand zu groß wären. Da sich jedoch die Leistung moderner Hardware permanent verbessert und sich dadurch der Zeitaufwand für das Durchprobieren aller Schlüssel einer bestimmten Länge erheblich reduziert, muss die minimale Schlüssellänge der Kryptosysteme periodisch vergrößert werden, um weiterhin gleichbleibende Sicherheit gewährleisten zu können. Somit ist auch bei modernen Kryptosystemen die Frage der Langzeitverschlüsselung noch ungelöst.

Beispiel. Bei der Caesar-Verschlüsselung ist ein Brute Force-Angriff sehr effizient, da der Schlüsselraum nur aus 26 möglichen Kombinationen besteht. Somit würde ein Angriff so ablaufen, dass ein Angreifer zuerst den Schlüssel $u = 0$ ausprobiert und damit schaut, ob ein sinnvoller Klartext entsteht. Ist dies nicht der Fall, so werden nacheinander die nächsten Schlüssel $u = 1$, $u = 2$ und so weiter probiert, bis irgendwann ein sinnvoller Klartext entziffert werden kann.

Bemerkung 4.1. Der Brute Force-Angriff, wie eigentlich alle anderen hier vorgestellten Angriffsmethoden auch, läuft natürlich ins Leere, wenn die anfängliche Informationsgewinnung fehlerhaft war, also wenn man zum Beispiel von einem falschen Kryptosystem ausgeht und daher der Schüsselraum anders ist als erwartet, oder die Nachricht in einer anderen Sprache geschrieben ist und daher eine automatische Auswertung kein verwertbares Ergebnis bringt.

4.2. Wörterbuchangriff

Der *Wörterbuchangriff* ist eine Spezialisierung des Brute Force-Angriffs. Als Basis werden für diese Methode nicht mehr alle möglichen Schlüssel des Schlüsselraums verwendet, sondern es wird ein vorher zusammengestelltes Wörterbuch durchprobiert. Dafür wird zuerst analysiert, welcher Schlüsselraum verwendet wird. Dann wird ein Wörterbuch erstellt, in dem die wahrscheinlichsten Schlüssel eingespeichert werden. Mit diesen wird dann der Wörterbuchangriff gestartet. Bei modernen Kryptosystemen spielt diese Angriffsmöglichkeit keine Rolle, da die Schlüssel zu überwiegenden Teil zufällig erstellt wurden und nicht für ein Wörterbuch vorhergesagt werden können.

[2] vgl. [Küh04]
[3] vgl. [TOP08]
[4] vgl. [Koi09]: Das Downadup-Botnetz besteht wahrscheinlich aus 9.000.000 Rechnern.

Anders sieht es zum Beispiel bei Passwortabfragen aus. Dort werden Wörterbuchangriffe häufig verwendet, da Menschen dazu neigen, Passwörter zu benutzen, die in der Alltagssprache vorkommen und daher leicht zu merken sind.[5]

4.3. Häufigkeitsanalyse

Die *Häufigkeitsanalyse* ist eine Methode der Kryptoanalyse, die wichtig bei der Entzifferung einer monoalphabetischen Verschlüsselung ist. Dazu muss man zuerst wissen, in welcher Sprache der Klartext geschrieben ist, denn für jede Sprache gibt es spezifische Buchstabenhäufigkeiten, die exemplarisch in Abbildung 4.1 für die Sprachen Deutsch, Englisch, Französisch und Spanisch dargestellt sind.

Bemerkung 4.2. Die angegebenen Wahrscheinlichkeiten sind natürlich nur ein statistischer Mittelwert. Natürlich kann die Häufigkeitsverteilung bei speziellen Texten oder sehr kurzen Nachrichten ganz anders aussehen, jedoch sollte dies einem erfahrenen Kryptoanalytiker keine Schwierigekeiten bereiten.

Beispiel 4.3. [7] Die oben angesprochenen Unterschiede in der Häufigkeitsverteilung kann man anhand der folgenden zwei Nachrichten nachvollziehen:

- **Ein von Zitaten strotzender zoologischer Text über den Einfluss von Ozon auf die Zebras im Zentrum von Zaire.**
- **Ein Traktat über die amourösen Aventüren des Balthasar Matzbach am Rande des Panamakanals.**

In diesen beiden Texten würden bei einer Häufigkeitsanalyse wohl zunächst **z** beziehungsweise **a** für ein **e** gehalten werden.

Weiterhin existieren für jede Sprache noch Häufigkeitsverteilungen für Buchstabenpaare und -tripel. Zu Beispiel ist die Wahrscheinlichkeit recht hoch, dass in einem deutschen Text auf ein **c** ein **h** oder **k** folgt oder dass auf ein **q** ein **u** folgt, während die Wahrscheinlichkeit, dass auf ein **q** ein **z** folgt, praktisch Null ist.

Bei einer monoalphabetischen Verschlüsselung eines deutschen Textes geht man nun folgendermaßen vor: Zuerst zählt man die Häufigkeit der einzelnen Buchstaben, Buchstabenpaare und -tripel. Dann stellt man Hypothesen auf, welcher Buchstabe, welchem Geheimtextzeichen entsprechen könnte. Dabei fängt man mit den Buchstaben **e** und **n** an, die die höchste Häufigkeit in deutschen Texten haben und macht mit den Buchstaben **i**, **s**, **r**, **a** und **t** weiter. Dann sollte man, wenn der Text lang genug ist, normalerweise etwa 60% des Geheimtextes entziffert haben. Weitere Buchstaben, etwa **c** oder **h**, lassen sich anhand der gefundenen Buchstabenpaare und -tripel zuordnen. Ist man erst einmal an diesem Punkt angelangt, so lässt sich mit Inspizieren, Raten und Probieren die vollständige Lösung ermitteln. Bei anderen Sprachen muss man gegebenenfalls die Buchstaben in einer anderen Reihenfolge betrachten.

[5] vgl. [Rüt09]
[6] aus [Beu07] (Deutsch), [Lew00] (Englisch), [WF06] (Französisch) und [Pra40] (Spanisch)
[7] aus [Beu07]

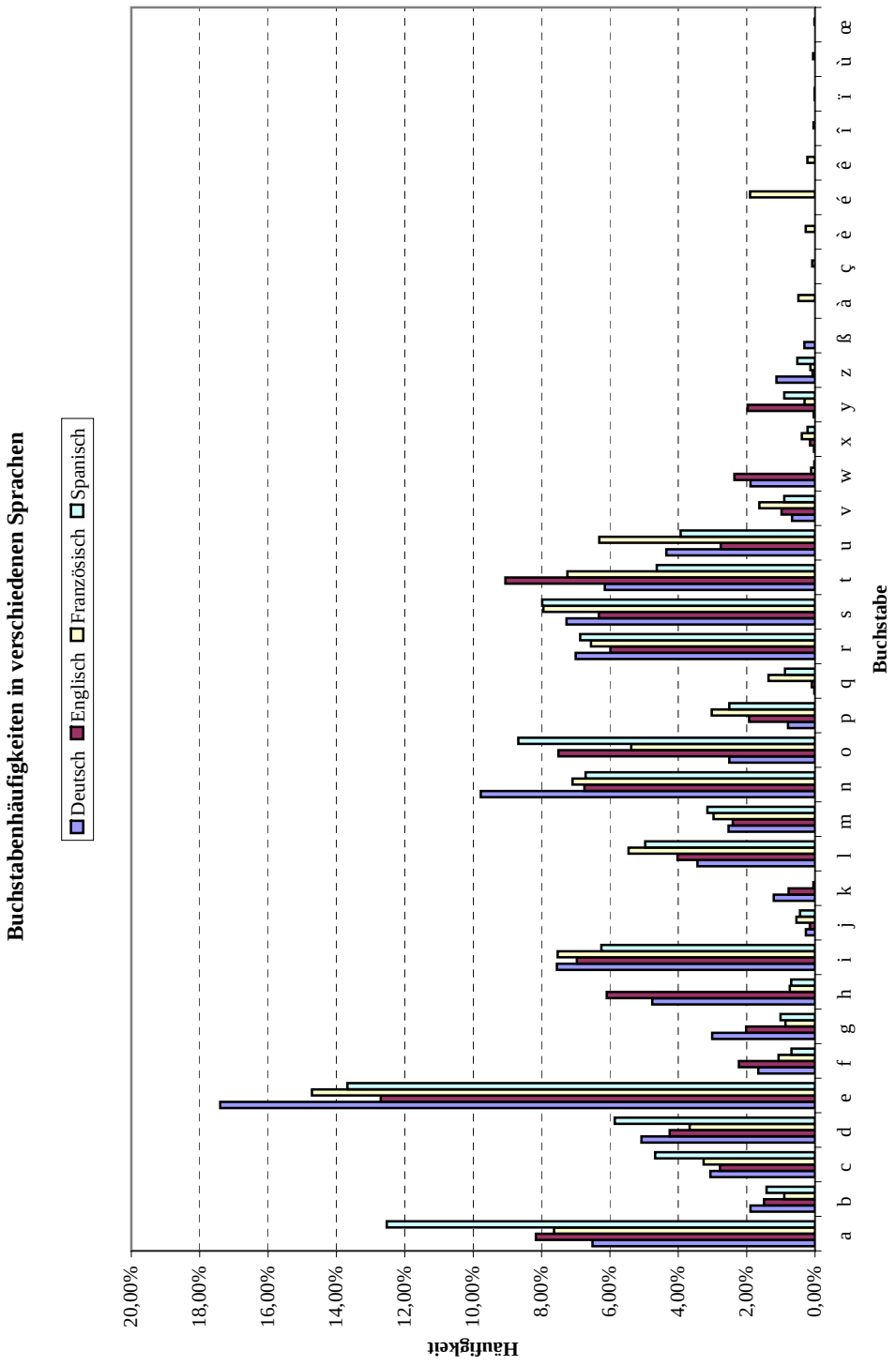

Abbildung 4.1.: Exemplarische Buchstabenhäufigkeit in den Sprachen Deutsch, Englisch, Französisch und Spanisch [6]

Beispiel 4.4.

- Der Geheimtext lautet **Cygvmzvmyf ycf jac Vaitfxymh jmz Ezutfqhqsym! Javmz cqhhfm kan kqnqahtvadmfycgvm Lmzcgvhimccmhins nygvf lmzomnjmn.**
- Auszählung der Häufigkeit der einzelnen Buchstaben:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
6	0	8	1	1	7	4	7	3	4	2	2	15	6	1	0	5	0	2	3	1	8	0	1	7	6

- **Hypothese 1:** **M**↔ *e*, da dies der mit Abstand häufigste Buchstabe ist.

  Hypothese 2: {**V**, **C**, **F**, **H**, **Y**, **Z**, **A**, **N**} sind Kandidaten für {*n*, *i*, *s*, *r*, *a*, *t*}.

  Hypothese 3: Da im Text fünf mal **MZ** auftaucht, vermuten wir nun **Z**↔*r*.

  Hypothese 4: Zweimal im Text **YM** und einmal **MY** lässt vermuten, dass **Y**↔*i*.

  Hypothese 5: Viermaliges Auftreten von **GV** im Text lässt vermuten, dass **G**↔*c* und **V**↔*h*.

  Hypothese 6: Wir können vermuten, dass **C**↔*s*, da einmal **CC**, zweimal **CGV** und bei den Wörtern, die aus drei Buchstaben bestehen, jeweils einmal ein **C** in der Mitte steht und einmal am Ende.

- Daraus ergibt sich bisher:
  Cygvmzvmyf ycf jac Vaitfxymh jmz Ezutfqhqsym! Javmz cqhhfm kan kqnqahtvadmfycgvm Lmzcgvhimccmhins nygvf lmzomnjmn. entspricht
  Sicherhei_ is_ _ _ s H_ _ _ _ _ ie_ _ er _r_ _ _ _ _ _ _ ie! _ _ her s_ _ _ _ e _ _ _ _ _ _ _ _ _ _ _ h_ _ e_ ische _ ersch_ _ esse_ _ _ _ _ ich_ _ er_ e_ _ e_ .

- **Hypothese 7:** Das erste Wort ergibt *Sicherheit*, daher **F**↔*t*.

  Hypothese 8: Das dritte Wort wird wahrscheinlich *das* lauten und damit **J**↔*d* und **A**↔*a*.

  Hypothese 9: Im vorletzten Wort kann man nun das Wort *nicht* erahnen und damit **N**↔*n*.

  Hypothese 10: Damit macht im neunten Wort nur die Ersetzung **K**↔*m* Sinn.

- Daraus ergibt sich bisher:
  Cygvmzvmyf ycf jac Vaitfxymh jmz Ezutfqhqsym! Javmz cqhhfm kan kqnqahtvadmfycgvm Lmzcgvhimccmhins nygvf lmzomnjmn. entspricht
  Sicherheit ist das Ha_ _ t_ ie_ der _r_ _ t_ _ _ _ ie! Daher s_ _ _ te man m_ n_ a_ _ ha_ etische _ ersch_ _ esse_ _ n_ nicht _ er_ enden.

Mit etwas Fantasie kann man nun auch noch den Rest der Zuordnungen erraten, so dass am Ende der Klartext *„Sicherheit ist das Hauptziel der Kryptologie! Daher sollte man monoalphabetische Verschluesselung nicht verwenden.“* erscheint mit der Zuordnungsregel

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	D	G	J	M	P	S	V	Y	B	E	H	K	N	Q	T	W	Z	C	F	I	L	O	R	U	X

Dabei ist aber die Zuordnung der Geheimtextzeichen **B**, **P**, **R** und **W** nicht eindeutig, da diese in der Nachricht gar nicht vorkommen.

4.4. Known Plaintext-Angriff

Wenn man weiß, welchen Inhalt eine verschlüsselte Nachricht hat, oder auch nur Teile daraus kennt, so kann man daraus auf den verwendeten Schlüssel schließen. So wurde zum Beispiel im Zweiten Weltkrieg von den Deutschen immer um 06:00 Uhr der Wetterbericht gefunkt[8]. Da dieser auch den Engländern bekannt war, hatte man nicht nur den abgefangenen Geheimtext, sondern auch den verwendeten Klartext zur Verfügung, um daraus den verwendeten Tagesschlüssel zu entziffern.

Beispiel 4.5. Bei der ENIGMA[9] wurde durch die Einführung der Umkehrwalze eine fixpunktfreie Permutation des Alphabets verwendet, das heißt, ein Buchstabe **x** kann durch die Verschlüsselung in alle anderen Buchstaben abgeändert werden, jedoch nie in ein **x**. Hat man nun einen *Known Plaintext*, zum Beispiel *„Das Oberkommando der Wehrmacht“*, so muss man untersuchen, an welchen Stellen das Wort nicht vorkommen kann, da an diesen Stellen einer oder mehrere Buchstaben des Geheimtextes mit dem *Known Plaintext* übereinstimmen. Hat man alle unbrauchbaren Stellungen eliminert, so kann man daraus auf die restlichen Zeichen schließen.

Bemerkung 4.6. Die in dem Beispiel vorgestellte Methode ist auch unter dem Begriff *Methode des Wahrscheinlichen Worts* oder *Mustersuche* bekannt.

Geheimtext	O	H	I	Y	P	D	O	M	Q	N	J	C	O	S	G	A	H	L	E	I	H	Y	N	O	N
Position 1			I	M	W	E	S	T	E	N	N	I	C	H	T	S	N	E	U	E	S				
Position 2				I	M	W	E	S	T	E	N	N	I	C	H	T	S	N	E	U	E	S			
Position 3					I	M	W	E	S	T	E	N	N	I	C	H	T	S	N	E	U	E	S		

Position 1 und 2 sind nicht möglich, da bei diesen Fällen einige Buchstaben des Geheimtextes mit dem möglichen Klartext übereinstimmen. Position 3 jedoch ist möglich.

Abbildung 4.2.: Mustersuche des Wortes *Im Westen nichts Neues* in einem Geheimtext, der mit einer fixpunktfreien Permutation erstellt wurde.

Bemerkung 4.7. Je mehr man von einer Nachricht weiß, desto mehr kann man mit dieser Methode erreichen. Deshalb waren die abgefangenen Wetterdaten so wichtig, da man bei diesen praktisch immer den kompletten Klartext kannte.

[8] vgl. [JK05]
[9] vgl. 3.2

4.5. Chosen Plaintext-Angriff

Diese Angriffsmethode entspricht der Known Plaintext Angriff, wobei hier der Angreifer die Nachricht vorgibt, die verschlüsselt gesendet werden soll. So kann man zum Beispiel gezielt Sabotage betreiben, um Nachrichten zu provozieren. Da dem Angreifer alle wichtigen Umstände der Sabotage bekannt sind, muss er nur noch den Funkspruch abwarten, der davon Meldung macht. Ein weiteres Beispiel kommt aus dem Zweiten Weltkrieg. Da die Engländer die deutschen Seekarten genau kannten und durch Spionage auch wussten, wie Minenfelder dem deutschen Oberkommando gemeldet wurden, legten die Engländer an vorher festgelegten Stellen Minenfelder an. Wurde dieses neue Minenfeld gemeldet, konnte an einer anderen Stelle ein neues Minenfeld gelegt werden.[10] Somit hatte man mehrere Nachrichten, dessen genauen Aufbau man kannte, sowohl im Klar- als auch im Geheimtext.

4.6. Seitenkanalangriff

Unter *Seitenkanalangriff* wird eine kryptoanalytische Methode bezeichnet, die die physikalische Implementierung eines Kryptosystems in einem Gerät oder in einer Software ausnutzt. Dabei wird bei einem modernen Kryptosystem zum Beispiel durch *Timing Attack* die verbrauchte Rechenzeit gemessen. Dazu provoziert man viele Verschlüsselungsvorgänge hintereinander und vergleicht die benötigte Rechenzeit und den Speicherbedarf. Da jedes Kryptosystem abhängig vom Schlüssel und der eingegebenen Nachricht unterschiedliche Ausführungszeiten besitzt, kann man durch eine Laufzeitanalyse sowohl das Kryptosystem als auch den Schlüssel rekonstruieren, wenn dieses nicht speziell dagegen geschützt worden ist, zum Beispiel durch Einfügen von Redundanzen, um Maschinenbefehle datenunabhängig auszuführen.[11]

Beispiel 4.8. [12] Ein sehr frühes Ausnutzen des Seitenkanalangriffs war theoretisch bei der Skytale möglich. Wenn der Papyrusstreifen nicht sehr ordentlich geglättet wurde oder die Buchstaben beim ausgerollten Papyrusstreifen zu sehr geneigt waren, konnte man auf die Dicke der Skytale schließen.

Bemerkung 4.9. Seitenkanalangriffe sind heutzutage mit die gefährlichsten Angriffsszenarien. Gerade bei Kryptosystemen, die auf einer speziellen Hardware laufen, wie zum Beispiel auf Chipkarten oder auf Java-Smartcards, werden immer wieder Seitenkanalangriffe ausprobiert. Und da dabei theoretisch alles möglich ist, was dem Angreifer einfällt, ist kein Public Key-Verfahren grundsätzlich gegen Seitenkanalangriffe gerüstet.[13] Vor allem die von den modernen Kryptosystemen verwendeten Zufallszahlengeneratoren ermöglichen mitunter einen erfolgreichen Angriff, da diese nur Pseudo-Zufallszahlen generieren können.

[10] vgl. [JN06]
[11] vgl. [Sch07]
[12] vgl. [MQ06]
[13] vgl. [May06] und [Sch07]

4.7. Der Kasiski-Test und der Friedman-Test

Der *Kasiski-Test*, der unabhängig voneinander von Charles Babbage 1854 entdeckt und von Friedrich Kasiski 1863 veröffentlicht wurde, und der *Friedman-Test*, der 1925 von William Friedman entwickelt wurde, bilden zusammen eine wirkungsvolle Methode, um eine polyalphabetische Verschlüsselung, etwa die Vigenère-Verschlüsselung, zu brechen.

Definition 4.10. Man untersucht den Geheimtext und bestimmt alle Abstände gleicher Folgen, die aus mindestens drei Buchstaben bestehen. Der größte gemeinsame Teiler dieser Abstände ist (vermutlich) ein Vielfaches oder Teiler der Schlüsselwortlänge. Diesen Test nennt man *Kasiski-Test*.

Bemerkung 4.11. Der Test beruht auf folgender Idee: Existieren im Klartext zwei Folgen aus gleichen Buchstaben, zum Beispiel **ein**, so werden diese normalerweise in verschiedene Geheimtextzeichen chiffriert. Werden aber beide Folgen mit den gleichen Schlüsselwortbuchstaben verschlüsselt, so ist die Folge im Geheimtext auch identisch.

Beispiel 4.12.

Schlüsselwort:	*Geheim*																	
Schlüssel	G	E	H	E	I	M	G	E	H	E	I	M	G	E	H	E	I	M	G	E
Klartext	...		e	i	n	...			...			e	i	n	...			
Geheimtext	...		L	M	V	...			...			Q	O	R	...			

Im Allgemeinen wird jede Klartextbuchstabenfolge im Geheimtext in eine andere Geheimtextbuchstabenfolge chiffriert.

Beispiel 4.13.

Schlüsselwort:	*Geheim*																	
Schlüssel	G	E	H	E	I	M	G	E	H	E	I	M	G	E	H	E	I	M	G	E
Klartext	...		e	i	n	...						...		e	i	n	...	
Geheimtext	...		L	M	V	...						...		L	M	V	...	

Stehen die Klartextbuchstabenfolgen aber an der gleichen Stelle bezogen auf das Schlüsselwort, so ist auch im Geheimtext diese Folge identisch.

Für den *Friedman-Test* fragt man sich zunächst, mit welcher Wahrscheinlichkeit ein willkürlich ausgewähltes Buchstabenpaar aus gleichen Buchstaben besteht. Dazu sei eine Buchstabenfolge der Länge n gegeben, die aus den Buchstaben **a** bis **z** besteht. Dann ist die Anzahl der **a**'s gleich n_1, die Anzahl der **b**'s gleich n_2, ... und n_{26} gleich der Anzahl der **z**'s. Da die Anzahl der nicht notwenigerweise zusammenhängenden Paare, bei denen zwei Buchstaben dem i-ten Buchstaben entsprechen, gleich $\frac{n_i(n_i-1)}{2}$ ist, es kommt nicht auf die Reihenfolge an, ist also die Gesamtanzahl aller Paare, bei denen zwei Buchstaben gleich sind,

$$\frac{n_1(n_1-1)}{2}+\frac{n_2(n_2-1)}{2}+\cdots+\frac{n_{26}(n_{26}-1)}{2}=\sum_{i=1}^{26}\frac{n_i(n_i-1)}{2}$$

Erster Schlüsselwort-buchstabe	Zweiter Schlüsselwort-buchstabe	Dritter Schlüsselwort-buchstabe	...	n-ter Schlüsselwort-buchstabe
1	2	3	...	h
h+1	h+2	h+3	...	2h
2h+1	2h+2	2h+3	...	3h
3h+1	3h+2	3h+3	...	4h
...	...	...	...	...

Abbildung 4.3.: Zusammenhang zwischen den Buchstaben des Geheimtextes und den Buchstaben des Schlüsselwortes bei der Vigenère-Verschlüsselung

Definition 4.14. Die Wahrscheinlichkeit, ein Paar aus gleichen Buchstaben in einer Nachricht der Länge n zu erhalten, beträgt

$$\kappa = \frac{\sum_{i=1}^{26} n_i(n_i - 1)}{n(n-1)}$$

Diese Zahl heißt *Koinzidenzindex* oder *Friedmanscher Koinzidenzindex.*

Bemerkung 4.15. Wenn wir für jeden Buchstaben i wissen, mit welcher Wahrscheinlichkeit p_i er in einem Text auftritt, dann können wir den *Koinzidenzindex* auch so berechnen:

$$\kappa = \sum_{i=1}^{26} p_i^2$$

Somit ergibt sich

- Der *Koinzidenzindex der deutschen Sprache* mit den Wahrscheinlichkeiten aus Abbildung 4.1 ist $\kappa = 0,0762$.
- Der *Koinzidenzindex einer zufälligen Buchstabenfolge*, bei der jeder Buchstabe die Wahrscheinlichkeit $p_i = \frac{1}{26}$ hat, entspricht $\kappa = 0,0385$.

Definition 4.16. Angenommen, es sei ein Geheimtext der Länge n in deutscher Sprache gegeben. Daraus kann man den Koinzidenzindex κ berechnen. Die Schlüssellänge h berechnet sich nun aus[14]

$$h = \frac{0,0377n}{\kappa(n-1) - 0,0385n + 0,0762}$$

Dieser Test heißt nach seinem Entwickler *Friedman-Test.*

[14] vgl. [Beu07]. Dort wird auch eine ausführliche Herleitung der Formel angegeben, auf die hier in der Arbeit verzichtet wird.

Bemerkung 4.17. Wenn man erst den Kasiski-Test und dann den Friedman-Test auf einen Geheimtext anwendet, wird zuerst durch den Kasiski-Test ein Teiler oder Vielfaches h der Anzahl der Schlüsselwortbuchstaben angegeben. Durch den Friedman-Test wird die Größenordnung von h festgelegt. Die darauffolgende Analyse des Schlüsselworts ist dann nur noch das Brechen einer monoalphabetischen Verschlüsselung, da in jeder Spalte von Abbildung 4.3 eine Caesar-Verschlüsselung stattfindet.

Beispiel 4.18. Der folgende Geheimtext wurde abgefangen :

MRJI QGN, NHY! XLLXOFTGPMH,
VUENJBIUQI HSU UIGUZVS,
LVH OQIQJI IYFT TUJFTSJUE
QZIKLDGS FYLLMHDT, ZNK PILESRR SMQXQHA.
IR AXHT IPM ECR, LOH NWDMV WAR!
HSU JMQ EO XQLO EOE WVJ QCZRD;
HRNJAI PMGVXKMV, KQIFXV LSNFOE LRZ
YQP ZVJYM WFTOA FE LMH LEUJE REKD
HRWRCJ, KQRNG LVH TGEE ZEL OUGMZ
RVQRH ECUZVTIU MN QJI VEVQ HRWLU –
YQP SRMV, LEVE WVW EQGKFS JNJAIQ WORSEMR!
GMS JNCT QLD SPMZMV GMS UJIH ZHDBEJEVIQ.
LWNW SQR LOH TJJKLHUTRW RTW DXL QNV TEIREA,
IFSXRDEA, RROMVFEE, XTPVHUBRW LVH SRASKVV;

Nun ergibt der Kasiski-Test für Buchstabenfolgen der Länge drei folgendes Ergebnis:

Buchstabenfolge	Abstand zwischen dem Auftreten	Primfaktoren
HSU	88	$2^3 \cdot 11$
UZV	194	$2 \cdot 97$
LVH	168	$2^3 \cdot 3 \cdot 7$
QJI	192	$2^6 \cdot 3$
LMH	112	$2^4 \cdot 7$
IPM	45	$3^2 \cdot 5$
RLO	216	$2^3 \cdot 3^3$
LOH	216	$2^3 \cdot 3^3$
SUJ	182	$2 \cdot 7 \cdot 13$
EWV	128	2^7
DHR	56	$2^3 \cdot 7$
NJA	128	2^7
JAI	128	2^7
YQP	80	$2^4 \cdot 5$
HRW	48	$2^4 \cdot 3$
RWR	136	$2^3 \cdot 17$
LVH	168	$2^3 \cdot 3 \cdot 7$
RWL	128	2^7
SJN	16	2^4
GMS	16	2^4

Nachdem hier nur die Buchstabenfolgen der Länge drei betrachtet wurden, ist das Ergebnis leider noch nicht eindeutig. Als mögliche Ergebnisse kommen die Zahlen 2, 3 und 7 in Frage sowie deren Vielfaches. Die anderen auftretenden Zahlen wie 17 oder 97 kommen höchstwahrscheinlich daher, dass in dem Text zufällige Buchstabenfolgenwiederholungen auftreten.

Anschließend wird der Friedman-Test durchgeführt. Um für diesen die oben angegebene Formel anwenden zu können, brauchen wir zwei Angaben: Die Textlänge n und den Koinzidenzindex κ des Textes. Wenn wir alle Buchstaben auszählen, so erhalten wir $n = 387$ und

$$\begin{aligned}\kappa &= \frac{\sum_{i=1}^{26} n_i(n_i-1)}{n(n-1)} \\ &= \frac{6.620}{149.382} \\ &= 0,044315915\end{aligned}$$

Eingesetzt in die Formel ergibt das

$$\begin{aligned}h &= \frac{0,0377n}{\kappa(n-1) - 0,0385n + 0,0762} \\ &= \frac{0,0377 \cdot 387}{0,044315915 \cdot 386 - 0,0385 \cdot 387 + 0,0762} \\ &= 6,39166923\end{aligned}$$

Nachdem Friedman-Test ist die Größenordnung der Schlüsselwortlänge vermutlich aus dem Intervall $[5, 9]$. Verteilt man nun, wie in Abbildung 4.3 angedeutet, die Geheimtextbuchstaben auf die Schlüsselbuchstaben, so hat man nur noch für die verschiedenen Schlüsselwortlängen Caesar-Verschlüsselungen zu brechen. Ist dies erledigt, ergibt sich der einzig sinnvolle Klartext bei der Schlüsselwortlänge 8 und der Klartext lautet

Habe nun, ach! Philosophie,
Juristerei und Medizin,
Und leider auch Theologie
Durchaus studiert, mit heissem Bemuehn.
Da steh ich nun, ich armer Tor!
Und bin so klug als wie zuvor;
Heisse Magister, heisse Doktor gar
Und ziehe schon an die zehen Jahr
Herauf, herab und quer und krumm
Meine Schueler an der Nase herum –
Und sehe, dass wir nichts wissen koennen!
Das will mir schier das Herz verbrennen.
Zwar bin ich gescheiter als all die Laffen,
Doktoren, Magister, Schreiber und Pfaffen;[15]

[15] aus [Goe86]

mit dem Schlüsselwort **FRIEDMAN**.

Bemerkung 4.19. Mithilfe des Koinzidenzindexes kann man auch sehr leicht nachweisen, ob ein Text monoalphabetisch verschlüsselt wurde oder polyalphabetisch. Da der Sinn einer polyalphabetischen Verschlüsselung darin liegt, die Häufigkeiten der einzelnen Buchstaben innerhalb einer Nachricht anzugleichen, muss man den Koinzidenzindex der Nachricht berechnen. Ist dieser bei einem deutschsprachigen Text ungefähr 0,0762, so ist die Nachricht wahrscheinlich monoalphabetisch verschlüsselt, ist er aber deutlich kleiner, wurde wahrscheinlich eine polyalphabetische Chiffrierung verwendet.

4.8. Bedienfehler

Oftmals wird die Kryptoanalyse durch einfache *Bedienfehler* bei der Benutzung von Kryptosystemen erleichtert. So zählt unter anderem ein einfach zu erratender Schlüssel dazu. Beispielsweise verwendete Julius Cäsar immer für seine geheime Kommunikation eine Caesar-Verschlüsselung mit der Verschiebung 3 und Augustus mit der Verschiebung 1, da dies den Anfangsbuchstaben ihrer Namen entspricht.[16] Auch Sorglosigkeit mit der Schlüsselverwaltung zählt dazu. So wurden zum Beispiel im Dritten Reich für das Auswärtige Amt Codebücher gedruckt, die Schlüssel für den One-Time-Pad enthielten. Allerdings nahm man es mit der Individualität der Schlüssel nicht so genau und deshalb druckte man von jedem Codebuch nicht zwei, sondern neun Kopien, von denen mindestens fünf an verschiedene Vertretungen im Ausland geschickt wurden.[17]

Beispiel 4.20. [18] Während des Zweiten Weltkrieges wurde von den Engländern eine chiffrierte Nachricht abgefangen, bei der nach aufmerksamer Betrachtung auffiel, dass der Text kein **L** enthielt. Da die ENIGMA, bedingt durch die Umkehrwalze, fixpunktfrei permutiert, erkannte man nach einer kurzen Untersuchung, dass der Klartextes nur aus lauter **L**-Buchstaben bestand. In diesem Fall hatte ein deutscher Funker in Italien versucht, durch Absetzen einer inhaltsleeren Meldung Funkverkehr vorzutäuschen und damit von wichtigeren Meldungen abzulenken.

Bei all diesen Beispielen haben Bedienfehler die Entzifferung der jeweiligen Nachrichten stark vereinfacht, oder, wie in dem Fall der Nachricht, die nur aus dem Buchstaben **L** bestand, konnte man sehr einfach den verwendeten Tagesschlüssel erraten und damit weitestgehend alle Funksprüche eines Tages entziffern. Es setzt jedoch eine große Erfahrung im Brechen von Kryptosystemen voraus, solche Bedienfehler überhaupt zu erkennen und da Bedienfehler nicht regelmäßig vorkommen, ist ein systematisches Ausnutzen nicht möglich.

[16] vgl. [SSP08]
[17] vgl. [Roh79]
[18] vgl. [Pah08]

5. Sicherheit von Kryptosystemen und Authentifizierung

Will man die Sicherheit eines Kryptosystems bewerten, so muss man sich dabei die zwei wesentlichen Aspekte eines Kryptosystems ansehen:

- Das Verfahren, das heißt, die zugrundeliegenden Ideen und ihre praktischen Umsetzungen.
- Die Schlüsselverwaltung von symmetrischen, aber auch asymmetrischen Kryptosystemen.

5.1. Sicherheit des Verfahrens

Oftmals bringt es Entwicklern nur sehr wenig, nach dem Prinzip *Security through obscurity* vorzugehen und das Verfahren geheim zu halten, um eine erhöhte Sicherheit zu gewinnen. In der Vergangenheit wurden häufig Verschlüsselungsmethoden, die geheimgehalten werden sollten, zuerst rekonstruiert und später dann auch noch gebrochen, so zum Beispiel das bei W-LAN[1]-Netzen eingesetzte WEP[2]-Verschlüsselungsverfahren[3].

Wir wollen uns daher zunächst mit dem Begriff der *perfekten Geheimhaltung* befassen. Sie wurde erstmals von Claude Shannon[4] formuliert.

Definition 5.1. [5] Sei g ein Geheimtext und k ein Klartext. Das Kryptosystem heißt *perfekt geheim*, wenn die Ereignisse, dass ein bestimmter Geheimtext auftritt und dass ein bestimmter Klartext vorliegt, unabhängig sind, das heißt, wenn die bedingte Wahrscheinlichkeit $\mathbb{P}(k|g)$ gleich der Wahrscheinlichkeit $\mathbb{P}(k)$ ist.

Bemerkung 5.2. Der One-Time-Pad genügt dieser Definition und ist daher perfekt geheim.

Beispiel 5.3. Wenn man annimmt, dass das verwendete Alphabet gleich $\{0, 1\}$ ist und das Kryptosystem darauf basiert, dass die einzelnen Buchstaben des Klartextes mit dem Schlüssel bitweise per `xor`[6] verknüpft werden. Wenn nun ein Angreifer, ohne den Schlüssel zu kennen, die Nachricht **01** abfängt, so kann er daraus unmöglich den Klartext rekonstruieren. Alle Möglichkeiten wären:

[1] Wireless LAN / Wireless Local Area Network
[2] Wired Equivalent Privacy
[3] vgl. [BTPW07]
[4] vgl. [Sha49]
[5] vgl. [Buc08]
[6] Diese Verknüpfung bewirkt: $0 \oplus 0 = 0$; $1 \oplus 0 = 1$; $0 \oplus 1 = 1$; $1 \oplus 1 = 0$

- **00** mit Schlüssel **01**.
- **01** mit Schlüssel **00**.
- **10** mit Schlüssel **11**.
- **11** mit Schlüssel **10**.

