technische universität
dortmund

Bachelor Thesis

K-means auf verschlüsselten Daten: FV im Praxiseinsatz

Sebastian Schröder
September 2018

Verfasser: Sebastian Schröder
Matr. Nr.: 123634
E-Mail: sebastian2.schroeder@tu-dortmund.de

Gutachter:
Prof. Dr. Katharina Morik
Dr. Thomas Liebig

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz (8)
http://www-ai.cs.uni-dortmund.de

Zusammenfassung

In dieser Bachelorarbeit mit explorativem Forschungshintergrund prüfen wir die Einsatzfähigkeit der Fan und Vercauteren (FV) Verschlüsselung [67] aus der öffentlich verfügbaren Bibliothek SEAL v2.3.0-4 von Microsoft [33]. Bei k-means Clusteranalysen an zwei unterschiedlichen Datensätzen, dem frei zugänglichen Iris Datensatz und den spatio-temporalen Verkehrsdaten aus Dublin, zeigte sich, dass mit FV verschlüsselte Daten zu 100 Prozent korrekt zugeordnet werden. Gemessene Laufzeiten von 15 Minuten bis zu über 14 Stunden weisen jedoch darauf hin, dass der Praxiseinsatz nur mit maßgeschneiderten Parametern zu korrekten Ergebnissen in akzeptablen Laufzeiten führt. Wir dokumentieren und erläutern die Herleitung dieser sich gegenseitig beeinflussenden Parameter detailliert. Damit sind unsere Ergebnisse nachvollziehbar und hinsichtlich des Iris Datensatz barrierefrei replizierbar. Des Weiteren geben wir Hinweise, in welchem Szenarium sich der Einsatz von FV heute schon lohnt und zeigen Alternativen auf. Wir erleichtern Praktikern und Forschern die herausfordernde Auswahl eines passenden homomorphen Verschlüsselungssystems für ihre zukünftigen Projekte durch unsere Erläuterungen zweier in der Literatur parallel gebräuchlichen Klassifikationsschemata [8, 7]. Damit ordnen wir das typische Begriffswirrwarr in diesem jungen Forschungsfeld. Zudem bieten wir eine Zusammenfassung des Forschungsstands der PHE, SHE, LHE und FHE Kryptosysteme. Wir stellen tabellarische Überblicke bereit, die Literaturangaben sowie erzielte Laufzeiten und die verwandte Technik vorheriger Forschung enthalten. Damit soll die eigene Einarbeitung in die Literatur erleichtert werden. Wir hoffen, mit dieser Untersuchung nicht nur die von vorheriger Forschung geforderten Daten bereitzustellen, sondern auch die Mauer zwischen Theorie und Praxis beim Einsatz kryptografischer Algorithmen als Tool der IT Sicherheit verkleinert zu haben.

Stichworte: Fan und Vercauteren (FV) Verschlüsselung, homomorphe Verschlüsselung, SEAL Bibliothek v.2.3.0-4, k-means Clusteranalyse, verschlüsselte Daten, FHE, LHE, SHE, PHE, IT Sicherheit.

Inhaltsverzeichnis

Abbildungsverzeichnis

Algorithmenverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AES	Advanced Encryption Standard
AGCD	Approximate Greatest Common Divisor
API	Application Programming Interface
APT	Advanced Persistent Threats
BGV	Brakerski, Gentry und Vaikuntanathan
BSI	Bundesamt für Sicherheit in der Informationstechnik
CCA1	Chosen Ciphertext Attack
CCA2	Adaptive Chosen ciphertext Attack
CPA	Chosen Plaintext attack
CRT	Chinese Remainder Theorem
CSPRNG	Cryptographically Secure Pseudo Random Number Generator
ct	Chiffretext
cuHE	CUDA Homomorphic Encryption Library
Dec	Decryption
DGHV	van Dijk, Gentry, Halev und Vaikuntanathan
D-H	Diffie Hellman
DLP	Discrete Logarithm Problem
Enc	Encryption
Eval	Evaluation
FE	Functional Encryption
FF	Finite Fields
FFT	Fast Fourier Transformation
FHE	Fully Homomorphic Encryption
F-NTRU	Flattening N-th degree TRUncated polynomial ring
FV	Fan und Vercauteren
GGH	Goldreich, Goldwasser und Halevi
GM	Goldwasser und Micalli
HE	Homomorphic Encryption
HEAAN	Homomorphic Encryption for Arithmetic of Approximate Numbers
HElib	Homomorphic Encryption library