Man sieht an diesem Beispiel, dass es dem Angreifer nicht einmal etwas nutzen würde, wenn er Teile des Schlüssels oder auch des Klartextes kennen würde, da sowohl der restliche Klartext als auch der weitere Geheimtext davon vollständig unabhängig sind.

Beispiel 5.4. In dem Fall, dass eine Nachricht nur aus einem Buchstaben besteht, ist sogar das Caesar-Verfahren perfekt geheim.

Neben der *perfekten Geheimhaltung* gibt es noch weitere Sicherheitsbegriffe, die in der Literatur oftmals unter verschiedenen Bezeichnungen geführt werden:[7]

Computationally secure / Sicher in der Praxis Dies ist die schwächste Sicherheit. Sie sagt nur aus, dass der beste *bekannte* Angriff in der Praxis scheitert, da er zu lange dauern würde. Da sich aber die Computer mit der Zeit[8] ständig verbessern und womöglich auch nicht alle Angriffe bekannt sind, muss man davon ausgehen, dass solche Kryptosysteme in der Zukunft gebrochen werden.

Beispiel 5.5. Die ENIGMA wurde von der deutschen Wehrmacht als *sicher in der Praxis* eingestuft, da man davon ausging, dass der Feind circa 14.000 Jahre brauchen würde, das System zu knacken.[9]

Provably secure / Beweisbar sicher Wenn die Sicherheit des Problems auf ein altbekanntes und wohluntersuchtes Problem zurückgeführt werden kann, spricht man von *beweisbar sicher relativ zu diesem Problem.* Da die zugrundeliegenden Probleme meist erschöpfend untersucht worden sind, ist die Sicherheit solcher Verfahren wahrscheinlich für längere Zeit gesichert.

Beispiel 5.6. Probleme, die sich dafür eignen, sind unter anderem die Faktorisierung ganzer Zahlen oder die Berechnung diskreter Logarithmen.

Unconditionally secure / Informationstheoretisch sicher / Perfekt geheim Verfahren mit der höchsten Sicherheit garantieren, dass selbst Angreifer mit unbegrenzter Rechenkapazität dieses Verfahren nicht brechen können. Daher wird das Verfahren für immer sicher bleiben.

Bemerkung 5.7. Diese Sicherheitseinstufungen sind natürlich nur theoretischer Natur. So ist es zum Beispiel einem Angreifer möglich, durch Zufall oder mit viel Glück, eine Nachricht sofort entschlüsseln zu können, auch wenn diese mit einem One-Time-Pad

[7] vgl. [Bai06]
[8] vgl. [Int09]: Das Mooresche Gesetz besagt, dass sich die Transistorzahl auf einem Chip alle 24 Monate verdoppelt und sich damit auch die Rechenleistung signifikant verbessert.
[9] vgl. [Sau02]

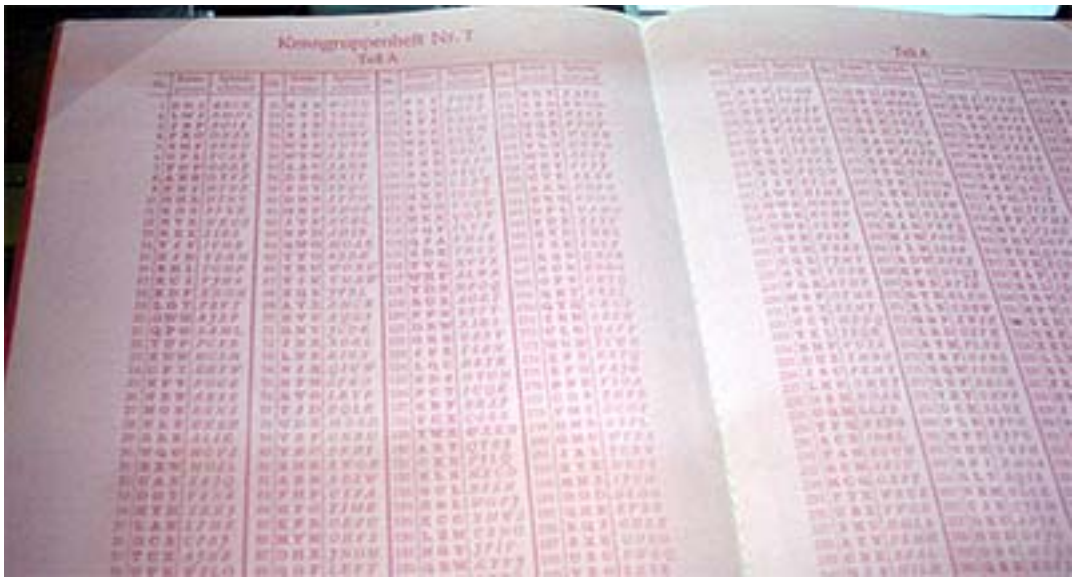

Abbildung 5.1.: Ein von der deutschen Wehrmacht verwendetes Codebuch

verschlüsselt worden ist. Anders herum gibt es auch heute, also mehr als 60 Jahre nach dem Zweiten Weltkrieg, immer noch mit der ENIGMA verschlüsselte Funksprüche, die nicht entziffert worden sind,[10] obwohl die ENIGMA „nur“ *sicher in der Praxis* ist.

5.2. Sichere Schlüsselverwaltung

Gerade bei symmetrischen Verschlüsselungen ist die Schlüsselverwaltung sehr problematisch. Wie kann man seinem Kommunikationspartner den zu verwendenden Schlüssel übertragen, ohne dass ein potentieller Angreifer diesen mitbekommt? Kann man den Schlüssel vor dem Absenden der Nachricht durch einen vertrauenswürdigen Boten überbringen, so muss danach bei beiden Kommunikationspartnern der Schlüssel so gut aufbewahrt werden, dass ein Dritter keinen Zugriff darauf haben kann. Vor allem müssen anfänglich genügend viele Schlüssel ausgetauscht werden, wenn große Mengen an Nachrichen geschickt werden müssen. Daher sind bei symmetrischen Verschlüsselungen häufig *Codebücher* im Einsatz, wie in Abbildung 5.1 zu sehen, in denen tage- oder nachrichtenweise Schlüssel zugeordnet werden können. Solche Codebücher gibt es in verschiedenster Ausführung und werden gerne mit Agenten und Spionage in Verbindung gebracht. Dies ist auch die größte Gefahr, die passieren kann, denn wenn ein feindlicher Agent ein Codebuch in seine Gewalt bekommen hat, so kann er unter Umständen die komplette Kommunikation eines gewissen Zeitraums mitlesen.

Beispiel 5.8. [11] Am 19. Januar 1917 sandte der Staatssekretär im Auswärtigen Amt, Arthur Zimmermann, ein verschlüsseltes Telegramm nach Mexiko, die sogenannte *Zim-*

[10] vgl. [Kra09]
[11] vgl. [Nas92]

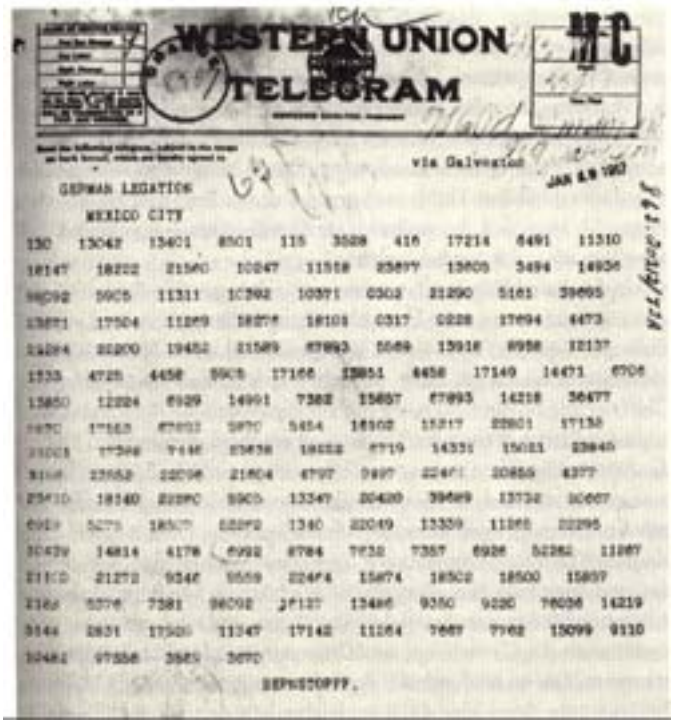

Abbildung 5.2.: Die Zimmermann-Depesche

mermann-Depesche, in dem er um einen Kriegseintritt Mexikos gegen Amerika warb. Da es aber Russland zuvor gelungen war, zwei Codebücher der Deutschen zu erbeuten und eines davon an England weitergegeben hatten, konnten die Engländer in Room 40[12] die abgefangene Nachricht entziffern und dann an Amerika weitergeben. Dieses Telegramm wird auch als einer der Gründe genannt, warum die USA sich in den europäischen Konflikt einmischten und Deutschland den Krieg erklärten.

Bei der asymmetrischen Verschlüsselung ist dieses Problem nur noch in abgeschwächter Form vorhanden, da der dafür verwendete Public Key zwar ohne Probleme veröffentlicht werden kann, der Private Key dafür aber geheim gehalten werden muss. Die Geheimhaltung des Private Keys stellt im Allgemeinen zwar kein Problem dar, ist aber zum Beispiel auf Chipkarten, die gestohlen werden können, nur problematisch umzusetzen.

5.3. Authentifizierung

Ein viel größeres Problem bei asymmetrischen Verschlüsselungen ist die *Authentifizierung* und dabei die *Nachrichtenauthentifizierung* und die *Teilnehmerauthentifizierung*. Während man bei einer symmetrischen Verschlüsselung meist davon ausgeht, dass der Kommunikationspartner derjenige ist, für den er sich ausgibt, da ja der Schlüssel niemand

[12] Eine Abteilung des britischen Marine-Nachrichtendienstes während des Ersten Weltkriegs

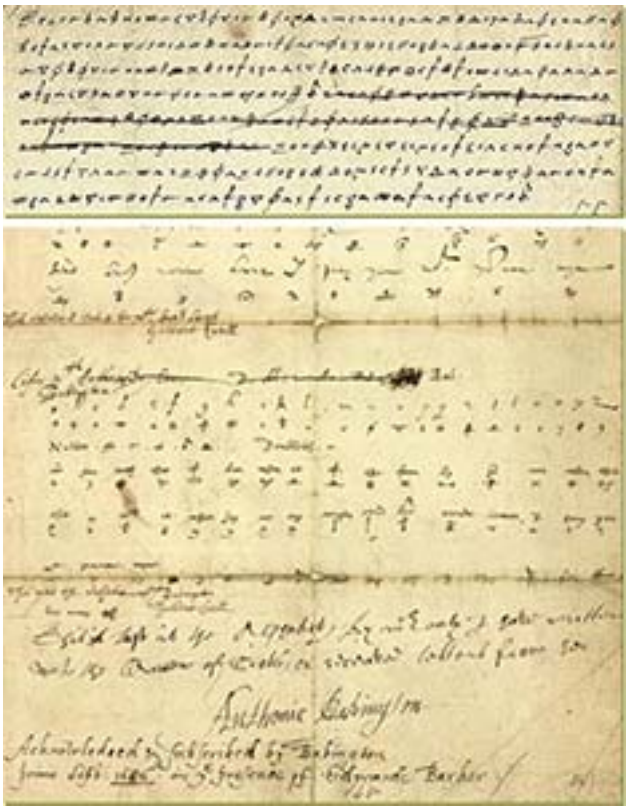

Abbildung 5.3.: Eine Korrespondenz zwischen Maria Stuart und den Mitverschwörern

anderem bekannt sein sollte, kann man dies bei der asymmetrischen Verschlüsselung nicht mehr. Aus dem gleichen Grund geht man bei der symmetrischen Verschlüsselung auch davon aus, dass die Nachricht nicht verändert worden ist.

Beispiel 5.9. [13] Dass man sich auch bei einer symmetrischen Verschlüsselung nicht zu sicher sein sollte, zeigt das Beispiel der *Babington-Verschwörung*. Bei dieser hatten sich einige junge katholische englische Adelige um Maria Stuart gesammelt, um die protestantische Königin Elisabeth I. zu beseitigen und anschließend mit Maria Stuart eine katholische Königin auf den Thron zu setzen. Da Maria Stuart zu dieser Zeit im Gefängnis saß, musste eine Kommunikation mit ihr verschlüsselt werden. Eine solche verschlüsselte Nachricht ist in Abbildung 5.3 zu sehen. Jedoch war der Bote, der die Nachrichten überbringen sollte, ein Spion der englischen Krone und kopierte alle überbrachten Nachrichten für Francis Walsingham, den Sicherheitsminister von Elisabeth. Es dauerte nicht lange, bis die Verschlüsselung gebrochen wurde und ab diesem Zeitpunkt konnte alle Korrespondenz von Walsingham mitgelesen werden. Da er für eine Anklage alle Mitverschwörer kennen wollte, ließ er durch den Boten die Nachrichten fälschen, indem er einen weiteren Text ergänzen sollte. Dies war möglich, da man das zugrunde liegende Kryptosystem und die verwendeten Schlüssel ermittelt hatte. Maria Stuart fiel die Fäl-

[13] vgl. [Sin06]

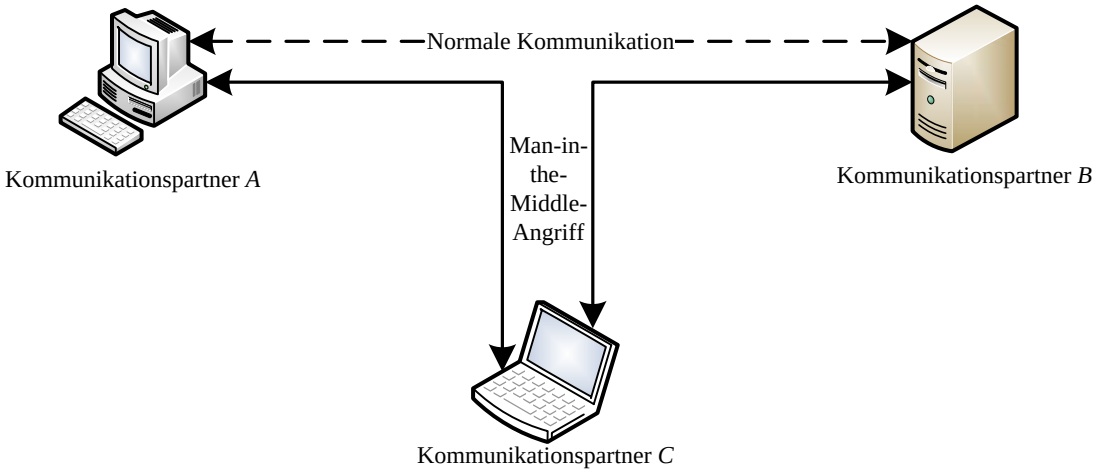

Abbildung 5.4.: Man-in-the-Middle-Angriff

schung nicht auf und da sie auf den gefälschten Brief antwortete, unterschrieb sie somit ihr Todesurteil und das aller Mitverschwörer.

Bemerkung 5.10. Das in diesem Beispiel beschriebene Verfahren, ist ein klassischer *Man-in-the-Middle-Angriff*, der auch in Abbildung 5.4 skizziert ist. Dieser ist gerade auch für asymmetrische Verschlüsselungen eine ernsthafte Bedrohung, da man bei diesen nicht darauf hoffen kann, dass der Feind den verwendeten Schlüssel nicht herausbekommt.

Prinzipiell läuft ein Man-in-the-Middle-Angriff bei einem Public Key-Verfahren nach folgendem Schema ab: A möchte B eine Nachricht schreiben. Da C diese Nachricht entweder abhören oder sogar verändern möchte, stört er die Kommunikation zwischen A und B und gibt sich selbst gegenüber A als B aus. Dafür schickt er A seinen Public Key. A hat selber keine Möglichkeit nachzuprüfen, ob diese Angabe stimmt und schickt C die mit dem Public Key von C verschlüsselte Nachricht. C entschlüsselt sie mit seinem Private Key und hat nun die Nachricht im Klartext vorliegen und kann sie nun eventuell verändern. Nun gibt sich C gegenüber B als A aus und schickt die eventuell veränderte Nachricht mit dem Public Key von B verschlüsselt weiter. Analog verhält sich C bei der Rückantwort von B.[14]

Um dieses Szenario auszuschließen, braucht man zwei Dinge:

- *Digitale Zertifikate*, die die Identität eines Kommunikationsteilnehmers garantieren.
- Eine *Digitale Signatur*, um eine Veränderung der Nachricht zu erkennen.

5.3.1. Zertifikate

Definition 5.11. Unter einem *digitalen Zertifikat* versteht man bei Public Key-Verschlüsselungsverfahren strukturierte Daten, die den Eigentümer eines Public Keys ein-

[14] vgl. [Heß06]

deutig identifizieren. Zumeist werden in einem *digitalen Zertifikat* noch weitere Daten gespeichert, wie zum Beispiel der Aussteller des Zertifikats oder dessen Gültigkeit.

Bemerkung 5.12. Ein *digitales Zertifikat* bringt so lange keinen zusätzlichen Nutzen, so lange man keine vertrauenswürdige Stelle hat, die dieses Zertifikat ausstellt. Denn wenn sich jeder selber ein Zertifikat ausstellen kann, dann ist es ein Einfaches, die benötigten Daten zu fälschen. Deshalb hat man dafür eine *Public Key Infrastructure* eingeführt.

Definition 5.13. Eine *Public Key Infrastructure* besteht aus

- einer *Zertifizierungsstelle*, die Zertifikatsanträge signiert und entweder das Root-CA-Zertifikat[15] bereitstellt oder als Zwischenzertifizierungsstelle fungiert,
- einer *Registrierungsstelle*, bei der Zertifikate beantragt werden können und die die Richtigkeit der in der Antragstellung angegebenen Daten überprüft,
- einem *Validierungsdienst*, der eine Überprüfung von Zertifikaten in Echtzeit ermöglicht und
- *digitalen Zertifikaten.*

In Abbildung 5.5 sind die Zusammenhänge zwischen den einzelnen Teilnehmern einer Public Key Infrastructure noch einmal symbolisch an einem Beispiel dargestellt. Das Beispiel lautet wie folgt: Ein Shopbetreiber möchte ein Zertifikat erhalten, mit dem er den Public Key seines Shops zertifizieren will. Dafür wendet er sich zuerst an die Registrierungsstelle, die seine Daten prüft. Sind diese in Ordnung, wird eine Meldung an die Zertifizierungsstelle ausgegeben, dass ein Zertifikat ausgestellt werden kann. Die Zertifizierungsstelle stellt darauf hin das Zertifikat aus und übermittelt dieses sowohl an den Shopbetreiber als auch an den Validierungsdienst. Nachdem der Shopbetreiber nun den Public Key und das Zertifikat in den Shop mit eingebunden hat, kann ein Besucher das Zertifikat und den Public Key abrufen und diese beim Validierungsdienst überprüfen lassen. Bestätigt der Validierungsdienst die Daten, kann der Benutzer einigermaßen sicher sein, dass das Zertifikat gültig und nicht gefälscht ist.[16]

Bemerkung 5.14.

- Gerade bei kleineren Public Key Infrastructuren kann sich die Registrierungsstelle und die Zertifizierungsstelle organisatorisch überschneiden, da man damit etwas Bürokratie vermeiden kann.
- Es kann vorkommen, dass innerhalb einer Public Key Infrastructure alle Organisationen mehrfach vorkommen und somit auch, dass mehrere Root-CA-Zertifikate existieren. So ist das Internet ein Beispiel für diesen Fall.

[15] vgl. Definition 5.16
[16] vgl. [Mic07]

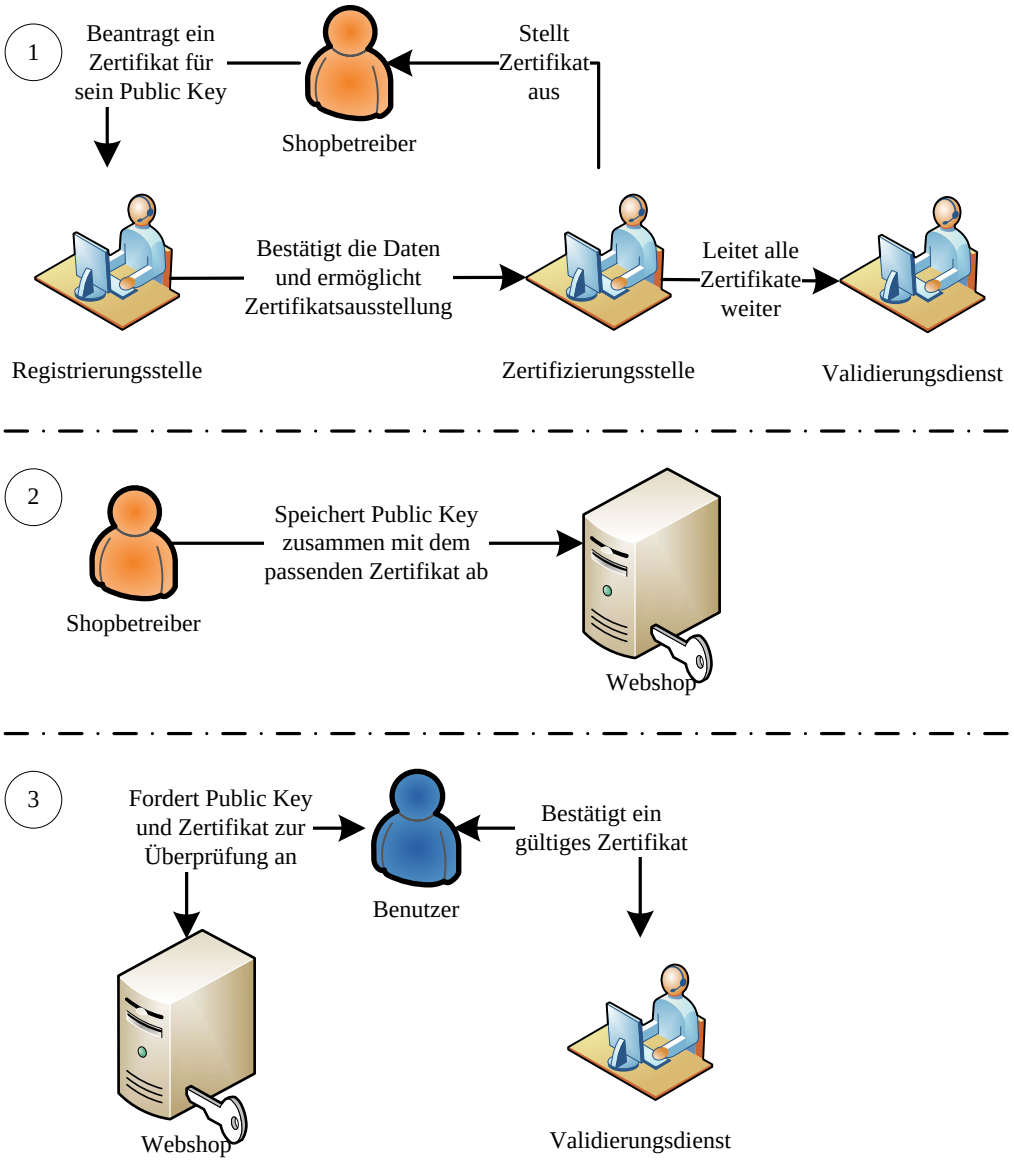

Abbildung 5.5.: Beispielschema einer Public Key Infrastructure

Beispiel 5.15. Im Internet ist der Browser ein Teil des Validierungsdienstes. Gerade die Root-CA-Zertifikate sind im Browser abgespeichert. Alle anderen Zertifikate werden entweder von einem Onlinevalidierungsdienst als ungültig eingestuft oder durch eine Zertifikatskette validiert.

Definition 5.16. [17] Eine *Zertifikatskette* ist eine Folge von Zertifikaten, an deren Anfang ein spezielles CA-Zertifikat, das *Root-CA-Zertifikat*, und an dessen Ende das zu betrachtende Zertifikat steht. Dazwischen stehen unter Umständen mehrere Zwischen-

[17] vgl. [Buc08]

zertifizierungsstellen, die durch ein CA-Zertifikat[18] ausgewiesen werden. Wenn innerhalb dieser Folge bei keinem Zertifikat irgendwelche Probleme auftauchen, ist das zu betrachtende Zertifikat gültig und der Besitzer damit identifiziert.

5.3.2. Signaturen

Neben der eben vorgestellten *Teilnehmerauthentifizierung* gibt es auch noch die *Nachrichtenauthentifizierung*, wobei man hierbei von *Signaturen* oder *digitalen Signaturen* spricht. Dabei wird aus den zu signierenden Daten und dem privaten Signaturschlüssel mithilfe eines Verfahrens, zum Beispiel NSS oder NTRUSign, die Signatur berechnet. Ein wichtiger Aspekt dabei ist, dass verschiedene Daten mit einer an Sicherheit grenzenden Wahrscheinlichkeit zu einer anderen Signatur führen müssen. Jeder, der den öffentlichen Signaturschlüssel hat, kann nun mit diesem überprüfen, ob die Signatur und die signierte Nachricht zusammenpassen. Wichtig bei einem Signaturverfahren, wie auch bei Public Key-Verfahren, ist, dass es praktisch nicht möglich sein soll, dass man aus dem öffentlichen Signaturschlüssel den privaten erhalten kann. Somit ist der Aussteller der Signatur eindeutig identifizierbar, vor allem, wenn man den öffentlichen Schlüssel mithilfe einer Public Key Infrastructure eindeutig zuordnen kann.

Bemerkung 5.17. Da viele Signaturverfahren sehr ähnlich ablaufen wie die dazugehörigen Public Key-Verschlüsselungsverfahren, spricht man beim Signieren auch häufig vom Verschlüsseln und beim Verifizieren vom Entschlüsseln. Ebenso wird der private Signaturschlüssel als *Private Key* und der private Signaturschlüssel als *Public Key* bezeichnet.

Eine große Gefahr für Signaturverfahren ist der *No-Message-Angriff*. Dieser läuft folgendermaßen ab: Es gibt drei Kommunikationspartner A, B und C, wobei A einen Public Key zum Signieren hat. Nun erfindet C eine Nachricht und behauptet, diese sei von A signiert worden und schickt sie an B. Dieser versucht daraufhin, die Signatur zu entschlüsseln, um an die ursprüngliche Nachricht m zu kommen. Wenn jetzt m eine sinnvolle Nachricht ist, konnte A eine Signatur der Nachricht m untergeschoben werden.

Beispiel 5.18. C möchte vom Konto von A Geld abheben. Dabei weiß er, dass er nur eine Nachricht an den Geldautomaten schicken muss, in der der Betrag steht. Natürlich muss diese Nachricht von A signiert werden, wobei ich als Signaturprüfung annehme, dass der Geldautomat mit Hilfe des Public Keys (a, b) die Nachricht s so prüft, ob $m = s^b \pmod a$ ein sinnvolles Ergebnis ergibt. Somit schickt C die Nachricht $s = 55$ an den Geldautomaten. Der Automat kennt den Public Key $(167, 7)$ von A und berechnet damit $m = 55^7 \pmod{167} = 159$. Da die Signaturprüfung ein sinnvolles Ergebnis ergibt, kann der Geldautomat annehmen, dass A die Anweisung, 159 Euro abzuheben, signiert hat. Tatsächlich aber hat C sich diese Nachricht nur ausgedacht.

Um diesem Problem zu begegnen und um zu gewährleisten, dass immer Nachrichten gleicher Länge signiert werden, muss eine Nachricht vor dem Signieren zuerst durch eine *Hashfunktion* modifiziert werden.

[18] Certification Authority-Zertifikat

Definition 5.19. Eine *Hashfunktion* ist ein Funktion, die eine Zeichenfolge beliebiger Länge auf den *Hashwert*, eine Zeichenfolge mit fester Länge, abbildet. Dabei sollte die *Hashfunktion* eine Einwegfunktion und *kollisionsfrei* sein. Eine Hashfunktion h heißt *kollisionsfrei*, wenn für alle Eingaben $x \neq x'$, gilt $h(x) \neq h(x')$.

Mit Hilfe einer Hashfunktion läuft ein allgemeiner Signaturvorgang folgendermaßen ab.

Signierung Sei m die Nachricht und h eine öffentliche Hashfunktion. Zuerst wird der Hashwert $h(m)$ berechnet und anschließend das Signaturverfahren auf $h(m)$ angewendet, so dass eine Signatur x entsteht. Abschließend wird x an die Nachricht m angehängt.

Verifikation Wie auch schon bei der Signierung wird zuerst der Hashwert $h(m)$ berechnet und anschließend die Signaturverifikation auf die Signatur x angewendet, die einen Wert $s(x)$ ergibt. Wenn die Signatur nicht gefälscht ist, gilt am Ende $h(m) = s(x)$.

Die Anwendung einer Hashfunktion hat zwei Vorteile:

1. Eine sehr lange Nachricht muss nicht aufgeteilt werden. Somit braucht man pro Nachricht wirklich nur eine Signatur.

2. Ein No-Message-Angriff ist nicht mehr möglich, da es bei einer guten Hashfunktion h praktisch unmöglich sein sollte, zwei Nachrichten m_1 und m_2 zu finden, für die gilt: $h(m_1) = h(m_2)$.

Der Nachteil bei der Verwendung von Hashfunktionen liegt darin, dass man ein zusätzliches Verfahren hat, um dessen Sicherheit man sich Gedanken machen muss. Dies soll jedoch nicht Teil dieser Arbeit sein, da ein solcher Angriff nicht ein spezielles Signaturverfahren, zum Beispiel NTRUSign, betreffen würde, sondern einen überwiegenden Teil der Public Key-Signaturverfahren.

6. NTRU und darauf aufbauende Kryptosysteme

1996[1] stellten Jeffrey Hoffstein, Jill Pipher und Joseph H. Silverman ein neues Kryptosystem vor: NTRU, was ausgeschrieben *Number Theory Research Unit*[2] heißt. Motivation der drei amerikanischen Mathematiker von der Brown University war die bis dahin sehr schlechte Effizienz asymmetrischer Verschlüsselungsverfahren.

	NTRU	RSA	McElice und GGH
Verschlüsseln	$O(N^2)$	$O(N^2)$	$O(N^2)$
Entschlüsseln	$O(N^2)$	$O(N^3)$	$O(N^2)$
Public Key	$O(N)$	$O(N)$	$O(N^2)$
Private Key	$O(N)$	$O(N)$	$O(N^2)$

Abbildung 6.1.: Vergleich der Geschwindigkeit verschiedener Public Key-Verfahren mit kleinem Sicherheitsparameter N

Wie man aus Abbildung 6.1[3] ablesen kann, kodiert RSA in der Zeit $O(N^2)$ unter Verwendung kurzer Verschlüsselungsexponenten und dekodiert in $O(N^3)$, wobei bei der Verschlüsselung bei langen Exponenten auch die Zeit $O(N^3)$ benötigt wird. Die in der Entschlüsselung deutlich schnelleren Verfahren McElice oder GGH[4] brauchen unpraktikabel große Public und Private Keys der Länge $O(N^2)$. Dabei repräsentiert N den Sicherheitsparameter, dessen Größe der theoretischen Sicherheit entspricht, wobei zum Beispiel $N = 503$ bei NTRU einer 1024 Bit-Verschlüsselung bei RSA entspricht und somit $N = 1024$ für RSA gilt. Das 1996 fertig gestellte NTRU verband die Vorteile der bisherigen Public Key-Verfahren, indem es eine schnelle Ver- und Entschlüsselung bei gleichzeitig kurzem Schlüssel ermöglicht. Somit ist das NTRU-Verfahren gerade für mobile Geräte oder Embedded Systems geeignet, bei denen Geschwindigkeit und niedriger Speicherverbrauch sehr wichtig sind.[5]

Um diese Vorteile zu erreichen, wurden als Basis von NTRU nicht die traditionellen Probleme wie Faktorisierung oder der diskrete Logarithmus verwendet, sondern NTRU sollte nur auf Polynomrechnungen[6] basieren. Diesen Ansatz mussten die Entwickler je-

[1] vgl. [NTR09]
[2] vgl. [Ker00]
[3] aus [HPS98]
[4] Goldreich Goldwasser Halevi
[5] vgl. [May99a]
[6] vgl. [HPS96b]

doch wieder überarbeiten, als Don Coppersmith und Adi Shamir[7] das Verfahren gebrochen haben. Heute basiert NTRU zusätzlich auf einem Gitterproblem.

Basierend auf NTRU wurden die Signaturverfahren NSS, R-NSS und NTRUSign entwickelt. Bei dem auf NTRU basierenden Verschlüsselungsverfahren NTRUEncrypt ist die größte Schwierigkeit, während der Entschlüsselung die Korrektheit zu garantieren. Nachdem bei den Signaturverfahren während der Verifikation eine genaue Berechnung nicht möglich ist, muss man sicherstellen, dass eine gefälschte oder zufällig erstellte Signatur als solche erkannt wird. Dafür wurden mit der Zeit immer mehr und bessere Tests entwickelt, jedoch musste immer darauf geachtet werden, dass die Tests auch effizient durchgeführt werden können, um die Geschwindigkeitsvorteile gegenüber anderen Verfahren zu erhalten.

Im Kapitel 6.1 werde ich zuerst einige Definitionen geben und Begriffe erklären, die ich im Folgenden benutzen werde. Dann werde ich in Kapitel 6.4 die Grundzüge von NTRU beschreiben und damit zeigen, worauf das Kryptosystem NTRUEncrypt basiert und im folgenden Kapitel 6.5 einige praktische Verbesserungen[8] aufzeigen. Anschließend stelle ich die Signaturverfahren vor und erläutere, in welchen Punkten sie sich unterscheiden.

6.1. Einführung

Das NTRU-Verfahren operiert auf einem Polynomring $R := {}^{\mathbb{Z}[X]}/_{(X^N-1)}$. Somit sind sowohl die Nachrichten als auch alle Schlüssel Polynome aus diesem Polynomring. Ich werde im Folgenden davon ausgehen, dass alle Nachrichten schon in geeigneter Weise in Polynome kodiert worden sind.

Definition 6.1. Sei q eine natürliche Zahl. Dann ist $\mathbb{Z}_q := {}^{\mathbb{Z}}/_{q\mathbb{Z}}$ der *Restklassenring modulo q*. Über $\mathbb{Z}_q$ lässt sich sich der *Polynomring* $R_q := {}^{\mathbb{Z}_q[X]}/_{(X^N-1)}$ definieren. Dabei bezeichnet N den *Sicherheitsparameter*, der den maximalen Grad der Polynome bestimmt.

Bemerkung 6.2. Im Folgenden sind die Koeffizienten der Polynome in R_q aus einem um die Null zentrierten Intervall. Genauer gilt $\mathbb{Z}_q := \left[-\left\lfloor\frac{q-1}{2}\right\rfloor, \left\lceil\frac{q-1}{2}\right\rceil\right]$. Dabei bezeichnet $\lfloor x\rfloor$ die größte ganze Zahl $\leq x$ und $\lceil x\rceil$ die kleinste ganze Zahl $\geq x$.

Diese Verschiebung des Intervalls garantiert, dass bei der Multiplikation zweier Polynome mit betragsmäßig kleinen Koeffizienten das Produktpolynom ebenfalls betragsmäßig kleine Koeffizienten hat.

Bemerkung 6.3. Der Polynomring R_q lässt sich auch anders einführen. Sei $\mathbb{Z}[X]$ der Polynomring über $\mathbb{Z}$. Dann existiert in $\mathbb{Z}[X]$ ein Ideal $I_q := <q, X^N - 1>$. Für den Polynomring gilt nun $R_q = {}^{\mathbb{Z}[X]}/_{I_q}$.
Weiterhin existiert in $\mathbb{Z}[X]$ ein *kanonisches Repräsentantensystem von* R_q

$$R'_q := \left\{ f(X) \in \mathbb{Z}[X] \;\middle|\; \deg(f) < N \wedge -\left\lfloor\frac{q-1}{2}\right\rfloor \leq f_i \leq \left\lceil\frac{q-1}{2}\right\rceil \text{ für } 0 \leq i \leq N-1 \right\}$$

[7] vgl. [CS97]
[8] aus [HS00]

Im Folgenden werde ich die Definitionen von R_q und R'_q synonym verwenden und beide Mengen mit R_q bezeichnen. Vor allem aber in Kapitel 6.4.4 betrachte ich R'_q als Raum, aus dem die Polynome kommen, da sonst ein Übergang von R_q nach R_p mit $q \neq p$ nicht definiert wäre. Im Raum R'_q besteht dieses Problem nicht, da ich das Polynom unverändert in $\mathbb{Z}[X]$ lasse und nur einen anderen Repräsentanten wähle.

Definition 6.4.

- Sei $f \in R_q$ ein *Polynom*, das heißt, $f = \sum_{i=0}^{N-1} f_i X^i$ mit $f_i \in \mathbb{Z}_q$. Wir können f auch mit dem *Koeffizientenvektor* $f = (f_0, f_1, \ldots, f_{N-1})$ identifizieren. Beide Schreibweisen, f als Koeffizientenvektor oder als Polynom, werden im Folgenden synonym verwendet, je nachdem, welche Schreibweise gerade passender ist.

- Seien f und g zwei Polynome vom Grad N als Koeffizientenvektor gegeben. Dann bezeichnet (f, g) die *vektorielle Konkatenation* der beiden Polynome, das heißt, (f, g) ist ein $2N$-Vektor.

- Das *Produkt zweier Polynome* $f, g \in R_q$ ist $f \circledast g = h \in R_q$, wobei für den k-ten Koeffizienten von h gilt:

$$h_k = \sum_{i=0}^{k} f_i g_{k-i} + \sum_{i=k+1}^{N-1} f_i g_{N+k-i} = \sum_{\substack{i+j \equiv k \\ (\mathrm{mod}\ N)}} f_i g_j \pmod q$$

  Das *Produkt zweier Polynome* nennt man auch *Konvolution*[9].

- Sei $a \in R$ ein zufällig gewähltes Polynom. Dann sei M_a eine $N \times N$-Matrix, bei der die Einträge an der Position (i, j) gleich $a_{(j-i) \pmod N}$ sind. Diese Matrix wird *Konvolutionsmatrix* genannt. Damit kann das Produkt von zwei Polynomen a und b auch als Produkt des Zeilenvektors $(a_0, a_1, \ldots, a_{N-1})$ mit der Matrix M_b angesehen werden.

Bemerkung 6.5. [10] Normalerweise ist die Berechnung des Produkts $f \circledast g$ in der Komplexitätsklasse $O(N^2)$, da N^2 Multiplikationen gebraucht werden. Bei den Polynomen, die beim NTRU-Verfahren benutzt werden, hat typischerweise entweder f oder g kleine Koeffizienten, so dass die Berechnung von $f \circledast g$ trotzdem sehr schnell geht. Sollte N zu groß werden, kann man mit Hilfe der schnellen Fourier-Transformation die Komplexität auf $O(N \log N)$ drücken. Weiterhin werden in Kapitel 6.5 weitere Methoden vorgestellt, mit deren Hilfe man die Effizienz der Polynommultiplikation noch weiter verbessern kann.

Definition 6.6. Mithilfe der Identifikation eines Polynoms mit einem Vektor kann man die *zentrierte Norm* eines Polynoms f folgendermaßen definieren:

$$||f||_z := \sqrt{\sum_{i=0}^{N-1} f_i^2 - \frac{1}{N}\left(\sum_{i=0}^{N-1} f_i\right)^2}$$

[9] vgl. [Hac08]
[10] vgl. [HPS98]

Weiterhin ist noch die *Weite* eines Polynoms f definiert als

$$||f||_w := \max_{0 \leq i < N} \{f_i\} - \min_{0 \leq i < N} \{f_i\}$$

Abschließend werde ich die *euklidische Norm* verwenden, die als

$$||f||_2 := \sqrt{\sum_{i=0}^{N-1} f_i^2}$$

definiert ist.

Definition 6.7. Der *Schlüsselraum* von NTRU ist eine Menge

$$\begin{aligned} K(r,s) := \{f \in R \mid\ & f_i \in \{-1,0,1\} \text{ für } 0 \leq i < N \wedge \\ & \#\{f_i \text{ mit } f_i = 1\} = r \wedge \\ & \#\{f_i \text{ mit } f_i = -1\} = s\} \end{aligned}$$

Für $p = 3$ gilt somit $K(r,s) \subseteq R_p$, was auch dem Normalfall in den hier in der Arbeit betrachteten Verfahren ist. Im Spezialfall $p = 2$ beziehungsweise $p = 2+X$, wie in Kapitel 6.5.2 eingeführt, ändert sich die Definition von $K(r,s)$, so dass nun gilt

$$\begin{aligned} K(r,s) := \{f \in R \mid\ & f_i \in \{0,1\} \text{ für } 0 \leq i < N \wedge \\ & \#\{f_i \text{ mit } f_i = 1\} = r\} \end{aligned}$$

In diesem Fall ist der Parameter s bedeutungslos. Ich werde ihn aber aus Konsistenzgründen mitführen.

Bemerkung 6.8. Wir nehmen an, dass alle Schlüssel innerhalb $K(r,s)$ mit gleicher Wahrscheinlichkeit ausgewählt werden. Weiterhin ist im Allgemeinen $r = s$ oder $r = s \pm 1$, da diese Wahl die höchste Sicherheit bietet und die Effizienz nur von der Summe $r + s$ abhängt.[11] Weiterhin garantiert $r = s \pm 1$, dass die daraus gewählten Polynome mit hoher Wahrscheinlichkeit invertierbar[12] in R_q und R_p sind.

Definition 6.9. Ein Polynom $f \in R_q$ heißt *klein*, falls alle Koeffizienten f_i des Polynoms f im Intervall $[-p-1, p+1]$ liegen. Ist p, wie in Kapitel 6.5.2 eingeführt, ein Polynom, so bezeichne $p_{\max}$ den größten Koeffizienten von p. Dann heißt ein Polynom f klein, wenn alle Koeffizienten f_i im Intervall $[-p_{\max} - 1, p_{\max} + 1]$ liegen.

Bemerkung 6.10. Ist ein Polynom aus $K(r,s)$, so ist es automatisch *klein*.

Definition 6.11. Eine Nachricht heißt *schwach kodiert*, falls das Polynom, in das die Nachricht kodiert wurde, eine zu geringe Anzahl von Koeffizienten ungleich Null aufweist, das heißt, wenn mehr als $\frac{3}{4}N$ Koeffizienten gleich Null sind. Ist dies der Fall, so kann ein direkter Angriff auf die Nachricht selbst erfolgen.

[11] vgl. [HHHGW09]
[12] vgl. Definition 6.12

Definition 6.12. Ein Polynom $f_q^{-1} \in R_q$ heißt das *Inverse* eines Polynoms f, wenn $f \circledast f_q^{-1} \equiv 1 \pmod{q}$ gilt. Das Polynom f heißt dann *invertierbar* in R_q oder auch *invertierbar* bezüglich q.

Während einiger Berechnungen brauchen wir das Inverse des Polynoms f. Jedoch ist im Allgemeinen der Restklassenring R_q kein Körper und es existiert nicht für jedes Polynom ein Inverses. Daher muss man bei der Auswahl von f prüfen, ob solch ein f_q^{-1} existiert und gegebenenfalls ein anderes f nehmen. Für Polynome $f \in K(d_f, d_f - 1)$ ist die Wahrscheinlichkeit, ein invertierbares Polynom in R_q zu finden, sehr hoch[13], und das Auffinden eines geeigneten Polynoms sollte kein Problem für die Effizienz des Verfahrens sein. Für $N = 101$ liegt die Wahrscheinlichkeit schon bei 98%, für größere N steigt die Wahrscheinlichkeit noch an.[14]

Definition 6.13. Seien f und g zwei Polynome vom Grad $\leq N - 1$ und p und q teilerfremd. Dann ist die *Abweichung* $\mathrm{Dev}(f, g)$ definiert als die Anzahl der Koeffizienten, bei denen sich $(f \pmod{q}) \pmod{p}$ von $(g \pmod{q}) \pmod{p}$ unterscheiden.

Bemerkung 6.14. [15] Wenn man annimmt, dass $p = 3$ gilt, so liegt die Wahrscheinlichkeit, dass $f_i \not\equiv g_i \pmod{p}$ bei etwa $\frac{2}{3}$, so dass man erwarten kann, dass auch $\mathrm{Dev}(f, g)$ in der Größenordnung $\frac{2}{3}N$ ist.

Definition 6.15. Sei f ein Polynom. Dann heißt $\overline{f}$ eine *Umkehrung* von f, wenn gilt $\overline{f_0} = f_0$ und $\overline{f_k} = f_{N-k}$ für alle k.

Bemerkung 6.16. Die Abbildung $f \to \overline{f}$ ist ein Automorphismus in R, da gilt $\overline{\overline{f}} = f$.

Definition 6.17. Ein *Palindrom* ist ein Polynom in R, für das gilt $\overline{f} = f$, das heißt, dass es durch die Umkehrung nicht verändert wird.

Bemerkung 6.18. Für jedes Polynom $f \in R$ gilt, dass das Produkt $f \circledast \overline{f}$ ein Palindrom ist, da die Konvolution kommutativ ist.

6.2. Parameterwahl

Öffentliche Parameter

- Der *Sicherheitsparameter* N. Dieser bestimmt den maximalen Grad minus Eins der Polynome im Ring R_q. Üblicherweise ist N eine Primzahl.
- Zwei *Moduln* p und q, die die Polynomringe R_p und R_q definieren. Der Modul q ist normalerweise eine Potenz von 2 und liegt im Intervall $(\frac{N}{2}, N)$. Die Wahl von q als eine Potenz von 2 liegt darin begründet, dass man mit einem so beschaffenen Modul q in R_q sehr effektiv das inverse Polynom berechnen kann. Der Modul p ist teilerfremd zu q und sollte so klein wie möglich gewählt

[13] vgl. [Sil98]
[14] vgl. [Rüc07]
[15] vgl. [GJSS01]

	N	q	p
Mittlere Sicherheit	167	128	3
Normale Sicherheit	251	128	3
Hohe Sicherheit	347	128	3
Höchste Sicherheit	503	256	3

Abbildung 6.2.: Sicherheitsabstufungen für die drei wichtigen Parameter von NTRU

werden, so dass häufig die Wahl auf $p = 3$ fällt. Somit erreicht man, dass Polynome aus R_p klein sind und man kann den Schlüsselraum $K \subseteq R_p$ wählen. Wichtig ist dies, da man innerhalb des Verfahrens garantieren muss, dass die verwendeten Polynome so klein wie möglich sind, damit die Entschlüsselung fehlerfrei abläuft. In Kapitel 6.5 werden eine andere Wahl von p vorgestellt und deren Vor- und Nachteile diskutiert.

Aus diesen Angaben lassen sich die beiden Polynomringe R_p und R_q bestimmen, aus denen dann die Schlüssel gewählt werden. Für diese Parameter gibt es von den Entwicklern eine Übersicht, in welcher Größenordnung die einzelnen Zahlen gewählt werden müssen, um eine ausreichende Sicherheit zu gewährleisten. Diese Übersicht ist in Abbildung 6.2[16] abgebildet. Dabei sollte man beachten, dass sich diese Empfehlungen mit der Zeit ändern können, da Computer immer leistungsfähiger werden und man somit immer längere Schlüssel braucht, um eine gleichbleibende Sicherheit zu gewährleisten.

- Die *Schlüsselraumparameter* d_f, d_g, d_m und d_z. Diese bestimmen die Schlüsselräume, die in dem in Kapitel 6.4 beschriebenen Verfahren benötigt werden. Während des Verfahrens ist es wichtig, dass die verwendeten Polynome alle klein sind, damit es bei der Entschlüsselung zu so wenig Fehlern wie möglich kommt. Wie schon in Bemerkung 6.10 eingeführt, garantiert die Wahl eines Polynoms aus $K(r, s)$, dass das Polynom klein ist. Deshalb werden während des Verfahrens alle Polynome, die nicht berechnet werden, aus $K(r, s)$ gewählt. Die Parameter r und s entsprechen nun den Schlüsselraumparametern, wobei wie in Bemerkung 6.8 angemerkt, $r = s$ oder $r = s \pm 1$ gilt.

  Im späteren Verlauf der Arbeit, in der ich Modifikationen und Weiterentwicklungen des NTRU-Verfahrens vorstelle, kommen noch weitere Schlüsselraumparameter hinzu. Diese werden immer mit d_ϕ bezeichnet werden, wobei ϕ ein Polynom ist, bei dem garantiert werden soll, dass es klein ist.

Die oben genannten öffentlichen Parameter können entweder frei gewählt beziehungsweise berechnet werden oder man benutzt vorgefertigte Parametersätze. Dabei muss man jedoch immer zwischen Sicherheit und Effizienz abwägen, da eine falsche Wahl eines Parameters oder mehrerer Parameter die Sicherheit des ganzen Verfahrens gefährden kann.[17]

[16] aus [NTR07]
[17] vgl. [Ngu06]

Private Key

- Ein in R_q und in R_p invertierbares Polynom $f \in K(d_f, d_f - 1)$. Sollte f in einem der beiden Polynomringe kein Inverses besitzen, so muss f neu gewählt werden.
- Das inverse Polynom $f_q^{-1} \in R_q$ von f, das zur Berechnung des Public Key benötigt wird.
- Das inverse Polynom $f_p^{-1} \in R_p$ von f, das bei der Entschlüsselung benötigt wird.
- Ein Polynom $g \in K(d_g, d_g)$, das zur Berechnung des Public Key h verwendet wird und dabei das Polynom f verschleiern soll.

Public Key

- Ein Polynom $h \in R_q$, dessen genaue Berechnung in den einzelnen NTRU-Versionen unterschiedlich ist.

Sonstiges

- Die Klartextnachricht $m \in K(d_m, d_m)$, die bei den Signaturverfahren vor der Verschlüsselung eine Hashfunktion durchlaufen muss. Dabei wird durch den Parameter d_m verhindert, dass m schwach kodiert wird.
- Ein zufällig gewähltes Polynom $z \in K(d_z, d_z)$, das bei der Verschlüsselung dafür sorgt, dass jede Klartextnachricht in einen anderen Geheimtext chiffriert wird.

In den Modifikationen von NTRU werden noch weitere Polynome benötigt. Diese werde ich an der Stelle ihres Auftretens definieren.

6.3. Zugrundeliegendes Problem

Definition 6.19. Sei $h \in R_q$ ein Polynom. Dann heißen die Polynome $f, g \in R_q$ eine *Faktorisierung* von h in R_q, wenn sie $f \circledast g \equiv h \pmod q$ erfüllen.

Satz 6.20. *Es gibt $|R_q^*|$ Möglichkeiten, ein Polynom $h \in R_q$ zu faktorisieren, wobei R_q^* die Menge der invertierbaren Polynome aus R_q ist.*

Beweis. Nach Voraussetzung ist ein Polynom $f \in R_q^*$ invertierbar. Setze nun $g \equiv f \circledast h \pmod q$. Dann ist $h \equiv f^{-1} \circledast g \pmod q$ und damit f^{-1} und g eine Faktorisierung von h. Da die Anzahl der Polynome $f \in R_q$ der Anzahl $|R_q^*|$ entspricht, gibt es auch $|R_q^*|$ Möglichkeiten für eine Faktorisierung von h. □

In Kapitel 6.1 habe ich begründet, dass die Wahrscheinlichkeit, ein invertierbares Polynom in R_q zu finden, sehr hoch ist. Daher kann man annehmen, dass es etwa q^N Faktorisierungen gibt, was hochgradig uneindeutig ist. Bei den auf NTRU basierenden Verfahren muss man jedoch noch eine zusätzliche Eigenschaft bedenken. Die Polynome f und g sind jeweils aus dem Schlüsselraum $K(r, s)$ und daher sind die Koeffizienten von

f und g auf $\{-1, 0, 1\}$ beziehungsweise auf $\{0, 1\}$ beschränkt und f und g haben eine kleine Norm.

Daraus lässt sich ein auf NTRU zugeschnittenes *Polynomfaktorisierungsproblem* definieren.

Behauptung 6.21 (Polynomfaktorisierungsproblem)**.** [18] *Sei $h = f_q^{-1} \circledast g \in R_q$ ein Polynom, wobei f und g kleine Koeffizienten haben. Für hinreichend großes N ist es schwer, eines der Polynome f oder g aus h zu rekonstruieren oder eine Faktorisierung f', g' von h zu finden, mit f' invertierbar in R_q und $||(f', g')||_z \leq ||(f, g)||_z$.*

Ob das Polynomfaktorisierungsproblem wirklich schwer zu lösen ist, ist bis heute noch nicht wirklich geklärt. Alle Experimente weisen darauf hin, jedoch fehlt bisher ein Beweis oder Gegenbeweis. Deshalb gibt es auch bisher noch keinen Algorithmus, der dieses Problem in polynomialer Laufzeit lösen kann. Somit entspricht das Polynomfaktorisierungsproblem dem RSA zugrundeliegenden Problem der Faktorisierung natürlicher Zahlen. Dieses ist jedoch in Anwesenheit von Quantencomputern unsicher, da damit nichttriviale Teiler von natürlichen Zahlen in polynomialer Laufzeit gefunden werden können.[19]

Ist das Polynomfaktorisierungsproblem jedoch wider Erwarten effizient zu lösen, so wäre NTRU gebrochen und alle darauf aufbauenden Systeme unsicher. Unabhängig von dieser Vermutung gibt es aber verschiedene andere Methoden, NTRU zu brechen.

6.4. Grundlegende Ideen von NTRU

Im Folgenden werde ich die Grundzüge, also die Schlüsselerzeugung, die Verschlüsselung und die Entschlüsselung, des NTRU-Kryptosystems vorstellen und einen Korrektheitsbeweis der Entschlüsselung geben. Dabei werde ich mich jedoch nicht auf das ursprüngliche Kryptosystem beziehen, dass 1996 im Rahmen der Rump Session[20] vorgestellt wurde, da dieses noch Mängel enthielt, die von Coppersmith und Shamir[21] aufgedeckt wurden. Ich werde mich auf die momentan empfohlene Implementierung für NTRU[22] beziehen. Baut man die in Kapitel 6.5 vorgestellten Verbesserungen mit ein, so beschreibt dies eine der Versionen des Kryptosystems NTRUEncrypt.

6.4.1. Schlüsselerzeugung

Die öffentlichen Parameter werden wie in 6.2 festgelegt. Dann werden die zwei Polynome $f \in K(d_f, d_f - 1)$ und $g \in K(d_g, d_g)$ zufällig gewählt und $h \in R_q$ mit Hilfe von

$$h \equiv p \cdot g \circledast f_q^{-1} \pmod{q}$$

berechnet.

[18] vgl. [May99a]
[19] vgl. [Sho97]
[20] vgl. [HPS96b]
[21] vgl. [CS97]
[22] vgl. [NTR07]

Dabei muss gewährleistet sein, dass f bezüglich p und q invertierbar ist. Ist dies nicht der Fall, muss man das Polynom f neu wählen. Weiterhin ist wichtig, dass die Polynome f und g *klein* sind, was aber durch die Wahl der Polynome aus den Mengen $K(d_f, d_f - 1)$ und $K(d_g, d_g)$ garantiert wird.

Für die Berechnung der inversen Polynome f_p^{-1} und f_q^{-1} kann man die zwei auf Seite 112 abgedruckten Algorithmen Prozedur 1 und Prozedur 2 verwenden. Der Algorithmus Prozedur 1 berechnet im Polynomring R_s das Inverse eines gegebenen Polynoms a für eine Primzahl s und Prozedur 2 berechnet entsprechend im Polynomring R_{s^r} das Inverse von a für eine Primzahlpotenz s^r. Da im Standardfall $p = 3$ gilt und q als eine Zweierpotenz gewählt wird, können diese effizienten Algorithmen für die Berechnung im NTRU-Verfahren angewendet werden.[23]

Der Private Key besteht nun aus den beiden Polynomen f und g und h ist der Public Key.

Beispiel 6.22. Sei $N = 11$, $q = 32$ und $p = 3$. Weiterhin seien $d_f = 4$ und $d_g = 3$. Die für den Private Key zufällig gewählten Polynome seien

$$g = -1 + X^2 + X^3 + X^5 - X^8 - X^{10}$$
$$f = 1 - X - X^2 + X^3 + X^6 + X^7 - X^{10}$$

Die Inversen f_p^{-1} und f_q^{-1} von f werden nun mit den oben angegebenen Algorithmen berechnet.

$$f_p^{-1} \equiv 1 - X + X^2 + X^3 + X^5 - X^6 - X^7 - X^8 - X^9 - X^{10}$$
$$f_q^{-1} \equiv -11 - 12X - 6X^2 + 3X^3 + 2X^4 - 5X^6 - 12X^7 + 4X^8 + 3X^9 + 3X^{10}$$

Mit diesen Angaben wird der Public Key durch $h \equiv p \cdot g \circledast f_q^{-1} \pmod q$ berechnet:

$$h \equiv 2 - 2X - 3X^2 + 4X^3 - 15X^4 - 7X^5 - 11X^6 + 3X^7 + 6X^8 + 5X^9 - 14X^{10} \pmod{32}$$

6.4.2. Verschlüsselung

Sei $m \in K(d_m, d_m)$ die Klartextnachricht. Weiterhin wird für jeden Verschlüsselungsvorgang zufällig ein Polynom $z \in K(d_z, d_z)$ gewählt, um den No-Message-Angriff[24] ins Leere laufen zu lassen. Die Geheimtextnachricht c berechnet sich nun aus

$$c \equiv z \circledast h + m \pmod q$$

Beispiel 6.23 (Fortsetzung von Beispiel 6.22). Seien die Parameter und Schlüssel wie in Beispiel 6.22 gegeben. Weiterhin sei $d_z = 3$ und $d_m = 5$ und das zufällige Polynom

$$z = -1 + X^2 + X^3 + X^5 - X^6 - X^9$$

[23] vgl. [Sil99]
[24] vgl. Kapitel 5.3.2

Soll jetzt die Nachricht

$$m = 1 - X + X^2 - X^3 + X^4 - X^5 + X^6 - X^7 + X^8 - X^9$$

verschlüsselt werden, wird dafür $c \equiv z \circledast h + m \pmod{q}$ berechnet.

$$\begin{aligned} c \equiv\ & 9 + 2X + 10X^2 + X^3 + 3X^4 - 12X^5 - \\ & 9X^6 - 2X^8 - 3X^9 + X^{10} \pmod{32} \end{aligned}$$

6.4.3. Entschlüsselung

Zur Entschlüsselung muss das zufällig gewählte Polynom z aus dem Geheimtext c entfernt werden. Das Problem dabei ist, dass dieses Polynom nicht bekannt ist, da es nur zum Verschlüsseln einer Nachricht erzeugt und anschließend sofort wieder gelöscht wird. Um diesem Problem zu begegnen, multipliziert man den Geheimtext mit f und betrachtet das Ergebnis modulo q.

$$\begin{aligned} a &\equiv f \circledast c \pmod{q} \\ &\equiv f \circledast z \circledast h + f \circledast m \pmod{q} \\ &\equiv f \circledast z \circledast p \cdot g \circledast f_q^{-1} + f \circledast m \pmod{q} \\ &\equiv p \cdot z \circledast g + f \circledast m \pmod{q} \end{aligned}$$

$$\begin{aligned} /\!/ c &\equiv z \circledast h + m \pmod{q} \\ /\!/ h &\equiv p \cdot g \circledast f_q^{-1} \pmod{q} \\ /\!/ f \circledast f_q^{-1} &\equiv 1 \pmod{q} \end{aligned}$$

Weiterhin betrachtet man

$$b \equiv a \circledast f_p^{-1} \pmod{p}$$

Dies hat zwei Folgen:

1. Der Term $p \cdot z \circledast g$ wird durch die Modulorechnung gleich 0 gesetzt.
2. $f \circledast m \circledast f_p^{-1} \equiv m \pmod{p}$, da $f \circledast f_p^{-1} \equiv 1 \pmod{p}$

Somit erhält man am Ende $b = m$.

Ein Angreifer kann, wenn er die Nachricht c abfängt, diese ohne weitere Kenntnisse nicht entziffern. Da ihm das Polynom f nicht bekannt ist, kann er $a \equiv f \circledast c$ nicht berechnen. Weiterhin kennt er das Polynom z nicht, so dass auch eine Berechnung von $m \equiv c - z \circledast h$ nicht funktioniert.

Beispiel 6.24 (Fortsetzung von Beispiel 6.23). Seien die Parameter und Schlüssel wie in Beispiel 6.23 gegeben und der Geheimtext c gleich

$$\begin{aligned} c \equiv\ & 9 + 2X + 10X^2 + X^3 + 3X^4 - 12X^5 - \\ & 9X^6 - 2X^8 - 3X^9 + X^{10} \pmod{32} \end{aligned}$$

Daraus wird bei der Entschlüsselung $a = f \circledast c$ berechnet.

$$\begin{aligned} a \equiv\ & -2 - 10X - 10X^2 - 7X^3 + X^4 + X^5 + \\ & 11X^6 + 5X^7 + 10X^8 + X^{10} \pmod{32} \end{aligned}$$

Berechnet man nun mit diesem Polynom $b = a \circledast f_p^{-1} \pmod 3$, so ergibt sich

$$b \equiv 1 - X + X^2 - X^3 + X^4 - X^5 + X^6 - X^7 + X^8 - X^9 \pmod 3$$

was der ursprünglichen Klartextnachricht m entspricht.

6.4.4. Korrektheit der Entschlüsselung

Die Korrektheit der Entschlüsselung ist ohne Voraussetzungen keineswegs garantiert, da p und q teilerfremd sind und man beim Übergang von der Berechnung modulo q zu modulo p normalerweise falsche Ergebnisse erhält.

Behauptung 6.25. [25] *Für alle $\varepsilon > 0$ existieren Konstanten γ_1 und γ_2, die abhängig von N und ε sind, so dass mit einer Wahrscheinlichkeit größer als $1 - \varepsilon$ für zufällig gewählte Polynome $f, g \in R$ die folgende Abschätzung gilt:*

$$\gamma_1 \cdot ||f||_z \cdot ||g||_z \leq ||f \circledast g||_w \leq \gamma_2 \cdot ||f||_z \cdot ||g||_z$$

Diese Abschätzung ist nur dann hilfreich, wenn das Verhältnis der beiden Konstanten γ_1 und γ_2 klein bleibt. In Simulationen haben die Entwickler von NTRU ermittelt, dass das Verhältnis auch bei großen N und relativ kleinen ε in den meisten Fällen klein genug bleibt. Als Beispiel für $N = 107$, $N = 167$ und $N = 503$ geben sie die Werte $\gamma_2 = 0,35$, $\gamma_2 = 0,27$ und $\gamma_2 = 0,17$ an.[26]

Satz 6.26 (Entschlüsselungskriterium)**.** *Damit die Entschlüsselung fehlerfrei abläuft, ist es notwendig, dass*

$$||p \cdot z \circledast g + f \circledast m||_w < q$$

gilt.

Bemerkung 6.27. Die Ungleichung ist so gut wie immer erfüllt, wenn die Parameter so gewählt sind, dass

$$||f \circledast m||_w \leq \frac{q}{4} \qquad \text{und} \qquad ||p \cdot z \circledast g||_w \leq \frac{q}{4}$$

Benutzt man nun die Abschätzung der Weite der Polynome aus Behauptung 6.25, dann ergibt sich daraus

$$||f||_z \cdot ||m||_z \approx \frac{q}{4\gamma_2} \qquad \text{und} \qquad ||g||_z \cdot ||z||_z \approx \frac{q}{4p \cdot \gamma_2}$$

Die hier in diesem Kontext interessanten Polynome sind $f \in K(d_f, d_f - 1)$, $g \in K(d_g, d_g)$, $z \in K(d_z, d_z)$ und $m \in K(d_m, d_m)$. Durch die spezielle Struktur der Schlüsselräume kann man die zentrierte Norm dieser Polynome in Abhängigkeit von den Schlüsselparametern genau angeben. Dabei gilt

[25] vgl. [HPS98]
[26] vgl. [HPS98]

- $||f||_z = \sqrt{2d_f - 1 - N^{-1}}$
- $||g||_z = \sqrt{2d_g}$
- $||z||_z = \sqrt{2d_z}$
- $||m||_z = \sqrt{2d_m}$

Damit kann man die Schlüsselparameter so wählen, dass die obige Ungleichung fast immer erfüllt ist.

Obwohl Entschlüsselungsfehler bei geeigneter Parameterwahl sehr selten sind, können sie einen Angriff auf das Verfahren ermöglichen. Dazu werden von einem Angreifer gezielt Fehler provoziert. Hat man genügend Fehler gefunden, so kann man Rückschlüsse auf den Private Key f ziehen und damit auch die restlichen Polynome berechnen.[27] Daher muss an dieser Stelle noch weiter überprüft werden, wie man Entschlüsselungsfehler vermeiden kann oder wie verhindert werden kann, dass Entschlüsselungsfehler ausgenutzt werden können.

Bemerkung 6.28. In wenigen seltenen Fällen kann es vorkommen, dass einzelne Koeffizienten von a *nicht* im Intervall $(-\frac{q}{2}, \frac{q}{2}]$ liegen. Dann muss entweder der Klartext neu verschlüsselt werden, da somit mithilfe eines anderen zufälligen Polynoms z ein passenderes Ergebnis erzeugt wird, oder der komplette Verschlüsselungsvorgang muss mit anderen Parametern wiederholt werden.

6.4.5. NTRU als Gitterproblem

Bevor Don Coppersmith und Adi Shamir in ihrer Arbeit[28] versuchten, NTRU mit Hilfe von Gittertheorie anzugreifen, formulierten die Entwickler NTRU als ein Kryptosystem, das nur auf Polynomrechnungen basiert. Nachdem aber der Angriff der zwei Kryptologen erfolgreich durchgeführt werden konnte, wurde das Verfahren überdacht und man versuchte, das Kryptosystem auf gittertheoretische Grundlagen zu stellen. Die Basis für dieses Gitter ist von Coppersmith und Shamir entwickelt und wird daher mit L_{CS} bezeichnet und erfüllt

$$L_{\text{CS}} = \begin{bmatrix} I_{(N)} & M_h \\ 0 & qI_{(N)} \end{bmatrix},$$

wobei M_h die Konvolutionsmatrix des Public Keys h ist und $I_{(N)}$ die N-dimensionale Einheitsmatrix. Da die Matrix M_h im Vergleich zu den Schlüsselpolynomen f und g große Einträge hat, vermuten die Entwickler von NTRU, dass die Koeffizientenvektoren dieser Polynome die kürzesten Vektoren im Gitter L_{CS} sind. Somit wäre dies ein Beispiel für das SVP[29], welches im Allgemeinen nicht effizient gelöst werden kann.

Bei der Betrachtung von NTRU als Gitterproblem treten jedoch zwei Problemstellungen auf.

[27] vgl. [Ove04]
[28] vgl. [CS97]
[29] Shortest Vector Problem

1. Das SVP ist für allgemeine Gitter schwer lösbar. Jedoch ist das NTRU-Gitter L_{CS} kein allgemeines Gitter, sondern es hat besondere Eigenschaften.

2. Auch wenn das SVP ein schweres Problem ist, so sind die Verknüpfungen der einzelnen Rechenoperationen von NTRU mit dem SVP eventuell angreifbar.

Die Untersuchung dieser Problemstellungen ist momentan Gegenstand der kryptoanalytischen Forschung. In Kapitel 7 wird hauptsächlich der zweite Punkt Beachtung finden, da gerade die Signaturverfahren NSS und R-NSS an dieser Stelle noch Schwachstellen hatten.

6.5. Verbesserungen und praktische Implementierungen

6.5.1. Effiziente Polynommultiplikation und Sicherheitsüberlegungen

Wie schon in Definition 6.4 und Bemerkung 6.5 eingeführt, müssen für das Produkt zweier Polynome s und t vom Grad N im Allgemeinen $(N+1)^2$ Multiplikationen und $N \cdot (N+1)$ Additionen durchgeführt werden. Ist nun mindestens eines der beiden Polynome, oBdA s, ein kleines Polynom, so bestehen dessen Koeffizienten nur aus $\{-1, 0, 1\}$ oder unter Umständen, wie ich in Kapitel 6.5.2 zeigen werde, aus $\{0, 1\}$. Durch diese spezielle Wahl der Koeffizienten wird *keine* Multiplikation mehr benötigt und auch die Anzahl der Additionen ist reduziert. Wenn man annimmt, dass i Koeffizienten von s ungleich Null sind, dann ist bei der Polynommultiplikation die Anzahl der Additionen gleich $(i-1) \cdot (N+1)$.

Bemerkung 6.29. Im weiteren Verlauf der Arbeit werde ich aus Platzgründen die Polynome in Koeffizientenvektorschreibweise schreiben. Somit wird aus dem Polynom

$$1 + X^4 + X^6$$

der Vektor

$$[1, 0, 0, 0, 1, 0, 1]$$

Beispiel 6.30. Betrachten wir zuerst den allgemeinen Fall zweier quadratischer Polynome:

$$\begin{aligned}
[7, 1, 5] \circledast [3, 3, 6] &= 7 \circledast [3, 3, 6] + 1 \circledast [6, 3, 3] + 5 \circledast [3, 6, 3] \\
&= [21, 21, 42] + [6, 3, 3] + [15, 30, 15] \\
&= [21 + 6 + 15, 21 + 3 + 30, 42 + 3 + 15] \\
&= [42, 54, 60]
\end{aligned}$$

Da die Polynome zweiten Grades sind, müssen hier neun Multiplikationen und sechs Additionen durchgeführt werden.

Betrachten wir nun den Fall eines für NTRU typischen binären Polynoms mit einem beliebigen Polynom:

$$\begin{aligned}[1,1,0,0,1] \circledast [7,3,2,9,2] &= 1 \circledast [7,3,2,9,2] + 1 \circledast [2,7,3,2,9] + 0 \circledast [9,2,7,3,2] + \\ &\quad 0 \circledast [2,9,2,7,3] + 1 \circledast [3,2,9,2,7] \\ &= [7,3,2,9,2] + [2,7,3,2,9] + [3,2,9,2,7] \\ &= [7+2+3, 3+7+2, 2+3+9, 9+2+2, 2+9+7] \\ &= [12,12,14,13,18]\end{aligned}$$

Durch die geschickte Wahl der Koeffizienten wird komplett auf alle Multiplikationen verzichtet und da drei Koeffizienten ungleich Null sind, müssen auch nur $(3-1)\cdot(4+1) = 10$ Additionen durchgeführt werden.

Beim Festlegen der Parameter des NTRU-Verfahrens muss man eine Abwägung machen zwischen Sicherheit und Geschwindigkeit. Je mehr Koeffizienten der Polynome gleich Null sind, desto schneller können die Polynomrechnungen durchgeführt werden. Sind jedoch zu viele Koeffizienten gleich Null, so ist eine ausreichende Sicherheit aus folgenden Gründen nicht mehr gewährleistet.