IDC	International Data Corporation
IEC	Internationale Elektrotechnische Kommission
IEEE	Institute of Electrical and Electronics Engineers
IFFT	Inverse Fast Fourier Transformation
IND	Inditinguishability (Nicht Unterscheidbarkeit)
ISO	Internationale Organisation für Normung
IT	Informationstechnik
KeyGen	Key Generator
LHE	Leveled Homomorphic Encryption
LPR	Lyubashevsky, Peikert und Regev
LWE	Learning With Errors
LWR	Learning With Rounding
NC^1	Nick's Class
NFLlib	NTT-based Fast Lattice Library
NIH	National Institutes of Health
NIST	National Institute of Standards and Technology
NP	Nichtdeterministisch Polynomielle Zeit
NTRU	N-th degree TRUncated polynomial ring
P	Polynomielle Zeit
PHE	Partial Homomorphic Encryption
pk	Public-Key
PPT	Probabilistic Polynomial Time
PRNG	Pseudorandom Number Generator
QR	Quadratic Residue (quadratische Reste)
RAM	Arbeitsspeicher
RLWE	Ring Learning With Errors
RLWR	Ring Learning With Rounding
RNG	Random Number Generator
RSA	Rivest, Shamir und Adleman
SEAL	Simple Encrypted Arithmetic Library
SHE	Somewhat Homomorphic Encryption
sk	Secret-Key
SMC	Secure Multi Party Computation
td	Trapdoor
US	United States
VaVel	Variety, Veracity, VaLue
YASHE	Yet Another Homomorphic Encryption

Kapitel 1

Einleitung

"To some, it is surprising that such a thing as homomorphic encryption is possible ..."
(Craig Gentry 2009, [72, S. 6])

Stellen wir uns vor, Alice übergibt Bob einen verschlossenen Koffer voller Geld und bittet Bob, das Geld in ihrem Koffer für sie zu zählen und zu ordnen. "Mach ich. Bitte gib mir den Schlüssel," wird Bob vielleicht antworten. Doch genau das möchte Alice vermeiden. Alice möchte, dass Bob ihr ein korrektes Ergebnis liefert, aber er selbst nichts über den genauen Inhalt ihres Koffers erfährt, das Ergebnis seiner eigenen Berechnungen nicht kennt und sogar nichts über die Methoden seiner Berechnungen weiß. Nur Alice selbst will in Besitz all dieser Informationen sein.

„Das geht nicht! Das ist unmöglich", denken wir vielleicht spontan. So sehr das Szenarium auch an Zauberei erinnern mag, es ist eine dringende Aufgabe der Informatik, idealerweise genau solche Techniken (weiter-) zu entwickeln, um unsere privaten Daten bei der Speicherung und Auslagerung ihrer Berechnungen mit Methoden der künstlichen Intelligenz zu schützen[1]. Solche Forschungsprojekte werden vom Bundesministerium für Bildung und Forschung und führenden Experten der Wissenschaft unterstützt[2].

Unter privaten Daten verstehen wir in dieser Arbeit nicht die von uns freiwillig ins Internet gestellten Posts, Fotos oder Videos. Es geht stattdessen zum einen um solche Daten, deren Sammlung wir nicht verhindern können, die aber aufgrund unseres Selbstbestimmungsrechtes einen strengen Datenschutz erfordern. Ein typisches Beispiel sind unsere Gesundheitsdaten, die über das Gesundheitssystem gesammelt werden. Zum anderen sind besonders solche Daten von uns gemeint, deren Sammlung uns wenig oder gar nicht bewusst ist, die aber leicht gegen uns verwendet werden können, wenn sie in falsche Hände geraten. Ein Beispiel hierfür sind unsere über Sensoren gesammelten raumzeitlichen (engl.: spatio-temporal) Daten. Aus Daten, ob oder wann und wie lange wir uns an bestimmten

[1] `https://www.plattform-lernende-systeme.de/it-sicherheit.html`
[2] `https://www.bildung-forschung.digital/de/plattform-lernende-systeme-2270.html`

Orten aufhalten, ist es nämlich leicht, Informationen über uns zu erlangen, die wir freiwillig nicht preisgeben wollen. Erschreckender Weise wird in einem aktuellen International Data Corporation[3] (IDC) Whitepaper prognostiziert, dass der Anteil solcher ungeschützter privater Daten an der Datenflut, getriggert durch das Internet of things, bis zum Jahr 2025 auf 47 Prozent steigen soll [136].