- Wenn in f zu wenige Koeffizienten ungleich Null sind, so ist es per Brute Force möglich, einfach alle Möglichkeiten durchzuprüfen.
- Während der Schlüsselerzeugungsphase wird der Public Key h aus zwei kleinen Polynomen erzeugt. Daher ist es möglich, wie in Kapitel 6.4.5 eingeführt, mit Hilfe des SVP f und g zu rekonstruieren, zum Beispiel durch Gitterreduktion mit dem LLL[30]-Algorithmus. Jedoch hängt die Ausführungszeit von Gitterreduktionsalgorithmen unter anderem von folgenden Punkten ab:
  1. Der Dimension der Gitter: Je größer die Dimension, desto schwieriger ist ein Angriff.
  2. Der Beschaffenheit des kürzesten Vektors: Je kürzer der kürzeste Vektor ist, desto einfacher ist es, ihn zu finden.

Also kann man zusammenfassen, dass die für NTRU verwendeten Polynome zwar klein, aber nicht zu klein sein sollten, so dass man am Ende einen ausgeglichenen Kompromiss zwischen Sicherheit und Geschwindigkeit finden kann.

6.5.2. Der Parameter p

Im vorherigen Abschnitt habe ich angenommen, dass $p = 3$ ist, um zu garantieren, dass die verwendeten Polynome alle klein genug sind, so dass bei der Entschlüsselung die Reduzierung modulo q ohne Effekt bleibt. Jedoch müssen dafür die Parameter in geeigneter Weise gewählt werden. Könnte man p verkleinern und damit auch die die Norm[31] $||p \cdot z \circledast g + f \circledast m||_w$, so wäre eine korrekte Entschlüsselung bei weitaus mehr

[30] Lenstra Lenstra Lovász
[31] vgl. Kapitel 6.4.4

Parametern möglich. Da p und q jedoch teilerfremd sein sollen und q in den meisten Fällen eine Zweierpotenz ist[32], kann man p nicht auf 2 setzen. Es wird jedoch nirgendwo im Verfahren vorausgesetzt, dass p eine natürliche Zahl sein muss, so dass p auch als Polynom gewählt werden kann, zum Beispiel als $p = 2 + X$. Damit ist zum Einen garantiert, dass p und q teilerfremd bleiben und zum Anderen ermöglicht diese Wahl von p, dass man für die Polynome statt $\{-1, 0, 1\}$ als Koeffizienten $\{0, 1\}$ wählen kann. Dies beschleunigt das Verfahren, da die Polynome so in natürlicher Weise in Binärform sind und damit eine umständliche Umkodierung der Nachrichten in Polynome und umgekehrt entfällt und auch die Schlüsselpolynome für die Übertragung nicht umkodiert werden müssen.

Während der Entschlüsselung gibt es einen Übergang von der Berechnung modulo q zu modulo p. Da der Raum der Polynome R_p vom Ideal $I_p := <p, X^N - 1>$ abhängt, muss man weiterhin beachten, dass $X^N - 1$ und p teilerfremd sind, da sonst die Menge der möglichen Geheimtexte[33] zu klein wird und damit eine eindeutige Entschlüsselung nicht mehr möglich ist. Die Polynome $2 + X$ und $X^N - 1$ sind jedoch teilerfremd, so dass an dieser Stelle keine Probleme auftauchen. Das Ideal $I_p := <p, X^N - 1>$ kann dann auch in der Form $I_p := <2 + X, (-2)^N - 1>$ geschrieben werden, da

$$X^N - 1 \equiv (-2)^N - 1 \pmod{2 + X}$$

Der Nachteil, mit der diese Änderung von p erkauft wird, ist, dass die Entschlüsselung etwas komplizierter wird, da das Ergebnispolynom a nicht mehr korrekt um Null zentriert ist, so dass es einen Unterschied macht, ob man modulo q betrachtet oder nicht. Es ist somit im Gegensatz zur bisherigen Betrachtung deutlich weniger wahrscheinlich, dass die Entschlüsselung korrekt abläuft. Wenn man jedoch das Intervall voraussagen könnte, in dem die Koeffizienten von a liegen müssen, damit eine korrekte Entschlüsselung möglich ist, so könnte man die Koeffizienten per Modulorechnung in die richtigen Intervallgrenzen korrigieren. Leider ist demjenigen, der die Entschlüsselung durchführt, das Polynom m, also die Klartextnachricht, nicht bekannt und somit ist eine direkte Berechnung des Intervalls nicht möglich. Eine direkte Übertragung des nötigen Intervalls zusammen mit der Geheimtextnachricht ist auch nicht möglich, da demjenigen, der verschlüsselt, die Polynome f und g unbekannt sind.

Für das NTRU-Verfahren wurde deshalb eine Methode entwickelt, auch ohne Kenntnis der verwendeten Polynome Rückschlüsse auf das Intervall zu schließen. Dafür berechnet man

$$I := f_q^{-1}(1) \cdot (a(1) + p(1) \cdot z(1) \cdot g(1)) \pmod{q}$$
$$Avg := \frac{p(1) \cdot z(1) \cdot g(1) + I \cdot f(1)}{N}$$

Da Avg im Allgemeinen keine ganze Zahl ist, besteht das voraussichtliche Intervall aus den q ganzen Zahlen zwischen $Avg - \frac{q}{2}$ und $Avg + \frac{q}{2}$.

[32] Eine solche Wahl von q garantiert eine effiziente Berechnung des inversen Polynoms f_q^{-1}. In späteren Versionen von NTRUEncrypt ist q nicht mehr auf eine Zweierpotenz festgelegt, sondern q ist eine Primzahl. In diesem Fall kann man auch $p = 2$ wählen.

[33] vgl. Definition 2.5

Wird beim Polynom a darauf geachtet, dass alle Koeffizienten innerhalb des so neu berechneten Intervalls sind, so ist es sehr sicher, dass die Entschlüsselung korrekt verläuft.

Bemerkung 6.31. Sei l ein Polynom. Dann entspricht $l(1)$ dem Einsetzen von $X = 1$, also der Summe der Koeffizienten des Polynoms l. Bei den Polynomen f, g und z entspricht $f(1) = d_f$, $g(1) = d_g$ und $z(1) = d_z$, da die Koeffizienten bei unseren Polynomen nur aus 0 und 1 bestehen. Somit kann man diese Berechnung auch durchführen, ohne dass das zufällige Polynom z bekannt sein muss.

Weiterhin ist die Reduzierung modulo $2 + X$ ein wenig komplizierter. Es ist jedoch weitestgehend immer möglich, für ein gegebenes Polynom s vom Grad N und Koeffizienten modulo q ein Polynom t vom Grad N mit binären Koeffizienten zu finden, so dass $t(-2) \equiv s(-2) \mod 2^N + 1$. Die einzige Ausnahme tritt dann auf, wenn $s(-2) \equiv j + 1 \pmod{2^N + 1}$ mit

$$j := \begin{cases} 2^{N-2} + 2^{N-4} + \cdots + 2^{N-N} = \frac{2^N - 1}{3} & \text{falls } N \text{ gerade} \\ 2^{N-1} + 2^{N-3} + \cdots + 2^{N-N} = \frac{2^{N+1} - 1}{3} & \text{falls } N \text{ ungerade} \end{cases}$$

In diesem Fall setzt man nun

$$t(X) := \begin{cases} 2 + X^2 + X^4 + \cdots + X^N & \text{falls } N \text{ ungerade} \\ 2 + X^2 + X^4 + \cdots + X^{N-1} & \text{falls } N \text{ ungerade} \end{cases}$$

Jedoch ist t in diesem Fall kein binäres Polynom mehr, so dass die Entschlüsselung fehlschlägt. Tritt dies auf, so muss mit einem größeren Sicherheitsparameter N die Nachricht neu verschlüsselt werden.

Bemerkung 6.32. In der in dieser Arbeit verwendeten Literatur wird fast immer angenommen, dass $p = 3$ gilt. Daher werde ich ebenfalls im Folgenden annehmen, dass $p = 3$ gilt. In Abschnitt 6.5.4 wird jedoch einmal beispielhaft demonstriert, wie sich die eben beschriebene Änderung von p auswirken kann.

6.5.3. Der Private Key f und das Zufallspolynom z

Bei einer geschickten Wahl des Private Keys f kann man deutlich Rechenzeit sparen. Wählt man $f = 1 + p \circledast F$, mit F einem Polynom mit kleinen Koeffizienten, so kann man den Verbrauch an Rechenzeit und Speicherbedarf deutlich reduzieren.

- In der Schlüsselerzeugungsphase muss kein Inverses modulo p berechnet werden, da das Inverse $f_p^{-1} = 1$ ist. Somit spart man sich sowohl eine Inversenberechnung als auch den erforderlichen Speicherplatz des inversen Polynoms.
- Da $f_p^{-1} = 1$, ist die zweite Polynommultiplikation in der Entschlüsselungsphase unnötig und kann eingespart werden. Somit ist nur noch eine statt zwei Polynommultiplikationen durchzuführen.

Ein Sicherheitsproblem sollte durch diese spezielle Wahl von f nicht auftauchen. Zwar ist einem Angreifer dann bekannt, wie f_p^{-1} aussieht, dieses Polynom wird jedoch nur

während der Entschlüsselung bei einer zweiten Polynommultiplikation verwendet. Die Sicherheit des Verfahrens lässt sich jedoch nicht durch eine vergrößerte Anzahl von Multiplikationen erhöhen und wenn ein Angreifer es schafft, die erste Polynommultiplikation durchzuführen, so hat er bereits alle nötigen Informationen, um die restlichen Rechnungen auch noch durchzuführen, da ihm dann das Polynom f bekannt ist oder er es daraus berechnen kann.

Eine weitere Effizienzsteigerung des Verfahrens kann man erreichen, indem man das Polynom F in einer bestimmten Form wählt. Statt $f = 1 + p \circledast F$ setzt man

$$f = 1 + p \circledast ((f_1 \circledast f_2) + f_3),$$

wobei f_1, f_2 und f_3 zufällig gewählte kleine Polynome sind.

Da das Polynom f nun nicht mehr zufällig gewählt ist, sondern es direkt aus F berechnet wurde und nun auch F aus den Polynomen f_1, f_2 und f_3 berechnet wird, brauchen wir den Parameter d_f nicht mehr und anstelle dessen die Parameter d_{f_1}, d_{f_2} und d_{f_3}.

Sei nun t ein beliebiges Polynom und wir nehmen an, dass 72 Koeffizienten des Polynoms F ungleich 0 sein sollen. Dafür können wir $d_{f_1} = d_{f_2} = d_{f_3} = 8$ wählen. Eine Multiplikation von $t \circledast F$ in der ursprünglichen Form würde $(72 - 1) \cdot N = 71 \cdot N$ Additionen benötigen. Betrachtet man aber die gleiche Multiplikation mit $F = (f_1 \circledast f_2) + f_3$, so spaltet sich die Multiplikation in mehrere kleinere Multiplikationen auf:

$$\begin{aligned} t \circledast F &= t \circledast ((f_1 \circledast f_2) + f_3) \\ &= t \circledast (f_1 \circledast f_2) + t \circledast f_3 \\ &= (t \circledast f_1) \circledast f_2 + t \circledast f_3 \end{aligned}$$

Da $d_{f_1} = d_{f_2} = d_{f_3} = 8$, bestehen die Multiplikationen $t \circledast f_1$ und $t \circledast f_3$ aus $N \cdot (8 - 1) = 7 \cdot N$ Additionen. Ebenfalls werden $7 \cdot N$ Additionen bei der zweiten Multiplikation von $(t \circledast f_1) \circledast f_2$ durchgeführt. Insgesamt ergeben sich also durch die Multiplikationen $21 \cdot N$ Additionen, so dass man am Ende mit $22N$ Additionen auskommt und somit etwa zwei Drittel der Additionen einspart.

Genauso kann man auch das Zufallspolynom z betrachten. Statt z benutzen wir $z = (z_1 \circledast z_2) + z_3$ mit den Parametern d_{z_1}, d_{z_2} und d_{z_3}. Auch hier können wir eine deutliche Effizienzsteigerung beobachten.

6.5.4. Zwei zusammenfassende Beispiele

Im Folgenden gebe ich zwei Beispiele[34] an, die zeigen, wie ein kompletter Verschlüsselungsvorgang mithilfe von NTRU durchgeführt werden kann. Auf die in Kapitel 6.5.3 angesprochene Aufspaltung der Polynome, zum Beispiel $F = (f_1 \circledast f_2) + f_3$, wird in den Beispielen verzichtet, da der Grad der hier verwendeten Beispielpolynome zu klein ist, als dass eine solche Aufspaltung einen wirklichen Nutzen hätte, ohne dass die Übersicht darunter leiden würde. Die Änderung des Schlüsselpolynoms $f = 1 + p \circledast F$ und $p = 2 + X$ werden hier aber demonstriert.

Für die Beispiele sind die Parameter folgendermaßen gewählt:

[34] vgl. [NTR07]

- $N = 11$
- $p = 2 + X = [2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$
- $q = 32$
- $d_F = 4$
- $d_g = 5$
- $d_z = 4$

6.5.4.1. Beispiel einer problemlosen Entschlüsselung

Schlüsselgenerierung Für die Schlüssel müssen zwei Polynome F und g beliebig aus den jeweiligen Schlüsselräumen gewählt werden. Dann wird der Private Key $f = 1 + p \circledast F$ berechnet, sowie dessen Inverses $f_q^{-1} \circledast f \equiv 1 \pmod{q}$. Schließlich kann man aus diesen Angaben den Public Key $h = p \circledast f_q^{-1} \circledast g$ berechnen.

$$\begin{aligned} F &= [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0] \\ f &= [3, 1, 0, 0, 2, 1, 0, 2, 1, 2, 1] \\ f_q^{-1} &= [-7, -5, 12, -3, -2, 6, 13, -10, -8, -8, -15] \\ g &= [1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1] \\ h &= [15, 11, 9, -14, -12, 12, -7, -12, -13, -8, -2] \end{aligned}$$

Verschlüsselung Um die Klartextnachricht m zu verschlüsseln, braucht man ein zufälliges Polynom z und den Public Key des Empfängers. Damit berechnet man den Geheimtext $c = z \circledast h + m$

$$\begin{aligned} m &= [0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1] \\ z &= [1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0] \\ c &= [8, 11, 11, -7, 2, -12, 12, -8, 3, 9, -11] \end{aligned}$$

Entschlüsselung Um aus dem Geheimtext den ursprünglichen Klartext zu erhalten, muss man $a = f \circledast c$ berechnen.

$$a = [7, 14, 10, 15, 14, 13, 10, 11, 15, 14, 15]$$

An dieser Stelle müsste das Intervall, in dem die Koeffizienten sein dürfen, noch richtig zentriert werden. Nachdem aber alle Koeffizienten innerhalb des Intervalls $[7, 14]$ liegen, scheint dieser Schritt nicht nötig zu sein.

Jetzt folgt noch abschließend die Reduzierung von a modulo p. Dafür suchen wir ein Polynom d mit $d(-2) = a(-2) \pmod{2^N + 1}$. Dafür berechnen wir

$$a(-2) \pmod{2^N + 1} = 10971 \pmod{2049} \equiv 726$$

Durch Intervallschachtelung findet man ein Polynom d mit $d(-2) = 726$:

$$d = [0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1]$$

Dieses Polynom entspricht auch der Klartextnachricht m und somit ist das Verfahren korrekt abgeschlossen.

6.5.4.2. Beispiel einer Entschlüsselung, in der das Intervall richtig zentriert werden muss

Schlüsselgenerierung Wie im vorherigen Beispiel werden wieder alle Schlüsselpolynome generiert.

$$\begin{aligned} F &= [1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0] \\ f &= [3, 1, 0, 0, 2, 1, 2, 3, 1, 0, 0] \\ f_q^{-1} &= [14, 4, -1, -5, 10, 9, 6, 13, 4, 3, 12] \\ g &= [0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0] \\ h &= [16, 9, -3, 8, -2, -1, 16, 4, -4, 8, -8] \end{aligned}$$

Verschlüsselung Um die Klartextnachricht m zu verschlüsseln, braucht man ein zufälliges Polynom z und den Public Key des Empfängers. Damit berechnet man den Geheimtext $c = z \circledast h + m$

$$\begin{aligned} m &= [0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1] \\ z &= [0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0] \\ c &= [-12, 9, -2, 6, 13, -1, 4, -11, -5, -3, -12] \end{aligned}$$

Entschlüsselung Um aus dem Geheimtext den ursprünglichen Klartext zu erhalten, muss man $a = f \circledast c$ berechnen.

$$\begin{aligned} a &= [-23, 12, -19, -50, -23, -22, -47, -49, 17, 12, 10] \quad \text{vor der Reduzierung modulo } q \\ a &= [9, 12, 13, 14, 9, 10, -15, 15, -15, 12, 10] \pmod q \end{aligned}$$

Würde man an dieser Stelle $a \pmod p$ berechnen, so bekäme man statt der korrekten Klartextnachricht

$$m = [0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1]$$

eine falsche Nachricht

$$m' = [0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0].$$

Deshalb muss man das Intervall, in dem die Koeffizienten von a vor der Reduzierung modulo p sein müssen, korrekt zentrieren. Dafür berechnen wir wie oben angegeben:

$$\begin{aligned} I &= f_q^{-1}(1) \cdot (a(1) - p(1) \cdot z(1) \cdot g(1)) \\ &= 69 \cdot (74 - 3 \cdot 4 \cdot 5) \\ &\equiv 6 \pmod{32} \end{aligned}$$

$$\begin{aligned} Avg &= \frac{p(1) \cdot z(1) \cdot g(1) + I \cdot f(1)}{N} \\ &= \frac{3 \cdot 4 \cdot 5 + 6 \cdot 13}{11} \\ &= 12, \overline{54} \end{aligned}$$

Das korrekte Intervall ist nun $[Avg-16; Avg+16] = [-3; 28]$. Innerhalb dieser Grenzen wird a zu

$$a = [9, 12, 13, 14, 9, 10, 17, 15, 17, 12, 10] \pmod{q}.$$

Nach der Reduzierung modulo p erhalten wir

$$d = [0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1],$$

was auch der Klartextnachricht m entspricht.

6.6. NSS

Das auf NTRU basierende Signaturverfahren NSS wurde erstmals während der Rump Session von Crypto 2000 beschrieben. Es war die erste Anwendung von NTRU auf Signaturprobleme, jedoch stellte sich heraus, dass NSS in der ursprünglichen Version nicht besonders sicher war. Deshalb wurde es noch einmal überarbeitet, bis es schließlich während Eurocrypt 2001 vollständig vorgestellt wurde.[35]

6.6.1. Schlüsselerzeugung

Die öffentlichen Parameter werden wie in 6.2 festgelegt. Mithilfe von zwei zufällig gewählten Polynome $f_1 \in K(d_f, d_f)$ und $g_1 \in K(d_g, d_g)$ wird

$$\begin{aligned} f &= f_0 + pf_1 \\ g &= g_0 + pg_1 \end{aligned}$$

berechnet, wobei die Polynome f_0 und g_0 öffentlich bekannt sind und somit zu den öffentlichen Parametern gehören. Typischerweise gilt $f_0 = 1$ und $g_0 = 1 - 2X$.[36] Da f auch hier invertierbar modulo q sein muss, wird gegebenenfalls ein neues Polynom f_1 gewählt.

Der Public Key $h \in R_q$ wird nun aus $h = f_q^{-1} \circledast g$ berechnet.

[35] vgl. [GS02]
[36] vgl. [HPS01b]

6.6.2. Signierung

Wie weiter oben beschrieben, muss für den Signiervorgang die Nachricht m mit Hilfe einer Hashfunktion modifiziert werden. Da für NSS keine spezielle Hashfunktion vorgesehen ist und dafür auf die marktüblichen Hashalgorithmen wie SHA[37] zurückgegriffen wird, werde ich in der Arbeit nicht weiter auf die Sicherheit der Hashfunktion eingehen und davon ausgehen, dass alle Nachrichten sicher in einen Hashwert überführt wurden.

Zum Signieren berechnet man zuerst

$$w = m + w_1 + pw_2,$$

mit zwei Polynomen w_1 und w_2 und anschließend

$$s \equiv f \circledast w \pmod{q}$$

wobei das Paar (m, s) die Signatur der Nachricht m ist.

Die Polynome w_1 und w_2 werden so gewählt, dass es für einen Angreifer möglichst schwer sein soll, den Private Key aus der Signierung zu extrahieren oder eine Signatur zu fälschen. Weiterhin enthält das Polynom w die Nachricht m, so dass diese quasi an die Signatur angehängt ist. Um die Polynome w_1 und w_2 berechnen zu können, wählt man zuerst das Polynom $w_2 \in K(d_{w_2}, d_{w_2})$ zufällig. Mit diesem berechnet man zwei vorläufige Signaturpolynome s' und t':

$$s' \equiv f \circledast (m + pw_2)$$
$$t' \equiv g \circledast (m + pw_2)$$

Dann startet man mit $w_1 = 0$ und betrachtet für $i = 0, 1, \ldots, N-1$ nacheinander alle Koeffizienten s'_i von s' und t'_i von t und führt folgende Schritte aus:

- Wenn $s'_i \not\equiv m_i \pmod{p}$ und $t'_i \not\equiv m_i \pmod{p}$ und $s'_i \equiv t'_i \pmod{p}$, dann setze $w_{1,i} \equiv m_i - s'_i \pmod{p}$.
- Wenn $s'_i \not\equiv m_i \pmod{p}$ und $t'_i \not\equiv m_i \pmod{p}$ und $s'_i \not\equiv t'_i \pmod{p}$, dann setze zufällig $w_{1,i} = 1$ oder $w_{1,i} = -1$.[38]
- Wenn $s'_i \not\equiv m_i \pmod{p}$ und $t'_i \equiv m_i \pmod{p}$, dann setze, mit einer Wahrscheinlichkeit von 25%, $w_{1,i} \equiv m_i - s'_i \pmod{p}$, ansonsten bleibt $w_{1,i} = 0$.
- Wenn $s'_i \equiv m_i \pmod{p}$ und $t'_i \not\equiv m_i \pmod{p}$, dann setze, mit einer Wahrscheinlichkeit von 25%, $w_{1,i} \equiv m_i - t'_i \pmod{p}$, ansonsten bleibt $w_{1,i} = 0$.
- Wenn $s'_i \equiv t'_i \equiv m_i \pmod{p}$, dann belass $w_{1,i} = 0$.

[37] Secure Hash Algorithm

[38] Die Wahl von $w_{1,i}$ gilt in dieser Form nur für $p = 3$. Die Idee, dass man, wie in 6.5.2 beschrieben, $p = 2 + X$ wählen kann, kam erst, nachdem NSS schon entwickelt war.

- Wenn $i = N - 1$, so ist die Konstruktion von w_1 abgeschlossen. Weiterhin gibt es einen öffentlichen Parameter, der besagt, wie viele Koeffizienten von w_1 maximal ungleich Null sein sollen. Ist diese Anzahl unabhängig von i erreicht, ist die Konstruktion von w_1 ebenfalls abgeschlossen.

Schlussendlich wird w_2 noch verwürfelt, um eine statistische Analyse bei Transcript-Angriffen[39] zu erschweren. Dafür verändert man für alle $0 \leq i < N$ mit einer Wahrscheinlichkeit von 33% den i-ten Koeffizienten von w_2 so, dass $w_{2,i} \equiv w_{2,i} - m_i - w_{1,i} \pmod{p}$. Mit den so gewählten Polynomen kann man die oben angegebene Signatur berechnen.

6.6.3. Verifikation

Um zu überprüfen, ob die Signatur s ungefälscht ist und zur Nachricht m gehört, wird zusätzlich zum Polynom s ein weiteres Polynom $t \equiv h \circledast s \pmod{q}$ aus dem Public Key berechnet. Wichtig ist, dass am Anfang geprüft wird, ob $s \neq 0$, da mithilfe des trivialen Polynoms möglicherweise eine Fälschung möglich ist.

Um eine Signatur als gültig zu akzeptieren, müssen zwei Bedingungen erfüllt sein:

$$\begin{aligned} D_{\min} \leq \mathrm{Dev}(s, f_0 \circledast m) \leq D_{\max} \\ D_{\min} \leq \mathrm{Dev}(t, g_0 \circledast m) \leq D_{\max}, \end{aligned}$$

wobei $D_{\min}$ und $D_{\max}$ öffentliche Parameter sind, die vorher festgelegt werden.

Zu $D_{\min} \leq \mathrm{Dev}(s, f_0 \circledast m) \leq D_{\max}$**:** Für das Polynom s gilt

$$\begin{aligned} s &\equiv f \circledast w \pmod{q} \\ &\equiv (f_0 + pf_1) \circledast (m + w_1 + pw_2) \pmod{q} \\ &\equiv f_0 \circledast m + f_0 \circledast w_1 + p \cdot f_0 \circledast w_2 + p \cdot f_1 \circledast w \pmod{q} \end{aligned}$$

Somit kann man erkennen, dass der i-te Koeffizient von s und $f_0 \circledast m$ modulo p übereinstimmt, außer in den Fällen

- der i-te Koeffizient von $f_0 \circledast w_1$ ist ungleich Null.
- der i-te Koeffizient von $f \circledast w$ ist außerhalb des Intervalls $(-\frac{q}{2}, \frac{q}{2}]$ und wird daher bei der Betrachtung modulo q verfälscht.

w_1 ist aber so konstruiert, dass die Anzahl der Koeffizienten ungleich Null beschränkt ist. Weiterhin ist sowohl das Polynom f als auch das Polynom w klein und somit ist im Allgemeinen die Anzahl der Koeffizienten von $f \circledast w$, die außerhalb des Intervalls $(-\frac{q}{2}, \frac{q}{2}]$ liegen, nicht sehr hoch.

Zu $D_{\min} \leq \mathrm{Dev}(t, g_0 \circledast m) \leq D_{\max}$**:** Für das Polynom t gilt

$$\begin{aligned} t &\equiv h \circledast s \pmod{q} \\ &\equiv (f_q^{-1} \circledast g) \circledast (f \circledast w) \pmod{q} \\ &\equiv g \circledast w \pmod{q} \end{aligned}$$

[39] vgl. Kapitel 7.2

Da f und g prinzipiell gleich aufgebaut sind, gilt hier die gleiche Argumentation wie im oben diskutierten Fall.

Durch eine geeignete Wahl der Parameter ist die Wahrscheinlichkeit relativ hoch, dass eine valide Signatur als solche angesehen wird. So liegt die Wahrscheinlichkeit, eine valide Signatur zu bekommen, bei den Parametern $(N, p, q, D_{\min}, D_{\max}) = (251, 3, 128, 55, 87)$ bei $79,40\%$.[40] Wird eine Signatur abgelehnt, so muss eine neue Signatur mit neuem Zufallspolynom w_2 erstellt werden.

Wie schon in Bemerkung 6.14 angedeutet, ist die erwartete Abweichung zweier zufälliger Polynome u und v gleich $\frac{2}{3}N$. Daher ist es wichtig, dass die Grenzen $D_{\min}$ und $D_{\max}$ so gewählt werden, dass es sehr unwahrscheinlich ist, eine gefälschte Signatur zu erstellen, die trotzdem valide ist.

6.7. R-NSS

Nachdem verschiedene Kryptoanalytiker bei NSS mehrere Schwachstellen aufzeigen konnte, wurde eine verbesserte Version von NSS entwickelt, die gegen die bisherigen Angriffe standhalten kann.

6.7.1. Schlüsselerzeugung

Die öffentlichen Parameter werden wie in 6.2 festgelegt. Dann werden die drei Polynome $f_1 \in K(d_f, d_f)$, $g_1 \in K(d_g, d_g)$ und $u \in K(d_u, d_u + 1)$ zufällig gewählt, und damit

$$f = u + pf_1$$
$$g = u + pg_1$$

berechnet. Wie auch oben schon geschrieben, muss f in R_q invertierbar sein. Ist dies nicht der Fall, so müssen andere Polynome f_1 und u gewählt werden. Weiterhin ist erforderlich, dass u in R_p invertierbar ist. Der Private Key des Signierers sind die Polynome f, g und u.

Den Public Key $h \in R_q$ berechnet man aus $h = f_q^{-1} \circledast g$.

6.7.2. Signierung

Sei $m \in K(d_m, d_m)$ die Klartextnachricht, die schon eine Hashfunktion durchlaufen hat. Dann werden zwei zufällige Polynome $w_1 \in K(d_1, d_2)$ und $w_2 \in K(d_{w_2}, d_{w_2})$ gewählt, wobei die Größen von d_1 und d_2 nicht festgelegt sind und sich abhängig von N, p und q in bestimmten Intervallen bewegen.[41] Damit werden dann folgende Polynome berechnet:

$$y = ((u_p^{-1} \circledast m) \pmod p) + ((u_p^{-1} \circledast w_1) \pmod p)$$
$$w = y + pw_2$$

[40] vgl. [HPS01b]
[41] vgl. [HPS01a]

Dabei gilt in R, dass $f \circledast w \equiv m \pmod{p}$. Im Ring R_q ist jedoch im Allgemeinen diese Kongruenz in einigen Koeffizienten f_i gestört, da durch die Reduzierung modulo q Abweichungen in f_i auftreten können, wie auch am Anfang von Kapitel 6.4.4 angedeutet ist. Das Ziel während der restlichen Signierung besteht nun darin, die Anzahl dieser Abweichungen so gering wie möglich zu halten. Dafür werden weitere Berechnungen durchgeführt:

$$\begin{aligned} s &= f \circledast w \pmod{q} \\ t &= g \circledast w \pmod{q} \\ \mathrm{Dev}_s &= (s - m) \pmod{p} \\ \mathrm{Dev}_t &= (t - m) \pmod{p} \end{aligned}$$

Die beiden Polynome Dev_s und Dev_t repräsentieren die Abweichungen der Polynome s und t von m. Könnte man beide Polynome auf das 0-Polynom setzen, wäre die ideale Signatur gefunden. Da jedoch sowohl s als auch t von w abhängen, bewirkt eine Korrektur der Abweichungen in s möglicherweise neue Abweichungen in t. Daher beschränkt man sich bei der Korrektur nur auf die Koeffizienten an der Stelle j, bei denen $(\mathrm{Dev}_s)_j = (\mathrm{Dev}_t)_j$ und führt dann folgende Schritte aus.

1. Setze $e := 0$ und $e_j := -(\mathrm{Dev}_s)_j$, für alle j mit $(\mathrm{Dev}_s)_j = (\mathrm{Dev}_t)_j$.
2. Berechne $e' = u_p^{-1} \circledast e \pmod{p}$.
3. Addiere e' an w und berechne $s = f \circledast w \pmod{q}$ erneut.

Durch diese Modifikation des Polynoms w ist die Anzahl der Abweichungen sowohl im Polynom s als auch im Polynom t insgesamt um die Anzahl der Übereinstimmungen kleiner.

Das Paar (m, s) ist nun die Signatur der Nachricht m.

6.7.3. Verifikation

Bei der Verifikation wird mit Hilfe des Public Keys h ein Verifikationspolynom t berechnet

$$\begin{aligned} t &\equiv h \circledast s \pmod{q} \\ &\equiv g \circledast w \pmod{q} \end{aligned}$$

Mit den beiden Polynomen s und t kann nun getestet werden, ob die Signatur valide ist. Dafür schlagen die Entwickler rund 20 Tests vor, wobei ich mich bei der Beschreibung auf die folgenden drei Tests beschränke. Dabei wird getestet, ob

1. für die Polynome $s' := p^{-1}(s - m) \pmod{q}$ und $t' := p^{-1}(t - m) \pmod{q}$ die Normen $||s'||_z$, $||t'||_z$ und $||(s', t')||_z$ unterhalb bestimmter Grenzen liegen.
2. für beide Polynome s und t gilt, dass $\mathrm{Dev}(s, m)$ und $\mathrm{Dev}(t, m)$ klein sind und insbesondere die abweichenden Koeffizienten eine bestimmte Verteilung haben.
3. die Koeffizienten von s und t annähernd normale Verteilung haben.

In den folgenden Fällen betrachte ich oBdA das Polynom s beziehungsweise s', da die Argumentation für t beziehungsweise t' durch den gleichen Aufbau der Polynome sehr ähnlich ist.

Zu 1. Betrachten wir zuerst, wie das Polynom s aufgebaut ist.

$$\begin{aligned} s &\equiv f \circledast w \pmod q \\ &\equiv (u + pf_1) \circledast (y + pw_2) \pmod q \\ &\equiv u \circledast y + p \cdot (\text{ein mittelgroßes Polynom}) \pmod q \\ &\equiv m + w_1 + p \cdot (\text{ein mittelgroßes Polynom}) \pmod q \end{aligned}$$

Dabei wurde benutzt, dass

$$u \circledast y \equiv u \circledast (u_p^{-1} \circledast m + u_p^{-1} \circledast w_1) \equiv m + w_1 \pmod p$$

wobei mögliche Abweichungen zwischen der Betrachtung modulo p und modulo q durch das mittelgroße Polynom ausgeglichen werden können. Somit gilt

$$\begin{aligned} s' &\equiv p^{-1} \cdot (s - m) \pmod q \\ &\equiv p^{-1} \cdot w_1 + (\text{ein mittelgroßes Polynom}) \pmod q \end{aligned}$$

Das Polynom w_1 hat nach Voraussetzung nur wenige Koeffizienten ungleich Null und somit auch $p^{-1} \cdot w_1$. Das mittelgroße Polynom besteht aus den Polynomen f_1, u und w_2, dem Modul p, sowie einem unbekannten Polynom, das für den vorherigen Ausgleich im Übergang der Betrachtung von modulo p zu modulo q steht. Aus heuristischen Überlegungen kann man bei geschickter Wahl der Parameter mit hoher Wahrscheinlichkeit davon ausgehen, dass s' eine Norm hat, die unterhalb einer von den Parametern abhängigen Grenze liegt.

Der Test, ob zusätzlich $||(s', t')||_z$ kleiner als eine bestimmte Schranke ist, folgt direkt aus dem Polynomfaktorisierungs-Problem, da dort die Voraussetzung war, dass für eine mögliche Lösung f' und g' die Norm des $2N$-dimensionalen Vektors (f', g') beschränkt sein muss.

Zu 2. Wie in der Erläuterung zu 1. schon gezeigt, gilt

$$s \equiv f \circledast w \equiv m + w_1 + p \cdot (\text{ein mittelgroßes Polynom}) \pmod q.$$

Betrachtet man nun $s - m \pmod p$, so gibt es zwei Gründe für von Null verschiedene Koeffizienten:

(a) Die betroffenen Koeffizienten von w_1 sind ungleich Null.

(b) Im Produkt $f \circledast w$ sind Koeffizienten außerhalb des Intervalls $(-\frac{q}{2}, \frac{q}{2}]$, so dass die Reduktion modulo q nichttrivial ist.

Die Abweichungen, die in (a) beschrieben werden, sind sehr selten. Während der Signierphase hat man bei der Wahl von w_1 viele Freiheiten, so dass man in diesem

Fall am geschicktesten ein Polynom wählt, dass einige Abweichungen verschwinden lässt.[42]

Die Abweichungen vom Typ (b) tendieren dazu, sich in einer bestimmten Konvergenzklasse modulo p zu häufen, je nachdem, ob der korrespondierende Koeffizient von s näher an der oberen oder unteren Intervallgrenze von q liegt. Speziell wird die Häufung bei $q \pmod p$ beziehungsweise bei $-q \pmod p$ liegen, wenn der Koeffizient von s nahe der oberen beziehungsweise unteren Grenze des Intervalls ist.

Um nun den Test durchzuführen, teilt man das Intervall in vier Quantile ein, so dass

$$I_1 = \left(-\frac{q}{2}, \frac{q}{4}\right], \qquad I_2 = \left(-\frac{q}{4}, 0\right], \qquad I_3 = \left(0, \frac{q}{4}\right], \qquad I_4 = \left(\frac{q}{4}, \frac{q}{2}\right]$$

Weiterhin berechnen wir zwei Maße, mit denen man die Abweichungen, die nicht im erwarteten Bereich liegen, erfassen kann.

$$\begin{aligned}\mathrm{Dev}_1 = &\#\{j \mid s_j \in I_4 \text{ und } s_j - m_j \not\equiv 0, q \pmod p\}\\ + &\#\{j \mid s_j \in I_1 \text{ und } s_j - m_j \not\equiv 0, -q \pmod p\}\\ + &\#\{j \mid t_j \in I_4 \text{ und } t_j - m_j \not\equiv 0, q \pmod p\}\\ + &\#\{j \mid t_j \in I_1 \text{ und } t_j - m_j \not\equiv 0, -q \pmod p\}\end{aligned}$$

$$\begin{aligned}\mathrm{Dev}_2 = &\#\{j \mid s_j \in I_3 \text{ und } s_j - m_j \not\equiv 0, q \pmod p\}\\ + &\#\{j \mid s_j \in I_2 \text{ und } s_j - m_j \not\equiv 0, -q \pmod p\}\\ + &\#\{j \mid t_j \in I_3 \text{ und } t_j - m_j \not\equiv 0, q \pmod p\}\\ + &\#\{j \mid t_j \in I_2 \text{ und } t_j - m_j \not\equiv 0, -q \pmod p\}\end{aligned}$$

Wenn man nun verifiziern kann, dass sowohl Dev_1 als auch Dev_2 kleiner als vorher festgelegte Schranken sind, gilt dieser Test als bestanden.

Zu 3. In 1. haben wir nachgewiesen, dass $||s'||_z$ durch eine vorher festgelegte obere Grenze beschränkt ist. Wenn man nun die einzelnen Koeffizienten von s' betrachtet, so werden diese eher zu ziemlich kleinen Werten tendieren, da alle Polynome, mit denen s' erzeugt wird, klein sind. Auf der anderen Seite wird es bei einer gültigen Signatur eher unwahrscheinlich sein, dass eine große Anzahl von Koeffizienten klein und einige wenige sehr groß sind, während dazwischen keine Werte angenommen werden. Daher basiert dieser Test darauf, zu prüfen, ob sich die Anzahl der Koeffizienten, deren Wert in jeweils einem der vier Quantilen J_1, J_2, J_3 oder J_4 liegt, in vorher festgelegten Intervallen befindet. Sei also

$$J_1 = \left(0, \frac{q}{8}\right], \qquad J_2 = \left(\frac{q}{8}, \frac{q}{4}\right], \qquad J_3 = \left(\frac{q}{4}, \frac{3q}{8}\right], \qquad J_4 = \left(\frac{3q}{8}, \frac{q}{2}\right].$$

[42] Vergleiche dazu das Polynom e' während der Signierphase.

Weiterhin ist

$$\text{Coef}_{s,i} = \#\{j \mid |s'_j| \in J_i\}$$

die Anzahl der Koeffizienten von s, deren Werte innerhalb des Quantils J_i liegen und

$$\text{Coef}_{t,i} = \#\{j \mid |t'_j| \in J_i\}$$

die Anzahl der Koeffizienten von t mit dieser Eigenschaft.

Wenn all diese Tests erfolgreich durchgeführt wurden, wird die Signatur als valide akzeptiert. Durchschnittlich muss der Signiervorgang zwei bis drei Mal wiederholt werden, bevor eine gültige Signatur erzeugt wird.

Beispiel 6.33 (Parameterwahl). [43] Sei $(N, p, q) = (251, 3, 128)$. Dann nehmen wir Polynome $f_1 \in K(52, 52)$, $g_1 \in K(36, 36)$, $u \in K(88, 87)$, $m \in K(80, 80)$ und $w_2 \in K(58, 58)$. Für die Parameter von $w_1 \in K(d_1, d_2)$ gilt: $12 \leq d_1 + d_2 \leq 20$.

Eine Signatur s und das Hilfspolynom $t = h \circledast s$ werden dann als valide akzeptiert, wenn folgende Bedingungen erfüllt sind:

$$\begin{aligned}
&(1) \quad \text{Dev}_1 \leq 10 \text{ und } \text{Dev}_2 \leq 18 \\
&(2) \quad ||(s', t')||_z \leq 485 \\
&(3) \quad ||s'||_z \leq 360 \text{ und } ||t'||_z \leq 360 \\
&(4) \quad 95 \leq \text{Coef}_{s,1}, \text{Coef}_{t,1} \leq 153 \\
&\qquad 50 \leq \text{Coef}_{s,2}, \text{Coef}_{t,2} \leq 100 \\
&\qquad\ 7 \leq \text{Coef}_{s,3}, \text{Coef}_{t,3} \leq 42 \\
&\qquad\qquad \text{Coef}_{s,4}, \text{Coef}_{t,4} \leq 14
\end{aligned}$$

6.7.4. Hauptunterschiede zu NSS

- Im Gegensatz zu NSS kommt bei R-NSS eine weitere geheime Schlüsselkomponente u hinzu, mit der während der Schlüsselgenerierung die öffentlichen Polynome f_0 und g_0 ersetzt werden.
- Während der Signierphase ist die Berechnung des Polynoms w abgeändert worden. Insbesondere wird nun zusätzlich das Inverse des neuen Schlüsselpolynoms u verwendet.
- Die verstärkten Verifikationsbedingungen von R-NSS, zum Beispiel, dass $||s'||_z = ||p^{-1}(s - m)||_z$ kleiner sein soll als eine vorher festgelegte Schranke, enthalten und erweitern die einfachen Abweichungsgrenzen von NSS.
- Um zu verhindern, dass eine gefälschte Signatur konstruiert werden kann, wird nun zusätzlich geprüft, ob s und t weitestgehend normal verteilt sind.

[43] vgl. [HPS01a]

6.8. NTRUSign

Auch bei R-NSS wurden Sicherheitslücken aufgedeckt. Insbesondere erkannten die Entwickler, dass die schwache Verflechtung zwischen dem Signierverfahren und dem darunterliegenden Gitterproblem eine Schwachstelle von NSS und auch von R-NSS darstellt. Daher wurde das Verfahren umgeändert, so dass es jetzt noch stärker auf einem Gitterproblem, in diesem Fall ist es CVP[44], auf dem NTRU-Gitter basiert.[45]

6.8.1. Öffentliche Parameter

Durch die grundsätzliche Modifikation des Verfahrens haben sich die öffentlichen Parameter geändert. Hauptsächliche Änderung ist der Wegfall des Moduls p. Damit entfällt das häufige Wechseln der Betrachtung modulo p und q. Die Definitionen von R beziehungsweise R_q bleiben aber wie in Kapitel 6.1 eingeführt. Hinzugefügt wurde ein weiterer Parameter ν, der eine obere Schranke bei der Verifikation angibt.

6.8.2. Schlüsselerzeugung

Zuerst werden für den Private Key zwei kleine Polynome f und g zufällig gewählt. Dann wird daraus der Public Key $h \equiv f^{-1} \circledast g \pmod{q}$ berechnet. Weiterhin werden als Teil des Private Keys zwei weitere kleine Polynome F und G berechnet, die

$$f \circledast G - g \circledast F = q$$

erfüllen. Die Größe der Polynome F und G kann man folgendermaßen abschätzen: Sei $||f||_z \approx c\sqrt{N}$ und $||g||_z \approx c\sqrt{N}$ für ein gegebenes festes c, dann ist es möglich, ein zugehöriges Paar F und G zu finden, mit

$$||F||_z \approx ||G||_z \approx \frac{cN}{\sqrt{12}}$$

Seien nun M_f, M_g, M_F, M_G und M_h die Konvolutionsmatrizen von f, g, F, G und h und $I_{(N)}$ die N-dimensionale Einheitsmatrix. Dann wird das NTRU-Gitter $\mathcal{L}^{\mathrm{NTRU}}$ von den Basen

$$B_{\mathrm{private}} = \begin{bmatrix} M_f & M_g \\ M_F & M_G \end{bmatrix}$$

und

$$B_{\mathrm{public}} = \begin{bmatrix} I_{(N)} & M_h \\ 0 & qI_{(N)} \end{bmatrix}$$

erzeugt.

[44] Closest Vector Problem
[45] vgl. [HHGP+03]

6.8.3. Signierung

Zum Signieren muss die Nachricht in einen Hashwert $m = (m_1, m_2)$ überführt werden. Die Signatur $(s,t) \in \mathcal{L}^{\mathrm{NTRU}}$ ist dann ein nächster Gitterpunkt nahe m. Dazu drückt man (m_1, m_2) als eine $\mathbb{Q}$-lineare Kombination der kurzen Basisvektoren aus und rundet die Koeffizienten auf die nächsten ganzen Zahlen. Konkret wird dabei folgendes berechnet.

- Berechne Polynome $a, A \in R_q$ und $b, B \in R$ durch

$$\begin{aligned} G \circledast m_1 - F \circledast m_2 &= A + q \circledast B \\ -g \circledast m_1 + f \circledast m_2 &= a + q \circledast b \end{aligned}$$

- Berechne Polynome s und t, so dass

$$\begin{aligned} s &\equiv f \circledast B + F \circledast b \\ t &\equiv g \circledast B + G \circledast b \end{aligned}$$

Bemerkung 6.34. Man kann beobachten, dass (s,t) auch im darunterliegenden Gitter liegt, da

$$(s,t) = B \circledast (f,g) + b \circledast (F,G) \pmod{q}$$

6.8.4. Verifikation

Zur Verifikation muss man folgende zwei Schritte durchführen.

1. Berechne $t \equiv h \circledast s \pmod{q}$.
2. Prüfe, ob $||(s - m_1, t - m_2)||_z \leq \nu$.

Die Verifikation funktioniert, da dabei überprüft wird, ob der Signierer das CVP effizient lösen kann, das heißt, ob er einen Vektor angeben kann, der keine große Abweichung von unserem festgelegten Nachrichtenpunkt $m = (m_1, m_2)$ besitzt. Ohne Kenntnis der Basisvektoren (f,g) und (F,G) ist es sehr schwer, einen solchen nahen Punkt zu erraten.

Für die Grenze ν gilt, dass es umso schwerer für einen Fälscher ist, eine falsche Signatur auszuliefern, je kleiner ν gewählt ist. Auf der anderen Seite macht ein zu kleines ν Probleme während der Signierung, da es dann schwieriger ist, eine gültige Signatur zu erstellen, wenn nämlich kein Gitterpunkt in diesem Abstand vorhanden ist.

7. Kryptoanalyse auf den NTRU-Signaturverfahren

Die Kryptoanalyse von auf NTRU basierenden Kryptosystemen begann kurz nach der ersten Veröffentlichung des Verfahrens. Nachdem mit Don Coppersmith und Adi Shamir[1] zwei sehr renommierte Kryptologen das Verfahren untersucht haben und, obwohl sie Schwachstellen im Verfahren aufgedeckt haben, anmerkten, dass noch Potential in der veröffentlichten Idee steckt, begannen mehrere Kryptologen, sich mit dem Verfahren auseinander zu setzen.

Im Folgenden werde ich mich hauptsächlich auf die in Kapitel 7.1 und in Kapitel 7.2 vorgestellten Angriffe von Gentry, Jonsson, Stern und Szydlo[2], sowie bei den restlichen Kapiteln auf die Arbeit von Gentry und Szydlo[3] beziehen. Dabei werde ich zeigen, wie diese Angriffe die Verfahren NSS[4] und R-NSS[5] brechen. Das momentan aktuelle Verfahren NTRUSign[6] scheint weniger gravierende Sicherheitslücken zu haben, so dass bisher noch kein erfolgreicher Angriff durchgeführt worden ist.[7]

Nach der Beschreibung der einzelnen Angriffe werde ich eine Übersicht geben, warum die vorgestellten Angriffe bei den Weiterentwicklungen der auf NTRU basierenden Signaturverfahren ins Leere laufen.

Die Entwickler haben bei der Präsentation ihrer Verfahren verschiedene Angriffsszenarien simuliert und erläutert, warum die Verfahren sicher sind gegen diese Angriffe. Jedoch offenbart sich hier das Problem von Sicherheitsbeweisen bei Kryptosystemen, da man nur sehr schwer alle möglichen Angriffsmöglichkeiten abdecken kann. So kann man zum Beispiel Seitenkanalangriffe nie komplett ausschließen. Ich werde daher in meiner Arbeit auf diese Angriffe nicht weiter eingehen, da diese Angriffe keine Probleme der Verfahren aufdecken.

[1] vgl. [CS97]
[2] vgl. [GJSS01]
[3] vgl. Gentry:2002
[4] vgl. [HPS96a]
[5] vgl. [HPS01a]
[6] vgl. [HHGP+03]
[7] vgl. [Ngu06]: Es wurde zwar ein Angriff konstruiert, bei dem man nur 400 gültige Signaturen braucht, um den Private Key zu berechnen, dieser Angriff funktioniert aber nur, so lange spezielle Parameter, in diesem Fall die von den Entwicklern vorgeschlagenen Parameter, verwendet werden. Bei einer anderen Wahl der Parameter funktioniert dieser Angriff nicht.

7.1. Erster Angriff auf NSS: Ein einfacher Fälschungsangriff

In Kapitel 6.6 habe ich das Signaturverfahren NSS vorgestellt. Ich werde im Folgenden die Parameter, wie auch in Kapitel 6.6.3 angedeutet, auf $(N, p, q, D_{\min}, D_{\max}) = (251, 3, 128, 55, 87)$ setzen. Wurde eine Signatur (s, m) erstellt, so kann man diese testen, indem man

$$D_{\min} \leq \mathrm{Dev}(s, f_0 \circledast m) \leq D_{\max}$$
$$D_{\min} \leq \mathrm{Dev}(t, g_0 \circledast m) \leq D_{\max}$$

überprüft, wobei f_0 und g_0 öffentlich bekannte Teile des Private Keys sind und $t \equiv s \circledast h \pmod{q}$.

Der im Folgenden vorgestellte Angriff zeigt, dass NSS nicht nur strukturelle Schwächen hat, sondern komplett unsicher ist. Wie im Folgenden gezeigt, gelingt es einem Angreifer auch ohne Kenntnis oder Berechnung des Private Keys zumeist, eine Signatur zu fälschen und das mit einem Aufwand, der sich kaum vom Aufwand unterscheidet, der beim normalen Signieren nötig ist.

7.1.1. Die Grundzüge

Will nun ein Fälscher eine gefälschte Signatur erstellen, so muss er ein Polynompaar (s, t) finden, dass $t \equiv s \circledast h \pmod{q}$ erfüllt und

$$55 \leq \mathrm{Dev}(s, f_0 \circledast m) \leq 87$$
$$55 \leq \mathrm{Dev}(t, g_0 \circledast m) \leq 87$$

Die Polynome s und t haben zusammen $2N$ Koeffizienten und die Gleichung $t \equiv s \circledast h \pmod{q}$ stellt N lineare Bedingungen, so dass der Fälscher noch N Koeffizienten in s und t frei wählen kann, um die Anzahl der Abweichungen zu reduzieren. Mit diesen N Freiheitsgraden kann man $\lfloor \frac{N}{2} \rfloor$ Koeffizienten von s und $\lceil \frac{N}{2} \rceil$ Koeffizienten von t so wählen, dass diese Koeffizienten keine Abweichungen aufweisen. Dies wird durch die Wahl

$$s_i \equiv (f_0 \circledast m)_i \pmod{p}$$
$$t_j \equiv (g_0 \circledast m)_j \pmod{p}$$

erreicht.

Somit hat die eine Hälfte der Koeffizienten von s und auch von t keine Abweichungen und wie in Bemerkung 6.14 eingeführt, werden bei bei der anderen Hälfte etwa $\frac{2}{3}$ der Koeffizienten eine Abweichung haben. Somit wird insgesamt die Abweichung der Koeffizienten von s und t bei etwa $\frac{1}{3}$ liegen. Nachdem $\frac{1}{3}N \approx 84 \leq 87 = D_{\max}$ für $(N, D_{\max}) = (251, 87)$, wird dieses Verfahren nach ein paar Iterationen eine gültige, aber gefälschte Signatur erzeugen.

Allgemein kann man feststellen, dass, wenn $p = 3$ und $D_{\max} \geq \frac{1}{3}N$, dann wird dieser Angriff unabhängig von der Größe von N Fälschungen generieren können.

7.1.2. Details

In der Praxis ist ein solcher Angriff etwas schwerer als eben eingeführt, da es möglich ist, dass die Bedingungen an s und t inkompatibel sind. In diesem Fall bezeichnet man den Angreifer als *glücklos*. Die Aufgabe des Angreifers besteht nun darin, zu verhindern, dass dieser Fall eintritt. Für das Polynom t muss der Angreifer die Bedingung $t \equiv s \circledast h \pmod{q}$ erfüllen und somit sind, wie eingangs eingeführt, $\frac{N}{2}$ Koeffizienten von s und $\frac{N}{2}$ Koeffizienten t an diese Bedingung gebunden und nur die restlichen Koeffizienten können frei gewählt werden, um die Abweichungen zu minimieren. Um nun zu verhindern, dass der Angreifer glücklos ist, wendet er die Bedingung nur auf $k < \frac{N}{2}$ Koeffizienten von s und t an.

Die Bedingung $t \equiv s \circledast h \pmod{q}$ kann als lineares Gleichungssystem gelöst werden. Somit erhält man, wie eben beschrieben, k lineare Gleichungen modulo q über $(N-k)$ Unbekannten. Die Koeffizienten dieser Unbekannten bilden eine $k \times (N-k)$-Untermatrix M von M_h, wobei h der Public Key desjenigen ist, dem der Fälscher eine Signatur unterschieben will und die Koeffizienten dieser Matrix ganze Zahlen modulo q sind. Aus heuristischen Vermutungen kann man davon ausgehen, dass diese Koeffizienten modulo 2 stochastisch unabhängige Binärzahlen sind.

Lemma 7.1. *Die Wahrscheinlichkeit, dass ein Angreifer glücklos ist, entspricht basierend auf heuristischen Vermutungen* $\varepsilon = \frac{1}{2^{N-2k}}$.

Beweis. Ich werde zeigen, dass, mit einer Wahrscheinlichkeit von mindestens $1-\varepsilon$, die Spalten von M modulo 2 den gesamten k-dimensionalen Raum über dem zweielementigen Körper erzeugen. Stimmt dies, so hat das Gleichungssystem Rang k und hat somit eine Lösung modulo 2 und damit auch modulo q, da q als eine Zweierpotenz gewählt ist.

Sei nun x ein binärer k-Vektor. Mit einer Wahrscheinlichkeit von $\frac{1}{2}$ ist das Skalarprodukt von x mit einem Spaltenvektor v gleich Null. Da M aus $(N-k)$ unabhängigen Spalten besteht, ist die Wahrscheinlichkeit, dass x orthogonal zu allen Spaltenvektoren ist, gleich $\frac{1}{2^{N-k}}$. Die Anzahl der Möglichkeiten, die Koeffizienten von x zu wählen, entspricht 2^k. Somit ist die Wahrscheinlichkeit, dass es keinen Vektor gibt, der orthogonal zu allen Spaltenvektoren ist, gleich $1 - \frac{1}{2^{N-2k}}$. Das heißt, diese Spaltenvektoren spannen den gesamten Raum auf. □

Setzt man nun $k = 121$, wird ein Angreifer mit einer Wahrscheinlichkeit von $1 - 2^{-9}$ nicht glücklos sein. Im Folgenden nehme ich an, dass der Angreifer nicht glücklos ist. Jetzt entspricht der Angriff dem Lösen eines Gleichungssystems mit 121 Gleichungen mit 130 Unbekannten. Dazu nehme ich an, dass nach geschickter Umnummerierung der Koeffizienten die von s gebundenen Koeffizienten am Anfang stehen und die von t gebundenen am Ende. Nach dieser Umnummerierung ist die Matrix M des Systems, das der Angreifer lösen muss, die untere rechte Ecke der Matrix M_h, also die letzten k Zeilen und die letzten $N-k$ Spalten von M_h.

Die Entwickler behaupten nun an dieser Stelle, dass man durch weiteres geschicktes Umnummerieren der Koeffizienten in der unteren rechten Ecke von M eine invertierbare Matrix U erhalten kann. Berechnet der Angreifer nun diese Inverse U^{-1}, kann er damit

Lösungen generieren, wobei er die $N - 2k = 9$ mittleren Koordinaten von S willkürlich wählt und die letzten k Koeffizienten durch eine Multiplikation mit U^{-1}. Diese Lösungen erfüllen die geforderte Abweichungsbedingung mit einer Wahrscheinlichkeit von $\geq \frac{1}{4}$, so dass man im Durchschnitt nur vier Versuche braucht, um eine gültige Signatur zu erhalten.

Um ein besseres Ergebnis zu erhalten, so dass zum Beispiel nur 75 Abweichungen entstehen, kann ein Angreifer alle möglichen Lösungen der linearen Gleichungen durchsuchen. Diese Möglichkeit ist aber nicht sehr effektiv, auch wenn die Entwickler des Angriffs angeben, dass „nur“ 128^9 Möglichkeiten durchsucht werden müssen und dies in „relativ kurzer Zeit“ durchzuführen sei. Jedoch entspricht diese Methode daher eher der nicht zu bevorzugenden Brute-Force Methode und bei deutlich größeren Parametern ist eine schnelle Berechnung nicht mehr sichergestellt. Ein wesentlich besseres Angriffsverfahren, dass einem Fälscher auch oftmals Lösungen ermöglicht, die bei der Abweichungsbedingung auch die untere Schranke $D_{\min}$ unterbietet, wird im Folgenden vorgestellt.

7.1.3. Gitterreduktion

Unter *Gitterreduktion* versteht man ein Verfahren, mit dessen Hilfe man in Gittern nützliche $\mathbb{Z}$-Basen finden kann. Ein effektiver Algorithmus dafür ist der von Lenstra, Lenstra und Lovász entwickelte LLL-Algorithmus, der in polynomialer Zeit arbeitet.[8] Im Folgenden werde ich jedoch nicht im Speziellen auf LLL eingehen, sondern nehme an, dass ein passender auf LLL aufbauender Algorithmus als Black-Box-Algorithmus vorliegt.

Sei nun (s'', t'') eine Signatur, die mit dem im vorherigen Abschnitt vorgestellten Fälschungsangriff erzeugt wurde. Bei diesem Angriff ist jedoch nicht garantiert, dass der Angreifer die Abweichungsbedingung auch dann erfüllen kann, wenn die Grenzen $(D_{min}, D_{\max})$ strikter gewählt werden. In diesem Abschnitt werde ich erläutern, wie man mit Hilfe von Gitterreduktionsmethoden selbst deutlich striktere Abweichungsgrenzen einhalten kann. Dafür wird ein hybrides Verfahren verwendet, bei dem zuerst mit dem oben vorgestellten Verfahren eine Initialsignatur gebildet wird. In einem zweiten Schritt werden einige der ursprünglichen Abweichungen mit Hilfe von Gitterreduktion korrigiert, so dass die gefälschte Signatur letztendlich durchschnittlich 56 Abweichungen besitzt. Dabei sollte man anmerken, dass für die untere Schranke im Standardfall $D_{\min} = 55$ gilt. Somit erhält man mit diesem Verfahren zum Teil sogar „zu gute“ Fälschungen. Um zu vermeiden, dass eine Signaturprüfung fehlschlägt, weil die Fälschung „zu gut“ ist, kann man im Bedarfsfall bei den freien Koeffizienten, die auf

$$\begin{aligned} s_i &\equiv (f_0 \circledast m)_i \pmod{p} \\ t_j &\equiv (g_0 \circledast m)_j \pmod{p} \end{aligned}$$

gesetzt wurden, absichtlich Abweichungen einbauen.

Die Entwickler geben den Zeitbedarf für die Berechung einer gefälschten Signatur mit wenigen Minuten an. Auch wenn diese Zeitschätzung vielleicht in einigen Fällen etwas optimistisch geschätzt ist, so zeigt die angegebene Größenordnung jedoch, dass NSS in

[8] vgl. [LLL82]

seiner ursprünglichen Version mit diesem Angriff gebrochen ist und als unsicher angesehen werden muss.

Sei nun, wie eben eingeführt, (s'', t'') eine Initialsignatur. Nachdem $t'' \equiv s'' \circledast h \pmod{q}$ gilt, liegt der Vektor (s'', t'') in dem Gitter, dass von der Matrix

$$L_{\mathrm{CS}} = \begin{bmatrix} I_{(N)} & M_h \\ 0 & qI_{(N)} \end{bmatrix}$$

erzeugt wird. Dabei bezeichnet $I_{(N)}$ die N-dimensionale Einheitsmatrix.

Wie auch schon im vorherigen Abschnitt eingeführt, kann man die Matrix M_h durch geschicktes Umnummerieren der Koeffizienten so aufteilen, dass in der rechten unteren Ecke eine invertierbare $(k \times k)$-Matrix U entsteht. Damit kann man ebenfalls die Gittermatrix L_{CS} umsortieren, so dass man eine neue Gittermatrix $L_{\mathrm{CS},2}$ für das gleiche Gitter erhält mit der Eigenschaft

$$L_{\mathrm{CS},2} = \begin{bmatrix} I_{(N-k)} & 0 & R & S \\ 0 & I_{(k)} & T & U \\ 0 & 0 & qI_{(N-k)} & 0 \\ 0 & 0 & 0 & qI_{(k)} \end{bmatrix}$$

Wie auch im vorherigen Abschnitt angemerkt, kann man, auch wegen der Invertierbarkeit von U, bei s'' und t'' die Koeffizienten so umsortieren, dass die ersten k Koeffizienten von s'' und die letzten k Koeffizienten von t'' so gewählt sind, dass diese keine Abweichungen generieren. Die Aufgabe des Angreifers besteht nun darin, die verbliebenen $N - k$ Koeffizienten von s'' beziehungsweise t'', die möglicherweise Abweichungen besitzen, zu korrigieren, ohne dass neue Abweichungen generiert werden. Dazu sucht der Angreifer im Gitter, dass durch $L_{\mathrm{CS},2}$ generiert wird, eine Menge von *harmlosen* Zeilenvektoren mit der Eigenschaft, dass die ersten k und letzten k Einträge gleich Null sind. Sei nun (v_s, v_t) ein solcher harmloser Zeilenvektor, so hat $(s'' + v_s, t'' + v_t)$ weiterhin an den ersten k und den letzten k Positionen keine Abweichung und in allen anderen Koeffizienten sind möglicherweise weniger Abweichungen enthalten.

Diese gewünschte Menge der harmlosen Zeilenvektoren kann man erhalten, wenn für das Gitter, das von $L_{\mathrm{CS},2}$ erzeugt wird, eine neue Basis $L_{\mathrm{CS},3}$ gewählt wird, so dass mit $V := SU^{-1} \pmod{q}$

$$L_{\mathrm{CS},3} = \begin{bmatrix} I_{(N-k)} & -V & R - VT & 0 \\ 0 & I_{(k)} & T & U \\ 0 & qI_{(k)} & 0 & 0 \\ 0 & 0 & qI_{(N-k)} & 0 \\ 0 & 0 & 0 & qI_{(k)} \end{bmatrix}$$

An dieser Stelle ist zu zeigen, dass sich die Basis von $L_{\mathrm{CS},3}$ durch die Basis von $L_{\mathrm{CS},2}$ darstellen lässt. Dazu sei wie oben angegeben $V := SU^{-1} \pmod{q}$, sowie

$$A = \begin{bmatrix} I_{(N-k)} & -V & -VT & -S \\ 0 & I_{(k)} & 0 & 0 \\ 0 & 0 & I_{(N-k)} & 0 \\ 0 & 0 & 0 & I_{(k)} \end{bmatrix},$$

das heißt, A ist eine invertierbare Matrix mit Determinante 1. Somit gilt $A \cdot L_{\mathrm{CS},2}$ ist wieder eine Basis mit

$$A \cdot L_{\mathrm{CS},2} = \begin{bmatrix} I_{(N-k)} & -V & R-VT & 0 \\ 0 & I_{(k)} & T & U \\ 0 & 0 & qI_{(N-k)} & 0 \\ 0 & 0 & 0 & qI_{(k)} \end{bmatrix}$$

Die Zeile $[0 \;\; qI_{(k)} \;\; 0 \;\; 0]$ in $L_{\mathrm{CS},3}$ ist nur eine kosmetische Erweiterung der Basis, da diese neuen Vektoren linear abhängig von den ursprünglichen Basisvektoren sind und deshalb die Basis nicht verändern. Damit beschreiben $L_{\mathrm{CS},2}$ und $L_{\mathrm{CS},3}$ ein identisches Gitter.

Betrachtet man die Matrix $L_{\mathrm{CS},3}$ genauer, so kann man erkennen, dass in den Zeilen $k+1$ bis $N-k$ und $N+1$ bis $2N$ alle Koeffizienten an den ersten k und den letzten k Positionen gleich Null sind. Offensichtlich sind diese Vektoren linear unabhängig und aus der Matrix L_{harmlos} dieser Vektoren kann man ein Gitter der Dimension $(2N-2k)$ erzeugen. Daraus erzeugen wir ein weiteres Gitter:

$$L_{pq} = \begin{bmatrix} pL_{\mathrm{harmlos}} \\ (s', t') \end{bmatrix},$$

wobei (s', t') der Zeilenvektor ist, dessen Koeffizienten folgende Bedingungen erfüllen:

$$\begin{aligned} s' &\equiv s'' \pmod{q} \\ t' &\equiv t'' \pmod{q} \\ s' &\equiv (f_0 \circledast m) \pmod{p} \\ t' &\equiv (g_0 \circledast m) \pmod{p} \end{aligned}$$

Kurze Vektoren entsprechen in diesem Gitter Vektoren mit wenigen Abweichungen. Das optimale Ergebnis kann man nun erreichen, indem man einen harmlosen Vektor sucht, der addiert zu (s'', t'') ein sehr kurzer Vektor ist. Unglücklicherweise ist dies ein Beispiel des CVP, das nicht effizient lösbar ist.

Sei nun (v_s, v_t) ein Vektor aus dem Gitter, das von L_{pq} erzeugt wird. Dann gilt zum einen $v_s \circledast h \equiv v_t \pmod{q}$ und zum anderen gilt abhängig von den Koeffizienten von (s', t') eine der folgenden drei Gleichungen modulo p:

$$\begin{aligned} v_s \equiv v_t \equiv 0 &\pmod{p} \\ v_s \equiv (f_0 \circledast m) &\pmod{p} \text{ und } v_t \equiv (g_0 \circledast m) \pmod{p} \\ -v_s \equiv (f_0 \circledast m) &\pmod{p} \text{ und } -v_t \equiv (g_0 \circledast m) \pmod{p} \end{aligned}$$

Wenn man nun einen Vektor (v_s, v_t) mit kleinen Koeffizienten, zum Beispiel im Intervall $(-\frac{q}{2}, \frac{q}{2}]$, finden kann, der nicht die erste Bedingung $v_s \equiv v_t \equiv 0 \pmod{p}$ erfüllt, so würde sowohl (v_s, v_t) als auch $(-v_s, -v_t)$ eine gültige Fälschung ohne Abweichungen generieren. Wie jedoch schon eingeführt, ist das Finden eines solchen Vektors nicht in praktikabler Zeit durchführbar in einem Gitter der Größe von L_{pq}.

Anstatt nun Gitterreduktion auf dem gesamten Gitter L_{pq} durchzuführen, wählt ein Angreifer sich c Spalten von L_{pq} entsprechend zu den noch nicht festgelegten Koeffizienten von (s'', t'') und definiert sich daraus eine Untermatrix L_{final}. Diese ist dann nur noch c-dimensional, so dass Gitterreduktion durchführbar ist. Jeder Koeffizient des sich daraus ergebenen c-dimensionalen Ergebnisvektors ist nun ohne Abweichung, wenn er im Intervall $(-\frac{q}{2}, \frac{q}{2}]$ liegt.

Im Allgemeinen ist die erwartete Anzahl von Abweichungen von s beziehungsweise von t nach diesem Prozess gleich $\frac{2N-2k-c}{3} + \frac{\Delta}{2}$, wobei Δ die erwartete Anzahl der Koeffizienten des c-dimensionalen Ergebnisvektors ist, die außerhalb des Intervalls $(-\frac{q}{2}, \frac{q}{2}]$ liegen.

Abschließend geben die Entwickler des Angriffs noch praktische Empfehlungen für die Variablen c und k. Sie geben an, dass bei der Wahl von $c = 150$, $k = 95$ und einer Blockgröße[9] von 20 der Gitterreduktionsalgorithmus nach wenigen Minuten terminiert und das Ergebnis eine gefälschte Signatur ist, deren Abweichung im Mittel bei 56 liegt.

7.1.4. Gegenmaßnahmen in R-NSS

Nachdem die Abweichungen der Fälschungen im Mittel bei 56 liegen, wäre die erste Maßnahme, die Grenzen $D_{\min}$ und $D_{\max}$ anzupassen, damit dieser Angriff nicht mehr funktioniert. Dies hat jedoch zwei entscheidende Nachteile:

1. Der vorgestellte Angriff kann eventuell noch weiter verbessert werden. Gerade eine andere Wahl von c und k ermöglichen eventuell noch bessere Fälschungen. Um dies zu verhindern, müsste man $D_{\min}$ und $D_{\max}$ entsprechend klein wählen.

2. Eine zu kleine Wahl von $D_{\min}$ und $D_{\max}$ verringert die Effizienz des Verfahrens erheblich, da beim Erstellen einer korrekten und gültigen Signatur der Ersteller diese Grenzen auch einhalten muss. Liegt eine erstellte Signatur nicht im gewünschten Intervall, muss diese verworfen und eine neue generiert werden. Die Wahrscheinlichkeit, eine gültige Signatur zu erzeugen sinkt jedoch, wenn die Grenzen $D_{\min}$ und $D_{\max}$ zu klein gewählt werden.

Aus diesen Gründen haben die Entwickler von NTRU bei der Überarbeitung von NSS die Verifikation von Signaturen grundsätzlich erneuert. Der einfache Abweichungstest wurde aufgegeben und dafür wurde eine Reihe von neuen Tests eingeführt, die garantieren sollen, dass eine Fälschung auch als solche erkannt wird. Warum die in Kapitel 6.7.3 vorgestellten drei Tests eine wie eben erzeugte Fälschung erkennen, werde ich im Folgenden erläutern.

In den drei Tests wird geprüft, ob die Norm der Signaturpolynome klein genug ist, ob die Koeffizienten von s und $t \equiv h \circledast s \pmod q$ eine normale Verteilung haben und auch ob die Abweichungen normal verteilt sind. Der Normtest ist dabei für einen Fälscher voraussichtlich kein Problem, da bei der Gitterreduktion höchstwahrscheinlich schon eine Signatur mit der gewünschten Normeigenschaft erzeugt wird. Schwieriger für einen Fälscher ist es, die anderen beiden Tests zu überwinden. Während der Erzeugung der Fälschung konnte man sich oft zunutze machen, dass man die Koeffizienten geeignet

[9] Die Blockgröße ist eine Inputvariable für Gitterreduktionsalgorithmen

umnummeriert. So ist zum Beispiel eingegangen, dass das gefälschte Polynom s vor der Gitterreduktion so umnummeriert wird, dass in den ersten k Positionen keine Abweichung existiert. Damit wurde die Dimension des Gitters von $2N$ auf $2N - 2k$ reduziert.

Eventuell kann man das hier vorgestellte Verfahren auch auf R-NSS anpassen. Dazu sind jedoch weitere Überlegungen nötig, wie man erreichen kann, dass sowohl die Positionen der Abweichungen als auch die Werte der Koeffizienten der Fälschung so verteilt sind, dass der Verifikationsprozess von R-NSS die Fälschung als gültige Signatur anerkennt. Im Weiteren werde ich jedoch noch andere Angriffe aufzeigen, so dass eine Anpassung nicht notwendig ist.

7.2. Zweiter Angriff auf NSS: Transcript Angriff

In diesem Kapitel werde ich einen Transcript-Angriff[10] beschreiben. Ziel dieses Angriffes ist, aus einem Transcript, das heißt, aus einer Menge gültiger Signaturen (m, s) den Private Key f zu berechnen. Das Entscheidende bei diesem Angriff ist die Anzahl der Signaturen, die gebraucht werden.

Wie auch in Kapitel 7.1 nehme ich wieder an, dass die Standardparameter $(N, q, p) = (251, 128, 3)$ verwendet werden. Dieser Angriff funktioniert aber auch mit anderen Parametersätzen, da er eine Schwachstelle von NSS ausnutzt. Weiterhin nehme ich an, dass für die hier verwendeten Polynome die für den Parametersatz $(N, q, p) = (251, 128, 3)$ empfohlenen Schlüsselraumparameter gelten, sodass das Polynom $m \in K(32, 32)$ und die Polynome w_1 und w_2 25 beziehungsweise 64 Koeffizienten ungleich Null haben. Weiterhin hat der Private Key f 140 Einträge ungleich Null.

In diesem Angriff wird nur eine Methode vorgestellt, die den Private Key f aus gegebenen Signaturen berechnen kann. Man könnte ebenfalls das Ziel haben, g zu berechnen, da die Eigenschaft, dass g nur 80 Koeffizienten ungleich Null hat im Gegensatz zu f, das 140 Einträge ungleich Null hat, möglicherweise eine Konvergenzbeschleunigung bewirkt. Einen solchen Nachweis, ob ein Angriff auf g deutlich schneller ist, gibt es bisher noch nicht. Die Konvergenzgeschwindigkeit von einem Angriff auf f reicht jedoch aus, um das Signaturverfahren NSS zu brechen.

7.2.1. Grundzüge des Angriffs

Sei (m, s) eine mit NSS erstellte gültige Signatur. Dann kann man daraus das zweite Signaturpolynom $t \equiv s \circledast h \pmod{q}$ berechnen. Das Prinzip des Angriffs besteht nun darin, die Verteilungen der Koeffizienten von s und t für eine Untermenge aller Nachrichten m zu untersuchen. Dabei kann man erkennen, dass die Verteilung der Koeffizienten gegen eine Grenzverteilung konvergiert, wenn man einen Koeffizienten von m auf einen Wert fixiert. Diese Grenzverteilung hängt von einem ausgewählten Koeffizienten von f oder g ab. Daher werde ich im Folgenden stichprobenmäßig Koeffizientenverteilungen von s und t mit erwarteten Werten der Grenzverteilung für jeden möglichen Wert der Koeffizienten von f und g vergleichen.

[10] Diese Bezeichnung haben die Entwickler des Angriffs in [GJSS01] gewählt.

In Kapitel 6.6.2 habe ich vorgestellt, wie der Signaturvorgang bei NSS abläuft. Dazu werden zwei Polynome w_1 und w_2 errechnet, mit denen dann die Signatur

$$s \equiv f \circledast (m + w_1 + pw_2) \pmod{q}$$

berechnen kann. Somit hängen die Koeffizienten von s nur vom Private Key f, der Nachricht m und den zwei zufällig generierten Polynomen w_1 und w_2 ab. Ebenso ist die Situation bei t, das durch

$$t \equiv g \circledast (m + w_1 + pw_2) \pmod{q}$$

berechnet wird.

Hat man nun genügend gültige Signaturen gesammelt, so sucht man sich alle Signaturen heraus, bei denen in m an der Stelle j_0 gilt, dass $m_{j_0} = 1$. Um den Koeffizienten von f an der Stelle k zu erhalten, berechnet man zusätzlich $i_0 \equiv j_0 + k \pmod{N}$ und belässt diese zwei Werte i_0 und j_0 bis zum Ende des Verfahrens fix.

Den Koeffizienten s_{i_0} kann man nun mit Hilfe der Konvolution durch

$$s_{i_0} \equiv \sum_{\substack{j+k \equiv i_0 \\ (\mathrm{mod}\ N)}} f_k(m_j + w_{1,j} + pw_{2,j}) \pmod{q} = \sum_{\substack{j+k \equiv i_0 \\ (\mathrm{mod}\ N)}} f_k W_j \pmod{q}$$

berechnen. Für die Größe W_j geben die Entwickler des Angriffs an, dass sie für jeden Index j annähernd gleichverteilt ist, wenn die Verteilung über zufällige Einträge von m genommen wird. Somit ist s_{i_0} annähernd eine Summe von 140 gleichverteilten zufälligen Variablen bezogen auf eine fixe Verteilung, da f 140 Einträge ungleich Null hat.

Durch die spezielle Wahl von $m_{j_0} = 1$, wobei man auch $m_{j_0} = -1$ oder $m_{j_0} = 0$ voraussetzen kann, sticht die Verteilung von W_{j_0} hervor. Die Entwickler betonen an dieser Stelle, dass sie die Beobachtung gemacht haben, dass abhängig von dem Wert f_k der Term $f_k W_{j_0}$ sich unterschiedlich mit einbringt.

Die exakte Verteilung von s_i könnte man basierend auf der komplexen Definition der Polynome w_1 und w_2 entwickeln.[11] Die Entwickler des Angriffs benutzen jedoch für diesen Angriff eine heuristische Argumentation, die sie in mehreren numerischen Experimenten bestätigt haben. Weiterhin schneiden sie das Thema an, dass man, wie oben erwähnt, m_{j_0} nicht auf Eins beschränken muss, sondern auch die Fälle $m_{j_0} = -1$ oder $m_{j_0} = 0$ betrachten kann, um weitere Informationen zu erhalten. Da sie jedoch weitestgehend darauf verzichten wollen, den Angriff noch weiter zu optimieren, führen sie den Gedankengang nicht zu Ende. Wie ich später zeigen werde, ist dies auch nicht unbedingt nötig, da der Angriff auch ohne weitere Verbesserungen effektiv genug ist, um NSS auch im praktischen Umfeld angreifbar zu machen. Die einzige Optimierung, die sie ansprechen, ist, dass man statt eines fest gewählten i_0 alle Indizes i betrachtet, für die gilt $m_j = 1$ und $i + j \equiv k \pmod{N}$. Da m 32 Koeffizienten gleich Eins hat, konvergiert der Angriff durch diese Anpassung um den Faktor 32 schneller. Daher wird die Verteilung von

$$s_{i,j} \equiv \sum_{\substack{j+k \equiv i_0 \ (\mathrm{mod}\ N) \\ m_j = 1}} f_k(m_j + w_{1,j} + pw_{2,j}) \pmod{q}$$

[11] vgl. dazu Kapitel 6.6.2

über einer großen Menge von Signaturen untersucht.

7.2.2. Effizienzbetrachtung

Um die erwartete Grenzverteilung zu erstellen, erzeugt man zuerst mehrere Millionen Nachrichten, die dann jeweils mit unterschiedlichen Private Keys verschlüsselt werden. Da bei der Entwicklung von NTRU gerade die Effizienz des Verfahrens eine wichtige Rolle gespielt hat, sollten auch eine große Anzahl von Signaturen schnell berechnet werden können. Bei all diesen Signaturen wird die Verteilung von s_k unter der Annahme berechnet, dass $m_j = 1$ und f_k ein bestimmter Wert in der Menge $\{-3, 0, 3\}$ ist. Daraus lassen sich dann die drei Grenzverteilungen F_{-3}, F_0 und F_3 von s_i definieren.

Hat ein Angreifer genügend Signaturen für eine Transcript-Angriff gesammelt, so kann er für jeden Index i die Verteilung s_i bestimmen und bezeichnet diese mit S_i. Als nächsten Schritt muss er S_i mit den Grenzverteilungen F_{-3}, F_0 und F_3 vergleichen. Dazu wird $S_i(x)$ als die Wahrscheinlichkeit definiert, dass $s_i = x$ für irgendein $x \pmod q$. Äquivalent werden auch $F_{-3,i}(x)$, $F_{0,i}(x)$ und $F_{3,i}(x)$ als die entsprechenden Wahrscheinlichkeiten definiert, dass $s_i = x$ unter der Annahme, dass $m_j = 1$, $i = j + k$ und f_k den vorgeschriebenen Wert hat.

Ein einfaches aber hilfreiches Maß zur Unterscheidung dieser Verteilungen ist

$$\Delta_i(v) = \sum_x (F_{v,i}(x) - A_i(x))(S_i(x) - A_i(x)),$$

wobei $A_i(x)$ der Durchschnitt von $F_{-3,i}(x)$, $F_{0,i}(x)$ und $F_{3,i}(x)$ ist. Dann wird für jeden Koeffizienten i der Wert $\Delta_i(v)$ für $v \in \{-3, 0, 3\}$ berechnet und anschließend werden die Werte $\Delta_i(-3)$ und $\Delta_i(3)$ der Größe nach geordnet. Wählt man nun die 70 kleinsten Werte aus, so kann man die Koeffizienten identifizieren, für die gilt $f = 3$ und $f = -3$.

Einzig der erste Eintrag von f hat durch die Konstruktion von f eine deutlich andere Verteilung, da es sich jedoch nur um einen einzelnen Index handelt, ist die Korrektur ohne großen Aufwand durchzuführen.

Die Entwickler des Angriffs haben die so prognostizierten Private Keys mit den aktuell verwendeten verglichen. Dabei ist herausgekommen, dass die meisten Fehler am Ende der Liste der 70 kleinsten Werte auftauchen. In der Tat haben sie insgesamt eine Untermenge von 40 Indices ausmachen können, die möglicherweise Fehler verursachen. Da es sich aber nur um eine geringe Anzahl handelt, ist es möglich, mit Hilfe von Brute-Force alle Möglichkeiten durchzuprobieren. Weiterhin geben sie an, dass es mit ihrer Methode möglich ist, den Private Key aus weniger als 100.000 Signaturen zu extrahieren. Als Ausblick geben sie noch an, dass bei weiteren Optimierungen der Angriff noch schneller ein Ergebnis bringt oder dass man auch mit Teilen, also ohne die 40 verdächtigen Indices, Fortschritte bei Gitterreduktionsangriffen erzielen kann.

7.2.3. Gegenmaßnahmen in R-NSS

Während der Schlüsselerzeugungsphase[12] wird der Private Key f aus einem öffentlich bekannten Polynom f_0 und einem zufällig gewählten Polynom f_1 generiert, so dass am Ende gilt:

$$f = f_0 + pf_1$$

Gleiches gilt auch für den zweiten Private Key g.

In R-NSS wurde ein weiteres zufälliges Polynom u mit eingeführt, das die öffentlich bekannten Polynome f_0 und g_0 ersetzt. Nachdem f_0 typischerweise auf 1 gesetzt wurde, konnte man beim Angriff die Verteilungen F_0, F_3 und F_{-3} betrachten, da f, bis auf den konstanten Term, nur Koeffizienten aus der Menge $\{-3, 0, 3\}$ hatte. Durch das Einfügen des neuen Polynoms u sind die Koeffizienten nun alle aus der Menge $\{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$.

Dadurch liegen die Verteilungen sehr dicht beieinander, so dass man selbst bei mehreren Millionen Signaturen keine eindeutigen Grenzverteilungen mehr ausmachen kann. Ohne diese ist das oben genannte Verfahren aber nicht mehr effizient durchzuführen.

7.3. Angriffe auf NSS: Eine kurze Bilanz

Die in Kapitel 7.1 und Kapitel 7.2 beschriebenen Angriffe zeigen, dass NSS gebrochen ist. Es ist nicht nur möglich, gefälschte Signaturen zu erstellen, sondern ein Angreifer kann auch den Private Key berechnen. Weiterhin habe ich erläutert, warum die beschriebenen Angriffe bei dem neuen Verfahren R-NSS nicht mehr funktionieren.

Auf der anderen Seite zeigen diese doch relativ einfachen Angriffe, dass es beim Entwickeln eines neuen Kryptosystems nichts bringt, sich ein schwieriges Problem zu suchen, in diesem Fall SVP, und zu behaupten, dass die Sicherheit des Kryptosystems alleine auf diesem Problem basiert. Deshalb ist auch NTRUEncrypt von diesen Angriffen nicht betroffen, da bei NTRUEncrypt die Verbindung zwischen dem schwierigen Problem und der Verschlüsselungsarithmetik besser gelöst ist.

7.4. Vorbereitung eines Angriffs auf R-NSS: Frühere Angriffe auf NTRUEncrypt und auf NTRU-basierte Signaturverfahren

Da der in den Kapiteln 7.5, 7.6 und 7.7 beschriebene Angriff zum Teil auf früheren Angriffen aufbaut, die gegen NTRUEncrypt oder gegen frühere Versionen von auf NTRU basierenden Signaturverfahren gerichtet waren, gebe ich hier einen kleinen Überblick über die relevanten Angriffe. Ich werde jedoch die Angriffe nicht im Detail erläutern, da dies keine weitere Erkenntnis liefert und an dieser Stelle nur die Grundprinzipien von Interesse sind.

[12] vgl. Kapitel 6.6.1

7.4.1. Coppersmith-Shamir

Wie schon erwähnt waren Coppersmith und Shamir[13] die ersten, die ernsthaft das NTRU-Kryptosystem untersuchten. Sie betrachteten dafür das gegebene Problem als Gitterproblem, um damit NTRU über Gitterreduktion anzugreifen. Sie wählten dafür die Basis

$$L_{\mathrm{CS}} = \begin{bmatrix} I_{(N)} & M_h \\ 0 & qI_{(N)} \end{bmatrix},$$

wobei $I_{(N)}$ für die n-dimensionale Einheitsmatrix und M_h die Konvolutionsmatrix von h ist. Nachdem $f \circledast h \equiv g \pmod{q}$, beinhaltet das Gitter auch den Vektor (f, g) und es ist auch sehr wahrscheinlich, dass dieser Vektor der kürzeste nichttriviale Vektor in L_{CS} ist. Somit entspricht das Erlangen des Private Keys dem Finden des kürzesten Vektors in L_{CS}. Für kleine N, zum Beispiel $N = 107$, kann man dies auch mit Gitterreduktionsalgorithmen effizient lösen.

Gitterreduktionsalgorithmen haben aber im Allgemeinen den Nachteil, dass sie entweder effizient oder genau sind, was damit zusammenhängt, dass das SVP für allgemeine Gitter bisher nicht in polynomialer Zeit gelöst werden kann. Somit erreichen die effizienten Algorithmen, wie beispielsweise der LLL, nicht den kürzesten sondern nur einen kurzen Vektor. So wird beim LLL-Algorithmus garantiert, dass der gefundene Vektor nicht länger ist als $2^{\frac{m-1}{2}}$ Mal der kürzeste Vektor.

Jedoch ist L_{CS} kein allgemeines Gitter und auch (f, g) ist ein Vektor mit speziellen Eigenschaften. Daher ist es fraglich, ob man die Komplexität des SVP von einem allgemeinen Gitter auf das NTRU-Gitter übertragen kann. Daher ist es beispielsweise May[14] gelungen, die Dimension des Gitters erheblich zu reduzieren, indem er ausnutzt, dass der Private Key g sehr viele Einträge gleich Null hat. Dafür betrachtet er sogenannte „zero-run"-Gitter, die eine deutlich geringere Dimension haben.

7.4.2. ggT-Gitter-Angriff

Nachdem direktes Reduzieren des Gitters L_{CS} nicht durchführbar ist, muss man sich überlegen, ob es Gitter mit kleinerer Dimension gibt, die trotzdem den Private Key enthalten. Die Entwickler von R-NSS haben ein solches Gitter bei der Entwicklung von NSS gefunden, dessen Dimension gleich N ist, wobei sie anmerken, dass das Reduzieren eines N-dimensionalen Gitters immer noch undurchführbar ist.[15] Jedoch ist eine Gitterreduktion auf einem N-dimensionalen Gitter ist deutlich einfacher als eine Gitterreduktion auf einem $2N$-dimensionalen Gitter wie L_{CS}.

Bei ihrer Untersuchung entdeckten sie, dass es einem Angreifer unter Umständen möglich ist, die Werte von einigen Vektoren $f \circledast w \in R$ „unreduziert" modulo q zu erlangen, so dass f der kürzeste Vektor im N-dimensionalen Gitter ist, das von $M_{f \circledast w}$ gebildet wird.

[13] vgl. [CS97]
[14] vgl. [May99b]
[15] vgl. [HPS96a]

Seien nun $f \circledast w_1$ und $f \circledast w_2$ zwei solche Vektoren. Dann entsprechen die Gitter $M_{f \circledast w_1}$ und $M_{f \circledast w_2}$ dem Ideal $I = <f \circledast w_1, f \circledast w_2>$.[16] Offensichtlich ist jedes Polynom im Ideal I ein multiplikatives Vielfaches von f und wenn die Polynome w_1 und w_2 teilerfremd sind, gilt sogar ggT $<f \circledast w_1, f \circledast w_2> = <f>$. Somit kann man diesen Angriff mit den Methoden des ggT auf Idealen durchführen.

Weiterhin kann man erkennen, dass die Polynome w_i, so wie sie in R-NSS gebildet werden, oftmals teilerfremd sind und gerade wenn man mehrere unreduzierte Signaturen findet, fällt die Konstruktion des N-dimensionalen Gitters nicht schwer. Zwar werden im Verlauf des R-NSS-Verfahrens Maskierungsmethoden verwendet, um das Errechnen von $f \circledast w$ in R zu erschweren, jedoch werde ich in Kapitel 7.5 zeigen, dass sich dieser Schutz aushebeln lässt.

7.4.3. Die Mittelwert-Angriff

Angenommen, ein Angreifer kann eine Menge von unreduzierten $f \circledast w$ abfangen. Daraus berechnet er den Durchschnitt

$$A_r = \frac{1}{r} \sum_{i=1}^{r} (f \circledast \overline{f}) \circledast (w_i \circledast \overline{w_i})$$

Für jedes i gilt, dass $(w_i \circledast \overline{w_i})_0 = ||w_i||_2^2$, was einer großen natürlichen Zahl entspricht. Für $k \neq 0$ hat $(w_i \circledast \overline{w_i})_k$ sowohl positive als auch negative Werte, die sich jedoch im Wesentlichen im Durchschnitt gegenseitig aufheben. Daher konvergiert A_r für steigende r im Wesentlichen gegen ein skalares Vielfaches von $f \circledast \overline{f}$. Nach ein paar Tausend Signaturen kann man eine gute Schätzung von $f \circledast \overline{f}$ erwarten, das heißt, man erhält eine Schätzung z, bei der für die meisten Koeffizienten gilt, dass $|z_i - (f \circledast \overline{f})_i| \leq 2$.

Selbst wenn man reduzierte Signaturen benutzt, kann man, mit einigen Korrekturen, eine Konvergenz erhalten, auch wenn diese etwa zehn mal langsamer ist.

7.5. Erster Teil des Angriffs auf R-NSS: Anheben der Signaturen

Der folgende Angriff zeigt, wie man aus einer kleinen Menge von gültigen Signaturen R-multiplikative Polynome des Private Keys f finden kann. Dazu berechnet ein Angreifer mit Hilfe des Public Keys folgende Elemente in R_q

$$\{f \circledast w_1 \pmod q, \ldots, f \circledast w_r \pmod q\} \text{ und } \{g \circledast w_1 \pmod q, \ldots, g \circledast w_r \pmod q\},$$

wobei (m_i, s_i) gültige abgefangene Signaturen sind und wie auch in Kapitel 6.7.3 gilt, dass $s_i \equiv f \circledast w_i \pmod q$ und $t_i \equiv h \circledast s_i \pmod q \equiv g \circledast w_i \pmod q$.

Im Verlauf des Angriffs werde ich zeigen, wie man diese Signaturen in den Ring R anheben kann, so dass man eine Liste

$$\{f \circledast w_1, \ldots, f \circledast w_r\} \text{ und } \{g \circledast w_1, \ldots, g \circledast w_r\},$$

[16] Für weitere Informationen bezüglich der Äquivalenz zwischen Gittern und Idealen: vgl. [GS02]

von Vielfachen von f und g erhält, die unreduziert modulo q sind. Dies ist ein verheerender Angriff gegen R-NSS, da ein Angreifer damit die Möglichkeit bekommt, den N-dimensionalen ggT-Angriff zu nutzen oder auch andere Angriffe, die später in dieser Arbeit vorgestellt werden.

7.5.1. Grundzüge des Angriffs

Aus der Signierphase[17] bekommt man folgende Gleichungen für jedes i

$$f \circledast w_i \equiv s_i \pmod{q} \text{ und } g \circledast w_i \equiv t_i \pmod{q}$$
$$f \circledast w_i \equiv g \circledast w_i \equiv m_i + e_i \pmod{p}$$

Würde man e_i kennen, so könnte man mit Hilfe des Chinesischen Restsatzes $f \circledast w_i$ und $g \circledast w_i$ modulo pq berechnen. Da die Koeffizienten von $f \circledast w_i$ und $g \circledast w_i$ aber fast alle im Intervall $(-\frac{pq}{2}, \frac{pq}{2}]$ liegen, kann man $f \circledast w_i$ und $g \circledast w_i$ ohne viele Probleme auch nach R heben.

Durch das Einfügen des Polynoms e_i ist das Heben der Signaturen nicht mehr direkt durchführbar. Bei den für R-NSS vorgeschlagenen Parametern wird e_i rund 20 Einträge ungleich Null haben, so dass ein simples Raten von e_i durch die große Anzahl der Möglichkeiten in der Größenordnung von $\binom{251}{20}2^{20}$ Möglichkeiten schwer durchzuführen ist.

Stattdessen wird eine iterative Approximation benutzt, die in jedem Schritt versucht, die vorherige Approximation zu verbessern. Sei dazu S_i die Approximation für $f \circledast w_i$ und T_i die entsprechende Approximation für $g \circledast w_i$. Zu Anfang werden S_i und T_i so gesetzt, dass sie in R_{pq} folgende Gleichungen erfüllen

$$S_i \equiv s_i \pmod{q} \text{ und } T_i \equiv t_i \pmod{q}$$
$$S_i \equiv T_i \equiv m_i \pmod{p}$$

In der Approximation gibt es nun zwei verschiedene Arten von Fehlern.

1. Der k-te Koeffizient von S_i und T_i ist falsch, falls der k-te Koeffizient von e_i ungleich Null ist.

2. Der k-te Koeffizient von S_i beziehungsweise von T_i ist zwar korrekt modulo pq, jedoch inkorrekt in R, falls der zugehörige Koeffizient außerhalb des Intervalls $(-\frac{pq}{2}, \frac{pq}{2}]$ liegt.

Im Schnitt werden etwa 25 der 251 Koeffizienten falsch sein.

Die Entwickler des Angriff beschreiben an dieser Stelle eine interessante Beobachtung: Für alle (i, j) gilt

$$(f \circledast w_i) \circledast (g \circledast w_j) - (f \circledast w_j) \circledast (g \circledast w_i) = 0$$

Basierend auf dieser Beobachtung kann man nun erwarten, dass $S_i \circledast T_j - S_j \circledast T_i$ im Laufe des Verfahrens gegen Null geht. Um dies zu testen, berechnet man in jedem Schritt die Norm $n_{ij} = ||S_i \circledast T_j - S_j \circledast T_i||_2$, um zu entscheiden, welche Korrekturen noch durchgeführt werden müssen oder wann das Verfahren ein korrektes Ergebnis ausgibt.

[17] vgl. Kapitel 6.7.2

7.5.2. Einige Anmerkungen zur Implementierung

Wie im vorherigen Abschnitt angesprochen, wird in jedem Schritt die Norm n_{ij} für jedes Approximationspaar (S_i, T_i) berechnet und dann in dem Produkt $P_i = \prod_{j \neq i} n_{ij}$ zusammengefasst. Für den nächsten Schritt wird zuerst das größte Produkt P_i und in diesem ein zufälliges Paar (S_i, T_i) gewählt. Nun werden an jeder Koeffizientenposition vorübergehend bestimmte Vielfache von q an S_i und T_i addiert oder subtrahiert und P_i neu berechnet. Die Vielfachen von q werden deshalb ausgewählt, da alle Approximationen schon korrekt modulo q sind. Nun merkt man sich diejenige Anpassung, die bei P_i die größte Abweichung erzeugt hat. Am Ende terminiert der Prozess genau dann, wenn das Normprodukt von irgendeinem Approximationspaar den Wert Null erreicht. An diesem Punkt erhält man zwei korrekte Approximationspaare.

In die Implementierung gehen einige Heuristiken, die auf den speziellen Eigenschaften des Signierprozesses basieren, mit ein, um die Geschwindigkeit zu verbessern. Speziell die Art, in der das Polynom e_i berechnet wird, macht es sehr wahrscheinlich, dass, wenn der k-te Koeffizient von e_i nicht Null ist, dann auch die k-ten Koeffizienten des Anfangsapproximationpaars (S_i, T_i) beide klein sind, das heißt, innerhalb des Intervalls $[-100, 100]$. Auf der anderen Seite ist es sehr wahrscheinlich, dass ein Koeffizient groß ist, das heißt, außerhalb des Intervalls $[-100, 100]$, wenn dieser Koeffizient im anfänglichen S_i oder T_i außerhalb eines Vielfachen von pq liegt.

Daraus folgt, dass man das oben erwähnte *bestimmte Vielfache* nun näher bestimmen kann. Wenn die entsprechenden Koeffizienten der anfänglichen Polynome S_i und T_i im Intervall $[-100, 100]$ liegen, so werden die Koeffizienten mit $\pm q$ angepasst. Liegt jedoch einer der Koeffizienten außerhalb von $[-100, 100]$, so wird die Anpassung mit $\pm pq$ durchgeführt. Die Vorzeichen richten sich danach, ob die anfänglichen Werte positiv oder negativ waren.

An dieser Stelle sollte man anmerken, dass nicht jeder Anpassungsschritt in die richtige Richtung geht und es häufiger mal vorkommt, dass ursprünglich korrekte Koeffizienten erst verfälscht und später wieder zurückgeändert werden. Das heißt, der Algorithmus verhält sich ähnlich wie eine Zufallsbewegung. Dies macht eine Laufzeitanalyse sehr schwer. Die Entwickler des Angriffs geben aber an, dass sie auf einem handelsüblichen Personal Computer[18] das Verfahren ausprogrammiert haben und ihr Ergebnis war, dass bei einem Transcript von vier Signaturen zwei Signaturen mit einer Wahrscheinlichkeit von 90% innerhalb von 25 Sekunden in den Ring R gehoben werden können. In den restlichen 10% konvergiert die Anzahl der Fehler nie gegen Null.

Zusammenfassend kann man sagen, dass dieser Algorithmus ermöglicht, bei einer sehr kleinen Anzahl von Signaturen, in diesem Fall sind es nur vier, sehr schnell die gewünschte Anhebung von f nach R zu erhalten.

[18] von 2002

7.6. Zweiter Teil des Angriffs auf R-NSS: Erlangen von $f \circledast \overline{f}$

Eine wichtiger Bestandteil des letzten Teils des Angriffs auf R-NSS ist das Produkt f mit seiner Umkehrung $\overline{f}$. Der folgende Gitterangriff ist abgeleitet von dem Angriff von Coppersmith und Shamir[19].

Wie in Bemerkung 6.16 angedeutet, ist die Umkehrung ein Automorphismus, daher gilt $(f \circledast \overline{f}) \circledast (h \circledast \overline{h}) \equiv (g \circledast \overline{g}) \pmod{q}$. Das heißt, dass der Vektor $((f \circledast \overline{f}), (g \circledast \overline{g}))$ im Gitter

$$L_{\text{norm}} = \begin{bmatrix} I_{(N)} & M_{h \circledast \overline{h}} \\ 0 & qI_{(N)} \end{bmatrix},$$

enthalten ist. Dieses Gitter hat zwar Dimension $2N$, es enthält jedoch auch ein $(N+1)$-dimensionales Untergitter von Palindromen, in dem auch $((f \circledast \overline{f}), (g \circledast \overline{g}))$ enthalten ist. Könnte man nun $((f \circledast \overline{f}), (g \circledast \overline{g}))$ aus diesem Gitter berechnen, so erhielte man wertvolle Informationen über f und g. Für typische NTRU beziehungsweise NSS-Parametersätze schlägt dieser Angriff jedoch fehl, da $((f \circledast \overline{f}), (g \circledast \overline{g}))$ normalerweise nicht der kürzeste Vektor in diesem Gitter ist.[20]

Es ist jedoch möglich, auf eine alternative Weise $f \circledast \overline{f}$ zu erhalten. Dafür kombiniert man die Ideen aus der ggT-Gitter-Angriff in Kapitel 7.4.2 mit der Mittelwert-Angriff aus Kapitel 7.4.3. Zuerst betrachtet man einige wenige unreduzierte Signaturen, um das Ideal $<f \circledast \overline{f}>$ zu formen. Dazu verwendet man einige unreduzierte Signaturenprodukte $s \circledast \overline{s} = f \circledast \overline{f} \circledast w_i \circledast \overline{w_i}$ und die Methoden aus Kapitel 7.4.2. Dann nimmt man den Unterring aller Palindrome aus $<f \circledast \overline{f}>$ und bildet daraus ein Gitter der Dimension $\frac{N+1}{2}$. Dieses Gitter entsteht aus dem Vektor $f \circledast \overline{f}$ und $\frac{N-1}{2}$ Vektoren der Form $(X^k + X^{N-k}) \circledast f \circledast \overline{f}$. Wie oben auch, ist auch hier im Allgemeinen $f \circledast \overline{f}$ nicht der kürzeste Vektor im Gitter.

Durch die Mittelwert-Angriff kann man jedoch eine gute Abschätzung t für $f \circledast \overline{f}$ finden. Verändert man das eben erzeugte Gitter, so dass t in diesem Gitter enthalten ist, so ist $t - f \circledast \overline{f}$ sicherlich der kürzeste Vektor, den man nun durch Gitterreduktion erhalten kann.

Dieser Angriff ist aus zwei Gründen sehr effektiv:

1. Die Dimension des Gitters ist nur $\frac{N+1}{2}$.
2. Selbst für sehr schlechte Abschätzungen t ist $||t - f \circledast \overline{f}||_2$ deutlich kleiner als $||f \circledast \overline{f}||_2$, so dass man nur wenige unreduzierte Signaturen braucht.

Am Ende geben die Entwickler des Angriffs noch an, dass sie mit nur zehn Signaturen innerhalb von weniger als zehn Sekunden den gewünschten Term $f \circledast \overline{f}$ gefunden haben.

7.7. Abschluss des Angriffs auf R-NSS: Orthogonale Kongruenz-Angriff

In diesem Kapitel beschreibe ich einen Algorithmus, der in Polynomialzeit den Private Key f aus $f \circledast \overline{f}$ und $f \circledast w$ extrahieren kann, solange $\overline{f}$ und w teilerfremd sind. Anders

[19] vgl. Kapitel 7.4.1
[20] Für mehr Details vgl. [GS02]

ausgedrückt braucht der Algorithmus $f \circledast \overline{f}$ und eine Basis B_f des Ideals $<f>$.

Der Algorithmus läuft nun folgendermaßen ab: Man beginnt mit einer großen Primzahl $P \equiv 1 \pmod{N}$. Danach versucht man mit Hilfe von $f \circledast \overline{f}$ und der Basis B_f durch Gitterreduktion $f^{P-1} \circledast a$ für ein a zu finden, dessen Norm $||a||_2 < \frac{P}{2}$. Anschließend benutzt man die Kongruenz $f^{P-1} \equiv 1 \pmod{P}$, um a zuerst modulo P und anschließend exakt auszurechnen. Damit kann man dann f^{P-1} berechnen und in einem letzten Schritt erhält man daraus f.

Im Folgenden werde ich die einzelnen Schritte dieses Algorithmus erläutern.

7.7.1. Orthogonale Gitter

Definition 7.2. Besitzt ein Gitter eine Basis von N gleich langen und senkrecht aufeinander stehenden Vektoren, so heißt dies ein *orthogonales Gitter*. Ein Polynom f heißt *orthogonales Polynom*, falls die Konvolutionsmatrix M_f ein orthogonales Gitter erzeugt.

Definition 7.3. Zwei Gitter heißen *homothetisch*, falls es bis auf einen konstanten Streckungsfaktor λ eine distanzerhaltende Abbildung von einem Gitter in das andere gibt.

Bemerkung 7.4. Alle orthogonalen Gitter sind homothetisch zum trivialen Gitter $\mathbb{Z}^N$.

Lemma 7.5. *Sei f ein orthogonales Polynom und a ein beliebiges Polynom. Dann gilt*

$$||f \circledast a||_2 = ||f||_2 \cdot ||a||_2$$

Wendet man für ein allgemeines Polynom f den LLL auf $<f>$ an, dann ist garantiert, dass man ein Vielfaches von f findet, beispielsweise $f \circledast a$, so dass die Norm $||f \circledast a||_2$ kleiner ist als ein spezieller Faktor mal die Norm des kürzesten Vektors im Gitter. Im Fall, wo f ein orthogonales Polynom ist, kann man $||a||_2$ als diesen Faktor setzen, da ja gilt, dass $||f \circledast a||_2 = ||f||_2 \cdot ||a||_2$. Somit kann man sicher sein, dass $||a||_2 < 2^{\frac{N-1}{2}}$.

7.7.2. Konstruktion eines implizit orthogonalen Gitters mit Hilfe von $f \circledast \overline{f}$

Nachdem B_f und M_f beides Basen von $<f>$ sind, gibt es eine unimodulare Matrix U mit $B_f = UM_f$. Das heißt, dass jede Zeile von B_f von der Form $f \circledast u_i$ ist, wobei u_i die i-te Zeile von U ist.

Angenommen, f ist kein Nullteiler in R, dann kann man durch $f \circledast \overline{f}$ dividieren, so dass man mit $D = M_{(1/(f \circledast \overline{f}))}$

$$B_f \cdot B \cdot B_f^T = U \cdot U^T$$

erhält. Dabei ist $U \cdot U^T$ die Gramsche Matrix der unbekannten unimodularen Matrix U. Obwohl man nichts Spezielles über U weiß, enthält $U \cdot U^T$ alle Informationen, die für die Durchführung des LLL nötig sind, um U zu reduzieren. Somit kann man eine speziell auf Gramsche Matrizen abgestimmte Version des LLL verwenden, dessen Ergebnis die unimodulare Transformationsmatrix V und die Gramsche Matrix des reduzierten Gitters

$(V \cdot U) \cdot (V \cdot U)^T$ ist. Bedingt durch die Grenzen des LLL wird die Norm der Zeilen der unbekannten Basis $V \cdot U$ durch $2^{\frac{N-1}{2}}$ beschränkt.

Nun kann man eine neue Basis von $<f>$ durch $(V \cdot U) \cdot M_f = V \cdot B_f$ berechnen, bei der man sicher sein kann, dass jede Zeile dieser Basis $f \circledast a_i$ mit $||a_i||_2 < 2^{\frac{N-1}{2}}$ entspricht.

Effektiv wurde hier gezeigt, wie man ein orthogonales Gitter reduziert, das von U aufgespannt wird, ohne jedoch eine explizite Basis dafür zu kennen.

7.7.3. Galois-Kongruenz

Sei P eine Primzahl mit $P \equiv 1 \pmod{N}$. Dann gilt für jedes Polynom f, wobei f kein Nullteiler in R_P ist, dass

$$f^P \equiv f \pmod{P}$$

und ebenfalls

$$f^{P-1} \equiv 1 \pmod{P}$$

Diese Kongruenzen lassen sich für beliebige Primzahlen P verallgemeinern, indem man eine *Galoissche Konjugationsfunktion* $\sigma(r) : R \to R$ verwendet, die definiert ist durch

$$f^{\sigma(r)}(x) = f(x^r)$$

Für jedes r, mit $\mathrm{ggT}(r, N) = 1$, definiert $\sigma(r)$ einen Automorphismus auf R. Da jeweils zwei Werte von r, die sich um ein Vielfaches von N unterscheiden, denselben Automorphismus definieren, gibt es insgesamt $N-1$ solche Automorphismen. Man nennt $\sigma(r)$ die *r-te Galoissche Konjugationsabbildung* und es gilt

$$f^P \equiv f^{\sigma(P)} \pmod{P}$$

Wie oben schon angesprochen, ist die Motivation solche Kongruenzen anzuschauen, dass man für ein gegebenes Vielfaches von f^{P-1}, in etwa $f^{P-1} \circledast a$, die Kongruenz $f^{P-1} \equiv 1 \pmod{P}$ benutzt, um nachzuweisen, dass

$$f^{P-1} \circledast a \equiv a \pmod{P}$$

Wenn a klein genug ist, das heißt, wenn alle Koeffizienten im Intervall $(-\frac{P}{2}, \frac{P}{2}]$ liegen, dann offenbart die Gleichung $f^{P-1} \circledast a \pmod{P}$ das Polynom a sofort und wenn man annimmt, dass a kein Nullteiler in R ist, dann kann man auf beiden Seiten durch a dividieren und erhält die exakten Werte von f^{P-1}.

Es ist jedoch nicht garantiert, dass a klein genug ist. Der LLL garantiert nur, dass $||a||_2 < 2^{\frac{N-1}{2}}$. Um nun sicher zu gehen, dass a nur Koeffizienten aus $(-\frac{P}{2}, \frac{P}{2}]$ hat, muss man P sehr groß wählen, im Worst Case auf $2^{\frac{N-1}{2}}$. Damit ist P exponentiell wachsend abhängig von N, womit eine Speicherung der Koeffizientenwerte unmöglich wird.

An dieser Stelle wäre der Angriff gescheitert, wenn die Entwickler des Angriffs nicht erkannt hätten, dass bei dem Angriff gar nicht nötig ist, mit f^{P-1} direkt zu arbeiten, sondern es ausreicht, wenn man f^{P-1} modulo irgendeiner Primzahl berechnen kann. Wie

später in Kapitel 7.7.4 diskutiert wird, kann man für solche Berechnungen ein Verfahren ähnlich dem wiederholten Quadrieren verwenden.

Um f zu erhalten, braucht man irgendeine Potenz von f, in der f unreduziert ist. Um dieses Problem zu lösen, sucht man sich eine weitere Primzahl $P' \equiv 1 \pmod{N}$ mit $\mathrm{ggT}(P-1, P'-1) = 2N$, für die man $f^{P'-1}$ modulo irgendeiner Primzahl berechnen kann. Mit Hilfe des Euklidischen Algorithmus kann man dann f^{2N} modulo dieser Primzahlen berechnen und mit Hilfe des Chinesischen Restsatzes kann man f^{2N} direkt berechnen. Mit f^{2N} kann man nun auch direkt arbeiten, da es nur polynomial mit N wächst.

In Kapitel 7.7.5 wird ein Verfahren vorgestellt, mit dem man den Private Key f aus der Potenz f^{2N} extrahieren kann.

7.7.4. Ideal-Potenz-Algorithmus

In der Praxis könnte es der Fall sein, dass kleinere P ausreichend sind, da diese erlauben, mit dem Ideal $<f^{P-1}>$ direkt zu arbeiten, jedoch muss man einige Tricks anwenden, um mit der sehr großen Primzahl hantieren zu können, die durch die LLL-Grenzen erzwungen wird. Angenommen, man hätte die Polynomketten $\{v_0^2 \circledast \overline{v_1}, \ldots, v_{r-1}^2 \circledast \overline{v_r}\}$ und $\{v_0 \circledast \overline{v_0}, \ldots, v_{r-1} \circledast \overline{v_{r-1}}\}$. Dann kann man die folgenden Berechnungen durchführen, die ähnlich dem wiederholten Quadrieren sind:

$$
\begin{aligned}
v_0^4 \circledast \overline{v_2} &= (v_0^2 \circledast \overline{v_1})^2 \circledast (v_1 \circledast \overline{v_1})^{-2} \circledast (v_1^2 \circledast \overline{v_2}) \pmod{P} \\
v_0^8 \circledast \overline{v_3} &= (v_0^4 \circledast \overline{v_2})^2 \circledast (v_2 \circledast \overline{v_2})^{-2} \circledast (v_2^2 \circledast \overline{v_3}) \pmod{P} \\
&\vdots \\
v_0^{2^r} \circledast \overline{v_r} &= (v_0^{2^{r-1}} \circledast \overline{v_{r-1}})^2 \circledast (v_{r-1} \circledast \overline{v_{r-1}})^{-2} \circledast (v_{r-1}^2 \circledast \overline{v_r}) \pmod{P}
\end{aligned}
$$

Somit kann man $v_0^{2^r} \circledast \overline{v_r} \pmod{P}$ effizient berechnen, auch wenn der Exponent 2^r sehr groß wird. Wenn man diesen Ansatz für $v_0^{P-1} \circledast \overline{v_r} \pmod{P}$ mit $||\overline{v_r}||_2 < \frac{P}{2}$ benutzen könnte, würde man $\overline{v_r}$ exakt berechnen. Weiterhin könnte man dann dieselben Polynomketten benutzen, um v_0^{P-1} modulo anderer Primzahlen zu berechnen.

Sei nun das Ideal $<f>$ in Form der Basisdarstellung $B_f = U \cdot M_f$ gegeben, dann kann man daraus das Ideal $<f^2>$ von den Zeilen $b_i \circledast b_j = u_i \circledast u_j \circledast f^2$ generieren. Da wir $f^2 \circledast \overline{f^2}$ kennen, können wir die in Kapitel 7.7.2 beschriebene Methode verwenden, um eine reduzierte Basis $B_{f^2} = U' M_{f^2}$ zu erhalten, wobei die Zeilen von U' eine Norm haben, die kleiner ist als $2^{\frac{N-1}{2}}$. Anschließend nehmen wir eine Zeile von B_{f^2} und nennen sie $f^2 \circledast \overline{v_1}$. Jetzt kann man $v_1 \circledast \overline{v_1}$ direkt berechnen sowie eine Basis von v_1: $U' \cdot M_{v_1}$. An dieser Stelle kann die Iteration von vorne begonnen werden, indem eine Basis für $<v_1^2>$ berechnet wird.

Satz 7.6. *Angenommen, v_0 ist kein Nullteiler in R. Sei weiterhin $k = \sum k_i 2^i$ mit $k_i \in \{0,1\}$ und $r = \lfloor \log_2(k) \rfloor$. Sei P eine Primzahl, so dass v_0 kein Nullteiler in R_P ist. Dann kann man für einen gegebenen Input $v_0 \circledast \overline{v_0}$ und eine Basis B_0 von $<v_0>$ die folgenden Polynomketten in Polynomialzeit in r, N und der Bitlänge des Inputs berechnen:*

$$\{v_0^{k_{r-1}} \circledast v_0^2 \circledast \overline{v_1}, \ldots, v_0^{k_0} \circledast v_{r-1}^2 \circledast \overline{v_r}\} \text{ und}$$

$$\{v_0 \circledast \overline{v_0}, \ldots, v_{r-1} \circledast \overline{v_{r-1}}\},$$

wobei für alle $i > 0$ gilt, dass v_i kein Nullteiler in R_P ist und $||v_i||_2 < 2^{\frac{N-1}{2}}$.

Benutzt man diese Polynomketten, so kann man $v_0^k \circledast \overline{v_r} \pmod P$ in Polynomialzeit berechnen. Ist $k = P-1 \geq 2^{\frac{N+1}{2}}$ mit $P \equiv 1 \pmod N$, so kann man $\overline{v_r}$ exakt berechnen und anschließend die obigen Polynomketten benutzen, um $v_0^{P-1} \pmod Q$ in Polynomialzeit zu berechnen für jede Primzahl Q, für die gilt, dass v_0 kein Nullteiler in R_Q ist.[21]

Bemerkung 7.7. Im Gegensatz zum Verfahren, das am Anfang des Kapitels vorgestellt worden ist, ist bei diesem Verfahren nicht nur eine deulich höhere Anzahl an Bedingungen an Nullteiler vorhanden, es ist bei dieser Version auch nicht festgelegt, das k eine Potenz der Zwei sein muss.

Satz 7.8. *Angenommen, f ist kein Nullteiler in R. Sei $f \circledast \overline{f}$ und eine Basis B_f von $< f >$ gegeben. Dann kann man f^{2N} in Polynomialzeit in N und der Bitlänge von f bestimmen.*

Beweis. Seien P und P' zwei Primzahlen, mit $\text{ggT}(P-1, P'-1) = 2N$, P und P' sind größer als $2^{\frac{N+1}{2}}$ und f ist weder in R_P noch in $R_{P'}$ ein Nullteiler. Mit Hilfe von Satz 7.6 kann man zwei Polynomketten berechnen, mit denen man $f^{P-1} \pmod{r_i}$ und $f^{P'-1} \pmod{r_i}$ in Polynomialzeit für jede Primzahl r_i berechnen kann, für die gilt, dass es in keiner Polynomkette einen Nullteiler v_i in R_{r_i} gibt. Benutzt man nun noch den Euklidischen Algorithmus, so berechnet man f^{2N} modulo jeder Primzahl r_i und schlussendlich erhält man f^{2N} exakt durch den Chinesischen Restsatz. □

7.7.5. Berechnung von f aus f^{2N}

Der letzte Schritt des Angriffs, ist f aus f^{2N} zu extrahieren. Dazu folgender Satz:

Satz 7.9. *Sei f^{2N} und $b \in \mathbb{Z}_N^*$ gegeben. In Polynomialzeit in b, N und der Bitlänge von f kann man nun $z = f^{\sigma(-b)} f^b$ berechnen.*

Beweis. Sei $P_2 > 2||f^{\sigma(-b)} f^b||_2$ eine Primzahl, so dass $P_2 = 2c_2N - b$ für eine beliebige natürliche Zahl c_2. Dann gilt:

$$\begin{aligned}(f^{2N})^{c_2} &\equiv f^{P_2} f^b \pmod{P_2} \\ &\equiv f^{\sigma(P_2)} f^b \pmod{P_2} \\ &\equiv f^{\sigma(2c_2N-b)} f^b \pmod{P_2} \\ &\equiv f^{\sigma(-b)} f^b \pmod{P_2}\end{aligned}$$

Da $P_2 > 2||f^{\sigma(-b)} f^b||_2$, stimmen die Koeffizienten der Moduloberechnung mit den unreduzierten überein und so erhält man $z = f^{\sigma(-b)} f^b$ exakt. □

[21] Für die Komplexitätsaussage vgl. [GS02]

Um f zu erhalten, sucht man eine komplexe N-te Einheitswurzel ζ und berechnet zunächst $f^{2N}(\zeta)$ und daraus die $2N$-te Wurzel, so dass man $f(\zeta)$ erhält. Indem man $-b$ als Primitivwurzel in $\mathbb{Z}_N$ wählt, kann man iterativ $f(\zeta^d)$ aus $f(\zeta)$ berechnen über die Formel:

$$f\left(\zeta^{(-b)^{i+1}}\right) = \frac{z\left(\zeta^{(-b)^i}\right)}{f^b\left(\zeta^{(-b)^i}\right)}$$

Bei dieser Berechnung ist z das Polynom aus Satz 7.9. Da $-b$ eine Primitivwurzel ist, erhält man $N-1$ linear unabhängige Gleichungen in den Koeffizienten von f, so dass man zusammen mit $f(1)$, das auch dem Angreifer bekannt ist, f vollständig bis auf Vorzeichen und Rotation, das heißt, bis auf eine Multiplikation mit $\pm X^k$, berechnen kann. Nachdem $k < N$ gilt, bleibt nur noch eine sehr kleine Anzahl von Polynomen, in der man den exakten Private Key f suchen muss.

7.8. Angriff auf R-NSS: Eine kurze Bilanz

Die ersten beiden Stufen des Angriffs sind heuristische Angriffe und daher ist eine Komplexitätsabschätzung nur sehr schwer möglich. Die Entwickler des Angriffs geben aber an, dass sie in der Praxis schnell sind, wie auch die Zeitangaben in Kapitel 7.5 und in Kapitel 7.6 zeigen.

Das Anheben der Signaturen hebt eine Menge Signaturen von R_q nach R durch Erlangung von unreduzierten Vielfachen von f in R. Dazu sind nach Angaben der Entwickler nur sehr wenige Signaturen nötig, um ein ausreichendes Ergebnis zu erhalten. Sie haben es geschaft, den Angriff mit nur vier gültigen Signaturen durchzuführen.

Die zweite Stufe des Angriffs benutzt die Mittelwert-Angriff, um $f \circledast \overline{f}$ zu approximieren und um anschließend das CVP zu lösen, um $f \circledast \overline{f}$ exakt auszurechnen.

Der Algorithmus der dritten Stufe benutzt die ersten beiden Schritte, um den Private Key in Polynomialzeit zu erhalten. Durch die Kombination von Gittermethoden und zahlentheoretischen Kongruenzen, kann der Algorithmus der letzten Stufe für Folgendes benutzt werden:

- Erlangen von f aus $f \circledast \overline{f}$ und einer Basis B_f von $<f>$.
- Erlangen von f nur aus B_f, wenn f ein orthogonales Polynom ist.
- Erlangen von $\frac{f}{\overline{f}}$ aus B_f, egal ob f ein orthogonales Polynom ist oder nicht.

Wie auch schon bei NSS Schwachstellen augezeigt wurden, zeigt dieser Angriff, dass R-NSS nicht viel sicherer ist und daher nicht benutzt werden sollte.

7.9. Angriff auf NTRUSign

NTRUSign hat genau wie die vorherigen Signaturverfahren das Problem, dass Informationen über den Private Key in die Signaturen mit einfließt. Daher ist es möglich, nach

dem Abfangen einer genügend großen Anzahl von Nachrichten, den Private Key zu extrahieren. Die Sicherheit von NTRUSign basiert nun auf der Mächtigkeit dieser Anzahl. Bei speziellen Parametersätzen wurde schon ein Angriff konstruiert, der mit nur 400 gültigen Signaturen den Private Key berechnen kann.[22] Jedoch ist dieser Angriff auf andere Parametersätze nicht übertragbar.

Allgemein ist die Situation momentan so, dass es für einige Parametersätze erfolgreiche Angriffe gibt, die sich jedoch nicht auf andere Parametersätze übertragen lassen. Für die übrigen Parametersätze benötigt man als Angreifer jedoch eine so große Anzahl an gültigen Signaturen, dass ein Angriff in der Praxis nicht durchführbar ist.

Der hier vorgestellte Angriff aud R-NSS lässt sich auch nicht einfach auf NTRUSign übertragen, da durch die Umformulierung des Verfahrens viel entfernt worden ist, was zum erfolgreichen Angreifen benötigt wird. So läuft der Signiervorgang nicht mehr in R_q ab, so dass schon der erste Teil des Angriffs ohne Angriffsziel ist.

Somit ist NTRUSign bisher noch ungebrochen.

[22] vgl. [Ngu06]

8. Fazit

Die Entwicklung von NTRU ist noch lange nicht abgeschlossen. Durch intensives Studieren der Verfahren sind bisher viele Lücken gefunden worden. Es ist den Entwicklern jedoch bis heute immer gelungen, die Systeme so zu modifizieren, dass die gefundenen Lücken geschlossen werden konnten und bis zum heutigen Zeitpunkt sind die aktuellen Kryptosysteme NTRUSign und NTRUEncrypt praktisch ungebrochen. Jedoch ist das Verfahren noch sehr jung und daher bleibt abzuwarten, ob sich NTRU gegen die momentan vorherrschende Kryptosysteme behaupten kann.

Über diesen hängt jedoch das Damoklesschwert in der Gestalt von Quantencomputern. Obwohl es wohl noch Jahre bis Jahrzehnte dauern wird, bis Quantencomputer weit genug entwickelt sind, dass sie für die angestammten Kryptosysteme gefährlich werden können,[1] sind schon heute Algorithmen bekannt, vor allem Shors Algorithmus, die mit Hilfe von Quantencomputern in polynomialer Zeit große Zahlen faktorisieren oder auch den diskreten Logarithmus berechnen können. Daher ist es wichtig, sich frühzeitig mit Alternativen zu beschäftigen und bisher sieht es so aus, als ob NTRU dafür ein Kandidat sein könnte. Vor allem ist interessant, dass die zugrundeliegenden Gitterprobleme SVP und CVP scheinbar nicht mit Hilfe von Quantencomputern in Polynomialzeit zu lösen sind.

Unabhängig von Quantencomputern ist es trotzdem wichtig, dass man immer wieder Alternativen zu gängigen Kryptosystemen sucht. So war, wie auch erwähnt, die Vigenère-Verschlüsselung etwa 300 Jahre ungebrochen. Trotz dieser langen Zeit wurde das Verfahren schlussendlich doch gebrochen und daher sollte man immer beachten, dass ein Kryptosystem nicht sicherer wird, je länger es ungebrochen ist. Gerade bei den Public Key-Verfahren, die im Vergleich zur langen Geschichte der Kryptologie noch relativ jung sind, muss noch viel in der Kryptoanalyse geforscht werden, damit bestehende Angriffe verbessert und neue entwickelt werden können.

Jedoch ist die Frage, die man sich bei jedem neuen Kryptosystem stellen muss, welchen Fokus das Kryptosystem haben soll. Auf der einen Seite steht das perfekt sichere Verfahren, wie zum Beispiel der One-Time-Pad, dessen Nachteil jedoch ist, dass der Schüssel dieselbe Länge haben muss wie der Klartext. Vor allem bei großen Datenmengen oder einer spontanen Kommunikation ist dies ein nicht zu vernachlässigender Nachteil. Auf der anderen Seite steht die Benutzbarkeit des Verfahrens, da je nach Größe und Umfang von Codebüchern oder Ähnlichem das Bedienen eines Kryptosystems sehr kompliziert wird.

Bei den modernen Public Key-Verfahren hat man versucht, einen ausreichenden Kompromiss zu finden. Bei diesen wird dem Benutzer des Kryptosystems erspart, Codebücher

[1] vgl. [VSB+01]: Ein erster Quantencomputer hat die Faktorisierung der Zahl 15 mit Hilfe von Shors Algorithmus korrekt ausgeführt.

oder Ähnliches zu benutzen, jedoch zu dem Preis, dass ein Angreifer nachrichtenunabhängig effektiv testen kann, ob ein gefälschter Schlüssel gültig ist, indem er einfach die Schlüsselerzeugungsphase des Kryptosystems verwendet.

Ein weiteres Problem moderner Kryptosysteme ist die Langzeitverschlüsselung, da niemand mit Sicherheit voraussagen kann, welche Schlüssellängen für wie viele Jahre ausreichend sind. Momentan geht man davon aus, dass bei sorgfältiger Konzeption und Implementierung kryptographische Verfahren eine Lebensdauer von fünf bis zwanzig Jahren haben.[2] Wie man adäquat darauf reagieren sollte, ist zur Zeit noch umstritten. Bedauerlicherweise ist dieses Problem bei NTRU genausowenig ein zentrales Thema wie auch bei den meisten anderen Kryptosystemen.

Beim Entwickeln eines neuen Kryptosystems ist es essentiell wichtig, dass man in einem frühen Entwicklungsstadium die neuen Ideen publik macht. Damit erreicht man viele erfahrene Kryptologen, die oftmals durch eine andere Sicht auf das Verfahren Schwachstellen oder Unsauberkeiten aufdecken, auf die man ohne Hilfe nie gekommen wäre. Dies haben die Entwickler von NTRU auch so praktiziert und deshalb sind bis heute auch so viele verschiedene verbesserte Versionen von auf NTRU basierenden Kryptosystemen veröffentlicht worden.

[2] vgl. [Ess08]

A. Gittertheorie

In diesem Kapitel werde ich den Begriff des *Gitters* erläutern.

Definition A.1. Sei $B = \{b_1, b_2, \ldots, b_d\}$ eine Menge linear unabhängiger Vektoren in $\mathbb{R}^n$ mit $d \leq n$. Ein *Gitter* $L(B)$ ist die Menge aller ganzzahligen Linearkombinationen der Vektoren aus B, das heißt,

$$L(B) := \left\{ v \in \mathbb{R}^n \;\middle|\; v = \sum_{i=1}^{d} a_i b_i, a_i \in \mathbb{Z} \right\}$$

Die Menge B wird *Basis* von $L(B)$ genannt und d ist die *Dimension* des Gitters.

Bemerkung A.2. Mit dieser Definition ist ein Gitter eine diskrete additive abelsche Untergruppe des $\mathbb{R}^n$.

Bemerkung A.3. Die hier in der Arbeit verwendeten Gitter haben alle *vollen Rang*, das heißt, es gilt $d = n$. Weiterhin schreibe ich in der Arbeit die Basis als Matrix, so dass

$$B = \begin{bmatrix} — & b_1 & — \\ — & b_2 & — \\ & \vdots & \\ — & b_n & — \end{bmatrix}$$

Definition A.4. Eine *Basistransformation* ist ein Übergang von einer Basis B in eine andere B', ohne dass das Gitter verändert wird. Für jede *Basistransformation* existiert eine *unimodulare* Matrix U, dass heißt, U ist eine invertierbare $n \times n$-Matrix mit $\det(U) = \pm 1$, für die gilt, dass $B' = U \cdot B$.

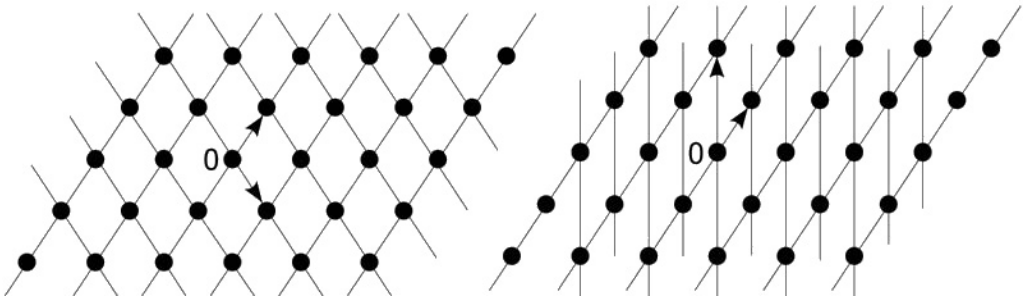

Abbildung A.1.: Ein Gitter mit zwei verschiedenen Basisdarstellungen

Bemerkung A.5. Somit gibt es für jedes Gitter unendlich viele Basen. In Abbildung A.1 ist ein solcher Basisübergang gezeigt.
Das Gitter, das von einer Basis erzeugt wird, ist jedoch eindeutig. Daher bezeichne ich ein Gitter mit seiner Basis, das heißt, wenn B eine Basis des Gitters $L(B)$ ist, so schreibe ich nur vom Gitter B und meine damit das von B erzeugte Gitter.

Definition A.6. Eine spezielle Basistransformation ist die *Gitterreduktion*. Dabei versucht man aus einer gegebenen Basis B eine *reduzierte Basis* B' desselben Gitters zu erzeugen. Dabei heißt *reduziert*, dass

1. die Basisvektoren weitestgehend orthogonal sind.
2. die Länge der Basisvektoren sind minimal.

Bemerkung A.7. Die Reduktion eines Gitters kann man mit Hilfe von Algorithmen wie dem LLL durchführen. Jedoch gibt es bisher noch keinen Algorithmus, der gleichzeitig effizient und genau ist.

In Gittern gibt es nun zwei Probleme SVP und CVP, die für allgemeine Gitter nicht effizient lösbar sind.

Definition A.8.

- Beim *CVP* geht es darum, für einen gegebenen Vektor a einen Gitterpunkt zu finden, der den kürzesten Abstand bezüglich eine bestimmten Norm zu a hat. Dabei ist a im Allgemeinen kein Gitterpunkt.
- Das *SVP* ist ein Spezialfall des CVP. Dabei sucht man einen Gitterpunkt $v \neq 0$, der den kürzesten Abstand bezüglich einer bestimmten Norm vom Nullpunkt hat.

B. Vorgestellte Kryptosysteme

Kryptosystem 1 Pig Latin – Verschlüsselung

Input: Klartext k
Output: Geheimtext g

1: **for all** Wörter w in k **do**
2: **while** w beginnt mit einem Konsonaten **do**
3: Setze den ersten Buchstaben von w an das Ende von w.
4: **end while**
5: Verlängere w mit den Buchstaben *ay*.
6: **end for**
7: Schreibe in g alle w in der richtigen Reihenfolge.
8: **return** g

Beispiel.

Klartext: Sicherheit ist das Hauptziel der Kryptologie!

Geheimtext: icherheitSay istay asday auptzielHay erday ologieKryptay!

Kryptosystem 2 Pig Latin – Entschlüsselung

Input: Geheimtext g
Output: Klartext k

1: **for all** Wörter w in g **do**
2: Verkürze w um die Buchstaben *ay*.
3: **while** w endet mit einem Konsonaten und w ist kein sinnvolles Wort **do**
4: Setze den letzten Buchstaben von w an den Anfang von w.
5: **end while**
6: **end for**
7: Schreibe in k alle w in der richtigen Reihenfolge.
8: **return** k

Bemerkung. Leider ist bei diesem Verfahren keine automatische Entschlüsselung möglich, da es vorkommen kann, dass zwei Wörter sich nur darin unterscheiden, dass das eine Wort am Anfang Konsonanten und das andere diese Konsonanten am Ende hat. Dies ist zum Beispiel bei den Wörtern *Amen* und *Name* so. Wenn diese beiden Wörter mit Hilfe des *Pig Latin-Verfahrens* verschlüsselt werden, entsteht beides Mal der Geheimtext *amenay*.

Kryptosystem 3 Atbash – Verschlüsselung

Input: Klartext k
Output: Geheimtext g

1: **for all** Wörter $w = (w_1, w_2, \ldots, w_n)$ in k **do** // $w_1, w_2, \ldots, w_n$ sind die Buchstaben der Wörter w
2: Setze A = das normale Alphabet und A' = das Atbash-Alphabet. // A entspricht also der ersten Zeile in Abbildung 3.1 und A' der zweiten.
3: **for** $i = 1$ to n **do**
4: Berechne $p(w_i)$ // Position des Buchstabens im Alphabet A
5: Setze $w_i = A'(p(w_i))$. // Nimm den Buchstaben aus A', der an der Position von w_i in A steht.
6: **end for**
7: **end for**
8: Schreibe in g alle w in der richtigen Reihenfolge.
9: **return** g

Beispiel.

Klartext: Sicherheit ist das Hauptziel der Kryptologie!

Geheimtext: Hrxsvisvrg rhg wzh Szfkgarvo wvi Pibkgloltrv!

Kryptosystem 4 Atbash – Entschlüsselung

Input: Geheimtext g
Output: Klartext k

1: **for all** Wörter $w = (w_1, w_2, \ldots, w_n)$ in g **do** // $w_1, w_2, \ldots, w_n$ sind die Buchstaben der Wörter w
2: Setze A = das Atbash-Alphabet und A' = das normale Alphabet. // A entspricht also der zweiten Zeile in Abbildung 3.1 und A' der ersten.
3: **for** $i = 1$ to n **do**
4: Berechne $p(w_i)$ // Position des Buchstabens im Alphabet A'
5: Setze $w_i = A(p(w_i))$. // Nimm den Buchstaben aus A, der an der Position von w_i in A' steht.
6: **end for**
7: **end for**
8: Schreibe in k alle w in der richtigen Reihenfolge.
9: **return** k

Bemerkung. Prinzipiell ist beim *Atbash-Verfahren* die Ver- und die Entschlüsselung identisch.

Kryptosystem 5 Skytale – Verschlüsselung

Input: Klartext k, Dicke des Stabes d // Mit Dicke des Stabes ist die Anzahl der Buchstaben gemeint, die auf den Stab passen, bevor man wieder an der Ausgangsposition ist.
Output: Geheimtext g

1: Setze $n =$ die Länge von k. // und damit $k = k_1, k_2, \ldots, k_n$
2: Zeilenbreite $z = \lceil \frac{n}{d} \rceil$ // aufrunden
3: **if** $n < z \cdot d$ **then**
4: Setze $k_{n+1}, k_{n+2}, \ldots, k_{z \cdot d} = \sqcup$ // Leerzeichen
5: **end if**
6: **for** $i = 1$ to $z \cdot d$ **do**
7: $g_i = k_j$ mit $j = (i-1) \cdot z + 1 \pmod{z \cdot d}$
8: **end for**
9: **return** g

Bemerkung. In diesem Algorithmus geben wir davon aus, dass der Klartext sehr gleichmäßig auf den Stab geschrieben wird und es nicht vorkommt, dass beispielsweise bei einer Dicke von 5 nur die ersten zwei Zeilen beschrieben werden.

Beispiel.

Klartext: Sicherheit ist das Hauptziel der Kryptologie!

Dicke des Stabes: 6

Geheimtext: Siaz oitsiKgc erihiHlyeesa p!rtudt h peo edtrl

Kryptosystem 6 Skytale – Entschlüsselung

Input: Geheimtext g, Dicke des Stabes d // Mit Dicke des Stabes ist die Anzahl der Buchstaben gemeint, die auf den Stab passen, bevor man wieder an der Ausgangsposition ist.
Output: Klartext k

1: Setze $n =$ die Länge von g. // und damit $g = g_1, g_2, \ldots, g_n$
2: Zeilenbreite $z = \lceil \frac{n}{d} \rceil$ // aufrunden
3: **if** $n < z \cdot d$ **then**
4: Setze $g_{n+1}, g_{n+2}, \ldots, g_{z \cdot d} = \sqcup$ // Leerzeichen
5: **end if**
6: **for** $i = 1$ to $z \cdot d$ **do**
7: $k_i = g_j$ mit $j = (i-1) \cdot d + 1 \pmod{z \cdot d}$
8: **end for**
9: **return** k

Bemerkung. Während beim Verschlüsseln mit der *Skytale* im Transpositionsschritt (Zeile 7 im Kryptosystem 5) die Zeichen um die Zeilenbreite verschoben werden, ist beim Entschlüsseln (Zeile 7 im Kryptosystem 6) die Dicke des Stabes entscheidend.

Kryptosystem 7 Caesar – Verschlüsselung

Input: Klartext k, Verschiebung v
Output: Geheimtext g

1: Setze $A =$ die Matrix, die durch die Abbildung 3.3 definiert ist. // ab Zeile 2, das heißt, das unveränderte Alphabet ist nur einmal in A in der Zeile 0 enthalten und Zeile 25 entspricht der letzten Zeile in der Abbildung.
2: **for all** Buchstaben $b \in k$ **do**
3: Setze $p(b)$ als die Position von b im Alphabet // Also $p(\text{a}) = 0, p(\text{b}) = 1, \ldots, p(\text{z}) = 25$.
4: Setze $b = A(v, p(b))$
5: **end for**
6: Setze $g = k$
7: **return** g

Beispiel.

Klartext: Sicherheit ist das Hauptziel der Kryptologie!

Verschiebung: 14

Geheimtext: Gwqvsfvswh wgh rog Voidhnwsz rsf Yfmdhczcuws!

Kryptosystem 8 Caesar – Entschlüsselung

Input: Geheimtext g, Verschiebung v
Output: Klartext k

1: Setze $v = 26 - v \pmod{26}$. // $\pmod{26}$, damit eine Verschiebung $v = 0$ auch wieder $v = 0$ wird.
2: Setze $A =$ die Matrix, die durch die Abbildung 3.3 definiert ist. // ab Zeile 2, das heißt, das unveränderte Alphabet ist nur einmal in A in der Zeile 0 enthalten und Zeile 25 entspricht der letzten Zeile in der Abbildung.
3: **for all** Buchstaben $b \in g$ **do**
4: Setze $p(b)$ als die Position von b im Alphabet // Also $p(\text{a}) = 0, p(\text{b}) = 1, \ldots, p(\text{z}) = 25$.
5: Setze $b = A(v, p(b))$
6: **end for**
7: Setze $k = g$
8: **return** k

Bemerkung. Hat man ein anderes Alphabet als das deutsche, so muss man in Zeile 1 die Zahl 26 durch die Anzahl der Zeichen im neuen Alphabet ersetzen.

Kryptosystem 9 Vigenère mit Anreihung des Schlüssels / One-Time-Pad – Verschlüsselung

Input: Klartext k, Schlüsselwort w
Output: Geheimtext g

1: Setze $n =$ die Anzahl der Zeichen in k.
2: Passe w so an, dass die Anzahl der Zeichen in w gleich n und setze $s = w$. // Wenn w zu lang ist, wird es durch Abschneiden der hinteren Zeichen verkürzt, ist w zu kurz, wird das Schlüsselwort so lange, eventuell beim letzten Mal nicht vollständig, wiederholt, bis w die Länge n hat.
3: Setze $A =$ die Matrix, die durch die Abbildung 3.5 definiert ist. // die erste Zeile der Abbildung entspricht in A der Zeile 0 und Zeile 25 entspricht der letzten Zeile in der Abbildung.
4: **for all** Buchstaben $b \in k$ und $b' \in s$ **do** // Hier müssen natürlich immer die Zeichen an der gleichen Position gleichzeitig betrachtet werden.
5: Setze $p(b)$ als die Position von b im Alphabet und $p(b')$ als die Position von b'. // Also $p(\text{a}) = 0, p(\text{b}) = 1, \ldots, p(\text{z}) = 25$.
6: Setze $b = A(p(b') + 1 \pmod{26}, p(b))$
7: **end for**
8: Setze $g = k$
9: **return** g

Bemerkung. Der *One-Time-Pad* benutzt denselben Code. Wichtig beim *One-Time-Pad* ist nur das Schlüsselwort, das dieselbe Länge wie der Klartext haben und vollständig zufällig gewählt muss.

Beispiel.

Klartext: Sicherheit ist das Hauptziel der Kryptologie!

Schlüsselwort: VIGENERE

⇒ **Schlüssel:** VIGENEREVI GEN ERE VIGENEREV IGE NEREVIGENER

Geheimtext: Opjkswzjec pxh isx Djbuheajh mlw Ywqupxstunw!

Bemerkung. Wie auch schon in Bemerkung 2.9 gesagt, sind auch hier die Leerzeichen beim Klartext und beim Schlüssel nur für die bessere Lesbarkeit mit angegeben. Ansonsten wäre dies ein zu großes Sicherheitsrisiko.

Kryptosystem 10 Vigenère mit Anreihung des Schlüssels / One-Time-Pad – Entschlüsselung

Input: Geheimtext g, Schlüsselwort v
Output: Klartext k

1: Setze $n =$ die Anzahl der Zeichen in g.
2: Passe w so an, dass die Anzahl der Zeichen in w gleich n und setze $s = w$. // Wenn w zu lang ist, wird es durch Abschneiden der hinteren Zeichen verkürzt, ist w zu kurz, wird das Schlüsselwort so lange, eventuell beim letzten Mal nicht vollständig, wiederholt, bis w die Länge n hat.
3: Setze $A =$ die Matrix, die durch die Abbildung 3.5 definiert ist. // die erste Zeile der Abbildung entspricht in A der Zeile 0 und Zeile 25 entspricht der letzten Zeile in der Abbildung.
4: **for all** Buchstaben $b \in g$ und $b' \in s$ **do** // Hier müssen natürlich immer die Zeichen an der gleichen Position gleichzeitig betrachtet werden.
5: Setze $p(b)$ als die Position von b im Alphabet und $p(b')$ als die Position von b'. // Also $p(\mathrm{a}) = 0, p(\mathrm{b}) = 1, \ldots, p(\mathrm{z}) = 25$.
6: Setze $v = 25 - p(b') \pmod{26}$
7: Setze $b = A(v, p(b))$
8: **end for**
9: Setze $k = g$
10: **return** k

Bemerkung. Hat man ein anderes Alphabet als das deutsche, so muss man in Zeile 6 die Zahl 26 beziehungsweise 25 durch die Anzahl der Zeichen im neuen Alphabet ersetzen.

Kryptosystem 11 ENIGMA – Verschlüsselung

Input: Klartext k, Schlüssel s
Output: Geheimtext g

for all Buchstaben $k_i \in k = (k_1, k_2, \ldots, k_n)$ **do**
  if $k_i \in s$[Steckbrett] **then** // das heißt, wenn der Buchstabe k_i im Steckbrett mit einem anderen Buchstaben verdrahtet ist
    Ersetze k_i nach Vorschrift s[Steckbrett].
  end if
  Ersetze k_i in der rechten Walze.
  Ersetze k_i in der mittleren Walze.
  Ersetze k_i in der linken Walze.
  Ersetze k_i in der Umkehrwalze.
  Ersetze k_i in der linken Walze.
  Ersetze k_i in der mittleren Walze.
  Ersetze k_i in der rechten Walze.
  if $k_i \in s$[Steckbrett] **then** // das heißt, wenn der Buchstabe k_i im Steckbrett mit einem anderen Buchstaben verdrahtet ist
    Ersetze k_i nach Vorschrift s[Steckbrett].
  end if
  Setze $g_i = k_i$
  Drehe die Walzen nach Nutstellungsregel weiter.
end for
return g

Beispiel.

Klartext: Sicherheit ist das Hauptziel der Kryptologie!

Schlüssel: Rotoren I,II,III und Positionen A,A,A in der ENIGMA-Simulation des Programms CryptTool Version 1.4.21 (`http://www.cryptool.de/`)

Geheimtext: Joekrttyja ban fmd Wlbeumvzg wxx Umscskrbxcn!

Kryptosystem 12 ENIGMA – Entschlüsselung

Input: Klartext g, Schlüssel s
Output: Geheimtext k

1: **for all** Buchstaben $g_i \in g = (g_1, g_2, \ldots, g_n)$ **do**
2: **if** $g_i \in s$[Steckbrett] **then** // das heißt, wenn der Buchstabe g_i im Steckbrett mit einem anderen Buchstaben verdrahtet ist
3: Ersetze g_i nach Vorschrift s[Steckbrett].
4: **end if**
5: Ersetze g_i in der rechten Walze.
6: Ersetze g_i in der mittleren Walze.
7: Ersetze g_i in der linken Walze.
8: Ersetze g_i in der Umkehrwalze.
9: Ersetze g_i in der linken Walze.
10: Ersetze g_i in der mittleren Walze.
11: Ersetze g_i in der rechten Walze.
12: **if** $g_i \in s$[Steckbrett] **then** // das heißt, wenn der Buchstabe g_i im Steckbrett mit einem anderen Buchstaben verdrahtet ist
13: Ersetze g_i nach Vorschrift s[Steckbrett].
14: **end if**
15: Setze $k_i = g_i$
16: Drehe die Walzen nach Nutstellungsregel weiter.
17: **end for**
18: **return** k

Kryptosystem 13 RSA – Schlüsselerzeugung

Input: Schlüssellänge l

Output: Public Key *pub*, Private Key *prv*

1: Suche Primzahl p und q, so dass $n = p \cdot q$ mindestens Länge l hat. // Hier werden im Allgemeinen zwei Zahlen zufällig gewählt und dann mit Hilfe eines Primzahltests, zum Beispiel dem Fermatscher Primzahltest, getestet, ob sie mit hoher Wahrscheinlichkeit Primzahlen sind. Selten gibt der Test ein falsches Ergebnis aus, so dass Zahlen als Primzahlen angesehen werden, die keine sind, wie zum Beispiel bei den Carmichael-Zahlen[1].

2: Berechne $\varphi(N) = (p-1) \cdot (q-1)$

3: Wähle eine Zahl e, die zu $\varphi(N)$ teilerfremd ist und $1 < e < \varphi(N)$ erfüllt.

4: Berechne d mit $e \cdot d \equiv 1 \pmod{\varphi(N)}$

5: Setze $pub = (e, N)$ und $prv = (d, N)$.

6: **return** *pub* und *prv*

Beispiel.

- Wir wählen $p = 11$ und $q = 13$ für die beiden Primzahlen und erhalten so $N = p \cdot q = 143$.
- Daraus ergibt sich $\varphi(N) = \varphi(143) = (p-1)(q-1) = 120$.
- Wir wählen $e = 23$, da 23 zu 120 teilerfremd ist und berechnen das von Inverse $d = 47$ von e zur Basis 120.

Somit erhalten wir den *Public Key* $(e, N) = (23, 143)$ und den *Private Key* $(d, N) = (47, 143)$.

[1] vgl. [Rib88]

Kryptosystem 14 RSA – Verschlüsselung

Input: Klartext k, Public Key (e, N)
Output: Geheimtext g
1: Berechne $g = k^e \pmod{N}$
2: **return** g

Beispiel.

Klartext: 7

Schlüssel: $(e, N) = (23, 143)$

Geheimtext: $g = 7^{23} \pmod{143} = 2$

Bemerkung. Mithilfe des RSA-Verfahrens werden nur Texte verschlüsselt, die vorher in Zahlen umgewandelt worden sind. Als Zuordnungsvorschrift Buchstaben $\leftrightarrow$ Zahlen bieten sich entweder einfache Zuordnungen wie *Leerzeichen* $\widehat{=}$ *00, A* $\widehat{=}$ *01, B* $\widehat{=}$ *02, . . . , Z* $\widehat{=}$ *26* an[2] oder man benutzt standardisierte Charactersets wie zum Beispiel ISO/IEC 8859-15[3].

Kryptosystem 15 RSA – Entschlüsselung

Input: Geheimtext g, Public Key (d, N)
Output: Klartext k
1: Berechne $k = k^d \pmod{N}$
2: **return** k

[2] vgl. [RSA78]
[3] vgl. [Mek03] und [KW99]

Kryptosystem 16 NTRU – Schlüsselerzeugung

Input: Sicherheitsparameter N, teilerfremde Zahlen p und q, Parameter d_f und d_g
Output: Public Key h und Private Key (f, g), sowie zu f inverse Polynome f_p und f_q.

1: **repeat**
2: Wähle f zufällig in $K(d_f, d_f - 1)$.
3: Berechne f_p und f_q mit Hilfe von Prozedur 1 und Prozedur 2.
4: **until** f_p und f_q konnten korrekt berechnet werden
5: Wähle g zufällig in $K(d_g, d_g)$.
6: Setze $h := p \cdot g * f_q$.
7: Setze $pub = h$ und $prv = (f, g, f_p, f_q)$.
8: **return** pub und prv

Prozedur 1: Polynominversion in ${}^{\mathbb{Z}_p[X]}\!/_{(X^N-1)}$

Input: Polynom $a(X)$, Primzahl p, Sicherheitsparameter N
Output: Inverses Polynom $b(X)$ modulo p

1: Setze $k := 0$, $b(X) := 1$, $c(X) := 0$, $f(X) := a(X)$, $g(X) := X^N - 1$
2: **loop**
3: **while** $f_0 = 0$ **do** // f_0 ist der konstante Koeffizient des Polynoms f
4: $f(X) := f(X)/X$, $c(X) := c(X) * X$, $k := k + 1$
5: **end while**
6: **if** $\deg(f) = 0$ **then**
7: $b() := f_0^{-1} b(X) \pmod p$
8: **return** $X^{N-k} b(X) \pmod{X^N - 1}$
9: **end if**
10: **if** $\deg(f) < \deg(g)$ **then**
11: vertausche f mit g und vertausche b mit c
12: **end if**
13: $u := f_0 g_0^{-1} \pmod p$
14: $f(X) := f(X) - u * g(X) \pmod p$
15: $b(X) := b(X) - u * c(X) \pmod p$
16: **end loop**

Prozedur 2: Polynominversion in ${}^{\mathbb{Z}_{p^r}[X]}\!/_{(X^N-1)}$

Input: Polynom $a(X)$, Polynom $b(X) \equiv a(X)^{-1} \pmod p$, Primzahl p, Exponent r und Sicherheitsparameter N
Output: Inverses Polynom $b(X)$ modulo p^r

1: Setze $q = p$
2: **while** $q < p^r$ **do**
3: $q = q^2$
4: $b(X) := b(X)(2 - (a(X)b(X) \pmod{X^N - 1})) \pmod q$
5: **end while**

Kryptosystem 17 NTRU – Verschlüsselung

Input: Klartext k, Public Key h, Parameter d_z und q
Output: Geheimtext g

1: Wähle z zufällig in $K(d_z, d_z)$.
2: Berechne $c \equiv z * h + k \pmod{q}$
3: **return** g

Kryptosystem 18 NTRU – Entschlüsselung

Input: Geheimtext g, Private Key (f, g), Public Key h, Inverses f_p, Parameter p und q
Output: Klartext k

1: Berechne $a = f * c \pmod{q}$.
2: Berechne $b = a \pmod{p}$.
3: Berechne $k = b * f_p \pmod{p}$.
4: **return** k

Abbildungsverzeichnis

Bildnachweis

Die in dieser Arbeit erstellten Abbildungen sind, unter anderem mit Microsoft® Office Visio®, selbst erstellt, mit folgendenden Ausnahmen:

- Logos auf der Titelseite (Quelle: `http://www.uni-augsburg.de` und `http://www.math.uni-augsburg.de`)
- Abb. 3.2, Abb. 3.4, Abb. 3.7, Abb. 5.1, Abb. 5.2 und Abb. 5.3 (Quelle: `http://wikipedia.org`)
- Abb. 3.6 (Quelle: [MQ06])
- Abb. 3.8 (Quelle: [Bau95])
- Abb. 3.9 (Quelle: [Kar07])
- Abb. 3.10 (Quelle: [BSW98])
- Abb. A.1 (Quelle: [Rüc07])

Liste der Kryptosysteme

Abkürzungsverzeichnis

CA-Zertifikat	Certification Authority-Zertifikat
CVP	Closest Vector Problem
ECC	Elliptic Curve Cryptography
GGH	Goldreich Goldwasser Halevi
IPsec	Internet Protocol Security
LLL	Lenstra Lenstra Lovász
MIT	Massachusetts Institute of Technology
NSS	NTRU Signature Scheme
NTRU	Number Theory Research Unit
NTRUEncrypt	NTRU Encryption Algorithm
NTRUSign	NTRU Signature Algorithm
oBdA	Ohne Beschränkung der Allgemeinheit
PIN	Persönliche Identifikationsnummer
R-NSS	Revised NTRU Signature Scheme
Room 40	Eine Abteilung des britischen Marine-Nachrichtendienstes während des Ersten Weltkriegs
RSA	Rivest Shamir Adleman
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
SVP	Shortest Vector Problem
TCP/IP	Transmission Control Protocol/Internet Protocol
W-LAN	Wireless LAN / Wireless Local Area Network
WEP	Wired Equivalent Privacy

Index

Literaturverzeichnis

[Bai06] Baier, Harald: *Vorlesung Kryptologie: Theoretische Grundlagen.* `http://fb2-linux.fh-bingen.de/~baier/WS06-07/Krypto/vorlesung_krypto_ws06_theorie.pdf`. Version: November 2006. – Teil einer im Wintersemester 2006/2007 gehaltenen Vorlesung im Fachbereich 2 der Fachhochschule Bingen

[Bau95] Bauer, Friedrich L.: *Entzifferte Geheimnisse : Methoden und Maximen der Kryptologie.* Berlin [u.a.] : Springer, 1995. – XII, 391 S. : Ill., graph. Darst.. – ISBN 3–540–58118–9. – Frühere Ausg. u.d.T.: Bauer, Friedrich L.: Kryptologie

[Beu07] Beutelspacher, Albrecht: *Kryptologie : eine Einführung in die Wissenschaft vom Verschlüsseln, Verbergen und Verheimlichen ; ohne alle Geheimniskrämerei, aber nicht ohne hinterlistigen Schalk, dargestellt zum Nutzen und Ergötzen des allgemeinen Publikums.* 8., aktualisierte Aufl. Wiesbaden : Vieweg, 2007 (Studium). – XII, 155 S. : Ill., graph. Darst.. – ISBN 978–3–8348–0253–8 — 978–3–8348–9209–6

[BSW98] Beutelspacher, Albrecht ; Schwenk, Jörg ; Wolfenstetter, Klaus-Dieter: *Moderne Verfahren der Kryptographie : von RSA zu Zero-Knowledge.* 2., verb. Aufl. Braunschweig [u.a.] : Vieweg, 1998. – X, 143 S. : Ill., graph. Darst.. – ISBN 3–528–16590–1

[BTPW07] Buchmann, Johannes ; Tews, Erik ; Pychkine, Andrei ; Weinmann, Ralf-Philipp: *aircrack-ptw : A tool to attack WEP.* `http://www.cdc.informatik.tu-darmstadt.de/aircrack-ptw/`. Version: April 2007. – Ein effektives Tool, mit dem man sehr schnell den WEP-Schlüssel entziffern kann. Entwickelt am Lehrstuhl Kryptographie und Computeralgebra an der Technischen Universität Darmstadt.

[Buc08] Buchmann, Johannes: *Einführung in die Kryptographie.* 4., erw. Aufl. Berlin [u.a.] : Springer, 2008 (Springer-Lehrbuch). – XXII, 274 S. : graph. Darst.. – ISBN 3–540–74451–7 — 978–3–540–74451–1

[Coh87] Cohen, Fred: *Introductory Information Protection.* Livermore : ASP Press, 1987 `http://all.net/books/ip/index.html`

[CS97] Coppersmith, Don ; Shamir, Adi: Lattice Attacks on NTRU. In: Fumy, Walter (Hrsg.) ; EUROCRYPT <16, 1997, Konstanz> (Veranst.): *Advances in cryptology - EUROCRYPT '97 : International Conference on the*

Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11 - 15, 1997 ; proceedings EUROCRYPT <16, 1997, Konstanz>, Springer, Mai 1997 (Lecture notes in computer science ; 1233), S. 52–61

[DKSW04] Deubler, Steffen ; Kronfeld, Thomas ; Steglich, Uwe ; Wagner, Maik: *Kryptologie: Allgemeiner Überblick - Geschichte - Hebräer, altes Testament (500 – 600 v. Chr.).* `http://www-user.tu-chemnitz.de/~uste/krypto/sources/gesch3.htm`. Version: 2004. – Im Rahmen der Vorlesung 'Mediengestaltung' an der Technischen Universität Chemnitz im Wintersemester 2003/2004 erstellt.

[Ess08] Esslinger, Bernhard: *Das CrypTool-Skript: Kryptographie, Mathematik und mehr.* `https://www.cryptool.org/download/CrypToolScript-de.pdf`. Version: Juli 2008. – Hintergrundmaterial und Zusatzinformationen zum freien E-Learning-Programm CrypTool; 9. Auflage

[GJSS01] Gentry, Craig ; Jonsson, Jakob ; Stern, Jacques ; Szydlo, Michael: Cryptanalysis of the NTRU Signature Scheme (NSS) from Eurocrypt 2001. In: Boyd, Colin (Hrsg.) ; ASIACRYPT <2001, Gold Coast, Queensland> (Veranst.): *Advances in cryptology - ASIACRYPT 2001 : 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9 - 13, 2001 ; proceedings* ASIACRYPT <2001, Gold Coast, Queensland>, Springer, Dezember 2001 (Lecture notes in computer science ; 2248), S. 1–20

[Goe86] Goethe, Johann Wolfgang v. ; Trunz, Erich (Hrsg.): *Faust : der Tragödie erster und zweiter Teil ; Urfaust.* 256. - 270. Tsd., Sonderausg. München : Beck, 1986. – 775 S.. – ISBN 3–406–31234–9. – Literaturverz. S. [761] - 775

[GS02] Gentry, Craig ; Szydlo, Mike: Cryptanalysis of the Revised NTRU Signature Scheme. In: Knudsen, Lars (Hrsg.) ; EUROCRYPT <21, 2002, Amsterdam> (Veranst.): *Advances in cryptology - EUROCRYPT 2002 : International Conference on the Theory and Application of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002 ; proceedings* EUROCRYPT <21, 2002, Amsterdam>, Springer, Mai 2002 (Lecture notes in computer science ; 2332), S. 299–320

[Hac08] Hachenberger, Dirk: *Mathematik für Informatiker.* 2., aktualisierte Aufl. München [u.a.] : Pearson Studium, 2008 (it-informatik). – XXXIII, 809 S. : graph. Darst.. – ISBN 978–3–8273–7320–5

[Heß06] Hess, Florian: *Aspekte der Kryptographie.* `http://www.math.tu-berlin.de/~hess/personal/ca-brief.pdf`. Version: Rohform, November 2006. – Eine Übersicht über Fragestellungen der Kryptographie ; Veröffentlicht am Institut für Mathematik der Technische Universität Berlin

[HHGP+03] Hoffstein, Jeffrey ; Howgrave-Graham, Nick ; Pipher, Jill ; Silverman, Joseph H. ; Whyte, William: NTRUSign: Digital Signatures Using the NTRU Lattice. In: Joye, Marc (Hrsg.) ; CT RSA <2003, San Francisco, Calif.> (Veranst.): *Topics in cryptology - CT-RSA 2003 : the Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13 - 17, 2003 ; proceedings* CT RSA <2003, San Francisco, Calif.>, Springer, April 2003 (Lecture notes in computer science ; 2612), S. 122–140. – Erweiterte Version unter `http://ntru.com/cryptolab/pdf/NTRUSign-preV2.pdf` abrufbar

[HHHGW09] Hirschhorn, Philip S. ; Hoffstein, Jeffrey ; Howgrave-Graham, Nick ; Whyte, William: *Choosing NTRU Parameters in Light of Combined Lattice Reduction and MITM Approaches.* März 2009. – Preprint ; International Conference on Applied Cryptography and Network Security — June 2-5, 2009

[HPS96a] Hoffstein, Jeffrey ; Pipher, Jill ; Silverman, Joseph H.: *NSS: The NTRU Signature Scheme.* `http://www.ntru.com/cryptolab/pdf/nss.pdf`. Version: November 1996. – Preprint

[HPS96b] Hoffstein, Jeffrey ; Pipher, Jill ; Silverman, Joseph H.: *NTRU: A new high speed public key cryptosystem.* August 1996. – Preprint ; presented at rump session of Crypto'96

[HPS98] Hoffstein, Jeffrey ; Pipher, Jill ; Silverman, Joseph H.: NTRU: A ring-based public key cryptosystem. In: Buhler, Joe P. (Hrsg.) ; ANTS <3, 1998, Portland, Or.> (Veranst.): *Algorithmic number theory : third international symposium, ANTS-III, Portland, Oregon, USA, June 21 - 25, 1998 ; proceedings* ANTS <3, 1998, Portland, Or.>, Springer, Juni 1998 (Lecture notes in computer science ; 1423), S. 267–288

[HPS01a] Hoffstein, Jeffrey ; Pipher, Jill ; Silverman, Joseph H.: Enhanced Encoding and Verification Methods for the NTRU Signature Scheme. In: *CryptoLab – Technical Notes* (2001), Mai. `http://www.ntru.com/cryptolab/pdf/NTRUTech017_v3.pdf`. – Report # 017, Version 2

[HPS01b] Hoffstein, Jeffrey ; Pipher, Jill ; Silverman, Joseph H.: NSS: An NTRU Lattice-Based Signature Scheme. In: Pfitzmann, Birgit (Hrsg.) ; EUROCRYPT <20, 2001, Innsbruck> (Veranst.): *Advances in cryptology - EUROCRYPT 2001 : International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6 - 10, 2001 ; proceedings* EUROCRYPT <20, 2001, Innsbruck>, Springer, Mai 2001 (Lecture notes in computer science ; 2045), S. 211–228

[HS00] Hoffstein, Jeffrey ; Silverman, Joseph H.: Optimizations for NTRU. In: Alster, Kazimierz (Hrsg.): *Public key cryptography and computatio-*

nal number theory : proceedings of the international conference ; Warsaw, Poland, September 11 - 15, 2000, de Gruyter, September 2000, S. 77–88

[Int09] Intel GmbH: *Das Mooresche Gesetz und die Zukunft -- Vierzig Jahre Mooresches Gesetz*. http://www.intel.com/cd/corporate/techtrends/emea/deu/209836.htm. Version: 13. Apr. 2009. – Leicht werbelastige Informationsseite mit Rück- und Ausblick in der Halbleitertechnologie

[JK05] Jasinska, Elisa ; Kopf, Gregor: *Enigma*. http://www.jasinska.de/uni/enigma.pdf. Version: Dezember 2005. – Vortrag zum Seminar 'Geschichte der Kryptologie' im Sommersemester 2005 am Institut für Informatik der Humboldt-Universität zu Berlin

[JN06] Jahn, Luisa ; Niebuhr, Toralf: *Kryptoanalyse*. http://www2.informatik.hu-berlin.de/~kreikenb/krypto/Sicherheit.pdf. Version: Oktober 2006. – Seminarvortrag zum Thema 'Sicherheit von Kryptosystemen' im Seminar Kryptologie im Wintersemester 2006/2007 am Institut für Informatik der Humboldt-Universität zu Berlin

[Kah96] Kahn, David: *The codebreakers : the story of secret writing*. [Nachdr.]. New York : Scribner, 1996. – XVIII, 1181, [24] S. : Ill., graph. Darst.. – ISBN 0–684–83130–9. – Literaturverz. S. 975 - 1130

[Kar07] Karbach, Nora: *ENIGMA*. http://www.math.uni-sb.de/ag-decker/WS0708Krypto/gruppen/vortraege/ENIGMA2.pps. Version: November 2007. – Vortrag zum Seminar 'Kryptologie' im Wintersemester 2007/2008 an der Fachrichtung Mathematik der Universität des Saarlandes

[Ker83] Kerckhoffs, Auguste: La Cryptographie Militaire. In: *Journal des Sciences Militaires* 9 (1883), Januar, 5–38. http://www.petitcolas.net/fabien/kerckhoffs/la_cryptographie_militaire_i.htm

[Ker00] Kerlin, Janet: Math professors patent computer security system. In: *George Street Journal* 25 (2000), September

[Kip99] Kippenhahn, Rudolf: *Verschlüsselte Botschaften : Geheimschrift, Enigma und Chipkarte*. Reinbek bei Hamburg : Rowohlt, 1999 (rororo ; 60807 : Sachbuch : rororo science). – 326 S. : Ill., grap. Darst.. – ISBN 3–499–60807–3. – Literaturverz. S. 394 - 400

[KNM03] Kessler, Marcel ; Näf, Michael ; Morelli, Ralph: *Kryptografie und Kryptoanalyse : Eine Einführung in die klassische Kryptologie*. http://www.swisseduc.ch/informatik/daten/kryptografie_gruppenarbeit/docs/zusammenfassung.pdf. Version: August 2003. – Unterrichtsmaterialien für eine Gruppenarbeit in der Sekundarstufe II

[Koi09] Koivunen, Toni: Calculating the Size of the Downadup Outbreak. In: *F-Secure Weblog* (2009), Januar. `http://www.f-secure.com/weblog/archives/00001584.html`

[Kra09] Krah, Stefan: *M4 Message Breaking Project.* `http://www.bytereef.org/m4_project.html`. Version: April 2009. – Nachdem die deutsche Marine während des zweiten Weltkriegs eine neue Version der ENIGMA, die ENIGMA M4, eingeführt hatten, konnten viele Funksprüche nicht mehr entziffert werden. Durch die fortschreitende Computertechnik werden viele Wettbewerbe veranstaltet, in denen es darum geht, verbliebene unentzifferte Nachrichten zu dechiffrieren.

[Kri08] Krimmel, Oliver: *Einführung in die Kryptologie.* `http://www.mi.uni-koeln.de/mi/Forschung/Kuepper/Kryptologie.pdf`. Version: November 2008. – Vortragsfolien für die Kinderuni-Veranstaltung 'Einführung in die Kryptologie' an der Universität zu Köln

[Küh04] Kühnle, Jens: *Kryptographie.* `http://www.mathematik.uni-ulm.de/sai/ss04/internet/kuehnle.pdf`. Version: Juli 2004. – Seminarausarbeitung zum Thema 'Internetdienste' im Sommersemester 2004 an der Fakultät für Mathematik und Wirtschaftswissenschaften der Universität Ulm

[KW99] Kuhn, Markus ; Whistler, Ken: *ISO/IEC 8859-15:1999 to Unicode.* `ftp://ftp.unicode.org/Public/MAPPINGS/ISO8859/8859-15.TXT`. Version: Juli 1999

[Lew00] Lewand, Robert E.: *Cryptological mathematics.* Washington, DC : Mathematical Assoc. of America, 2000. – XIV, 199 S. : Ill., graph. Darst.. – ISBN 0–88385–719–7

[LLL82] Lenstra, Arjen K. ; Lenstra, Hendrik W. Jr. ; Lovász, László: Factoring polynomials with rational coefficients. In: *Mathematische Annalen* 261 (1982), Dezember, Nr. 4, S. 656–715

[LPV05] Lovász, László ; Pelikán, Jozsef ; Vesztergombi, Katalin: *Diskrete Mathematik.* Berlin [u.a.] : Springer, 2005 (Springer-Lehrbuch). – IX, 362 S. : graph. Darst.. – ISBN 3–540–20653–1 — 978–3–540–27553–4. – Discrete mathematics - elementary and beyond <dt.>. - Elektronische Ressource

[May99a] May, Alexander: *Auf Polynomgleichungen basierende Public-Key-Kryptosysteme*, Johann Wolfgang Goethe-Universität Frankfurt am Main, Diplomarbeit, Juni 1999

[May99b] May, Alexander: *Cryptanalysis of NTRU-107.* `http://www.cits.rub.de/imperia/md/content/may/paper/cryptanalysisofntru.ps`. Version: März 1999. – Preprint

[May06] May, Alexander: *Public Key Kryptanalyse: Seitenkanalangriffe*. http://www.informatik.tu-darmstadt.de/KP/lehre/ws0506/vl/skript/seiten.pdf. Version: Januar 2006. – Teil einer im Wintersemester 2005/2006 gehaltenen Vorlesung am Fachbereich Informatik der Technischen Universität Darmstadt

[Mek03] Mekelburg, Hans-G.: *Verschlüsselte Botschaften: Kryptologische Algorithmen*. http://www.nwn.de/hgm/krypto/algo.htm. Version: Oktober 2003. – Übersicht einiger Kryptosysteme in der Notation von JavaScript

[Mic07] Microsoft Corporation: Description of Digital Certificates. In: *Microsoft Help and Support* (2007), Januar. http://support.microsoft.com/kb/195724. – Article ID: 195724 – Revision: 3.2

[MQ06] Müller-Quade, Jörn: *Hieroglyphen, Enigma, RSA – Eine Geschichte der Kryptographie*. http://iaks-www.ira.uka.de/eiss/fileadmin/User/enigma.pdf. Version: Oktober 2006. – Vortrag im Rahmen des E.I.S.S. – Kolloquiums am Institut für Algorithmen und Kognitive Systeme der Universität Karlsruhe (TH)

[MVOV97] Menezes, Alfred J. ; Van Oorschot, Paul C. ; Vanstone, Scott A.: *Handbook of applied cryptography*. Rev. reprint with updates. Boca Raton, Fla. [u.a.] : CRC Press, 1997 (CRC Press series on discrete mathematics and its applications). – XXVIII, 780 S. : Ill., graph. Darst.. – ISBN 0–8493–8523–7

[Nas92] Nassua, Martin: *'Gemeinsame Kriegführung, gemeinsamer Friedensschluss' : das Zimmermann-Telegramm vom 13. Januar 1917 und der Eintritt der USA in den 1. Weltkrieg*. Frankfurt am Main [u.a.] : Lang, 1992 ([Europäische Hochschulschriften / 03] ; 520). – IV, 163 S.. – ISBN 3–631–44752–3. – Zugl.: Berlin, Freie Univ., Magisterarb., 1990

[Ngu06] Nguyen, Phong Q.: *A Note on the Security of NTRUSign*. http://eprint.iacr.org/2006/387.pdf. Version: November 2006

[NTR07] NTRU Cryptosystems, Inc: *The NTRU Public Key Cryptosystem — A Tutorial*. http://ntru.com/downloads/Ntru_Public_Key_Cryptosystem_Tutorial.pdf. Version: September 2007

[NTR09] NTRU Cryptosystems, Inc: *Corporate Brief*. http://www.ntru.com/downloads/Ntru_Corporate_Brief.pdf. Version: April 2009

[Ove04] Overbeck, Raphael: *Potential und Grenzen der Anwendung von Gitterreduktionsalgorithmen in der Kryptographie*, Technische Universität Darmstadt, Diplomarbeit, März 2004

[Pah08] Pahl, Robert: *Jenseits von Kahn.* `http://www.tronland.net/cryptron/jvk.htm`. Version: Oktober 2008. – Eine Darstellung über die nicht-mathematischen Prinzipien zum Kodebrechen

[Pra40] Pratt, Fletcher: *Histoire de la cryptographie : Les écritures secrètes depuis l'antiquitité jusqu'à nos jours. Trad. du capitaine E. Arnaud.* Paris : Payot, 1940 ((Bibliothèque historique.) [215]). – 296 S.

[Rib88] Ribenboim, Paulo: *The book of prime number records.* New York [u.a.] : Springer, 1988. – XXIII, 476 S.. – ISBN 0–387–96573–4 — 3–540–96573–4. – Literaturverz. S. 355 - 455

[Roh79] Rohwer, Jürgen (Hrsg.): *Die Funkaufklärung und ihre Rolle im Zweiten Weltkrieg : eine internationale Tagung in Bonn-Bad Godesberg und Stuttgart vom 15.-18. Nov. 1978.* 1. Aufl. Stuttgart : Motorbuch-Verl., 1979. – 406 S. : Ill.. – ISBN 3–87943–666–5

[RSA78] Rivest, Ronald L. ; Shamir, Adi ; Adleman, Leonard: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. In: *Communications of the ACM* 21 (1978), S. 120–126

[Rüc07] Rückert, Markus: *Implementierung und Analyse von gitterbasierten Angriffen auf NTRU*, Technische Universität Darmstadt, Diplomarbeit, April 2007

[Rup01] Ruppert, Wolfgang M.: *Ausgewählte Kapitel aus der Kryptographie - Enigma.* `http://www.mathematik.uni-erlangen.de/~ruppert/SS01/akryp4.ps.gz`. Version: Juli 2001. – Teil einer im Sommersemester 2001 gehaltenen Vorlesung am Mathematischen Institut der Universität Erlangen

[Rüt09] Rütten, Christiane: Effektiver Passwörter raten mit Wikipedia und Co. In: *heise Security* (2009), März. `http://www.heise.de/newsticker/meldung/135291`. – Möglichkeiten für effiziente Wörterbuchangriffe mithilfe von freien Online-Enzyklopädien

[Sau02] Sauer, Steffen: *Die Enigma: Sicherheit.* `http://ivs.cs.uni-magdeburg.de/bs/lehre/wise0102/progb/vortraege/steffensauer/enigma_sicherheit.html`. Version: Oktober 2002. – Seminarausarbeitung zum Thema 'Geheime Botschaften' im Wintersemester 2001/2002 am Institut für Verteilte Systeme der Otto-von-Guericke-Universität Magdeburg

[Sch96] Schneier, Bruce: *Applied cryptography : protocols, algorithms, and source code in C.* 2. ed. New York [u.a.] : Wiley, 1996. – XXIII, 758 S. : graph. Darst.. – ISBN 0–471–11709–9 — 0–471–12845–7

[Sch07] Schindler, Werner (Hrsg.) ; Bonn-Aachen International Center for Information Technology (Veranst.): *Für jedes Problem das richtige Werkzeug: Stochastik bei Seitenkanalangriffen.* 7. Kryptotag, November 2007

[Sha49] Shannon, Claude E.: Communication theory of secrecy systems. In: *The Bell system technical journal* XXVIII (1949), Oktober, S. 656–715

[Sho97] Shor, Peter W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. In: *SIAM Journal on Computing* 26 Issue 5 (1997), Oktober, S. 1484–1509

[Sil98] Silverman, Joseph H.: Invertibility in Truncated Polynomial Rings. In: *CryptoLab – Technical Notes* (1998), Oktober. `http://ntru.com/cryptolab/pdf/NTRUTech009.pdf`. – Report # 009, Version 1

[Sil99] Silverman, Joseph H.: Almost Inverses and Fast NTRU Key Creation. In: *CryptoLab – Technical Notes* (1999), März. `http://ntru.com/cryptolab/pdf/NTRUTech014.pdf`. – Report # 014, Version 1

[Sin06] Singh, Simon: *Geheime Botschaften : die Kunst der Verschlüsselung von der Antike bis in die Zeiten des Internet.* 7. Aufl. München : Dt. Taschenbuch-Verl., 2006 (dtv ; 33071). – 459 S. : Ill., graph. Darst.. – ISBN 978–3–446–19873–8. – The code book <dt.>

[SSP08] Swoboda, Joachim ; Spitz, Stephan ; Pramateftakis, Michael: *Kryptographie und IT-Sicherheit : Grundlagen und Anwendungen.* 1. Aufl. Wiesbaden : Vieweg+Teubner, 2008 (IT-Sicherheit und Datenschutz). – XVIII, 263 S. : Ill., graph. Darst.. – ISBN 978–3–8348–0248–4 — 3–8348–0248–4. – Literaturverz. S. [241] - 248

[Sti95] Stinson, Douglas R.: *Cryptography : theory and practice.* 5. print. Boca Raton [u.a.] : CRC Press, 1995 (The CRC Press series on discrete mathematics and its applications). – 434 S. : graph. Darst.. – ISBN 0–8493–8521–0

[TOP08] TOP500.Org: *TOP500 List - November 2008 (1-100) — TOP500 Supercomputing Sites.* `http://www.top500.org/list/2008/11/100`. Version: November 2008. – Die Liste der 500 schnellsten Supercomputer der Welt

[VSB+01] Vandersypen, Lieven M. K. ; Steffen, Matthias ; Breyta, Gregory ; Yannoni, Costantino S. ; Sherwood, Mark H. ; Chuang, Isaac L.: Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. In: *Letters to Nature* 414 (2001), Dezember, S. 883–887

[WF06] Wilberg, Julian ; Fink, Marius: *Facharbeit 'Buchstabenprofile' : Französisches Buchstabenprofil.* `http://buchstabenprofile.mariusfink.org/doc/tabellen/detail_fr.ods`. Version: Juni 2006

[Wik09a] WIKIPEDIA: *Kryptographie*. `http://de.wikipedia.org/w/index.php?title=Kryptographie&oldid=56756819`. Version: 16. Februar 2009

[Wik09b] WIKIPEDIA: *Enigma*. `http://de.wikipedia.org/w/index.php?title=Enigma_(Maschine)&oldid=57995489`. Version: 17. März 2009

[Wik09c] WIKIPEDIA: *Kryptoanalyse*. `http://de.wikipedia.org/w/index.php?title=Kryptoanalyse&oldid=58063501`. Version: 19. März 2009

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Augsburg, den 6. August 2009 _____