1.1 Verwandte Arbeiten

In diesem Forschungsfeld können Randomisierungstechniken [107, 96] von kryptografischen Algorithmen [60, 92] abgegrenzt werden. Bei dem vom Agarwal und Srikant [3] im Jahr 2000 vorgeschlagenen Randomisierungsschema wird dem sensitiven Wert eines Attributs eine Zufallszahl hinzufügt. Die Autoren zeigen wie es möglich ist, die Verteilung der Originaldaten aus dem Zufallsrauschen zu rekonstruieren. Diese Technik wurde im Laufe der Jahre verbessert. Eine auf spektraler Filtertechnik basierende Zufallsmatrix [2] und weitere Modifikationen [123, 86] führten dazu, dass Randomisierungstechniken heute zwar einfach und effizient implementiert werden können, doch während des Transformationsprozesses gehen Informationen verloren [150]. Zudem fehlt die Möglichkeit, ihre Sicherheit zu formalisieren [133]. Weiterere Nachteile sind der hohe Rechen- und Kommunikationsaufwand [133]. Bei Anwendungen wie etwa in der wissenschaftlichen Forschung, wo aus ethischen und rechtlichen Gründen der Datenschutz, die Sicherheit und genaue Ergebnisse gleichermaßen hohe Priorität besitzen, kommen solche Methoden eher nicht in Frage.

Bei den kryptografischen Ansätzen können Secure-Multi-Party Computation (SMC) und homomorphe Verschlüsselungen (engl.: Homomorphic Encryption, HE) unterschieden werden. SMC wurde 1982 von Yao [163] eingeführt und erlaubt verschiedenen Parteien, gemeinsam an einer bestimmten Funktion ihrer privaten Daten zu arbeiten, wobei der Datenschutz jeder Partei gesichert ist. Secret Sharing [60] und Oblivious Transfer [155, 92] sind zwei bekannte Protokolle des SMC. Sie gehen dabei von einem sogenannten halb ehrlichen Modell aus [131], d.h., ein gewisses Maß an Unsicherheit verbleibt. SMC sind komplex in der Anwendung und mit einem relativ hohen Kooperations- und Kommunikationsaufwand verbunden [131].

HE haben dagegen ein besseres Potential, die dringlichen Probleme der IT Sicherheit beim Data Mining zu lösen. Bei solch einem Verfahren muss kein Vertrauensverhältnis zum Server bestehen, da die Daten durchgängig unlesbar für Nichtbefugte sind. Selbst wenn die Daten auf einem unsicheren Übertragungsweg abgefangen werden, könnten sie nicht (leicht) gelesen werden. Kurz: sie sind für jeden unbrauchbar, außer für die autorisierten Schlüsselbesitzer. Zudem ist die Sicherheit der HE Verfahren formalisierbar [148]. HE Systeme lassen sich je nach Art und Anzahl der Rechenoperationen unterscheiden, die sie zulassen. Solche, die entweder "nur" die Addition oder "nur" die Multiplikation unterstützen, sind

[3] `https://idc.de/de/`

aufgrund der mathematischen Beschränkungen für maschinelle Lernalgorithmen bei der Berechnung großer Datenmengen wie etwa im Forschungsbereich eher unpraktisch, da sie die Nichtlinearität, die in nützlichen Modellen vorhanden ist, nicht erlauben [4]. Allerdings sind sie praxiserprobt. Sie werden heute zum Beispiel genutzt, um auf mobilen Geräten spatio-temorale Daten unter Wahrung der Privatsphäre zu berechnen. Beispielsweise konstruierten Hallgren, Ochoa und Sabelfeld ein HE Protokoll zum Schutz der Privatsphäre, das sie InnerCircle [89] nannten. Man kann Freunde in der Nähe finden, ohne die eigenen Ortsdaten preiszugeben [165] oder den optimalen Treffpunkt für ein Gruppentreffen zu bestimmen [15]. Bei Forschungsprojekten mit großen Datenmengen wird HE in Kombination mit anderen kryptografischen Methoden dazu genutzt, dass das neu erworbene Wissen bei der Datenübertragung (engl.: data-in-motion) oder bei der Speicherung (engl.: data-at-rest) vor Verfälschung, Diebstahl oder Missbrauch geschützt wird. Die HE haben nämlich im Vergleich zu anderen Verfahren keine weiteren Nachteile außer den für alle probabilistischen Verfahren typischen längeren Laufzeiten. Das zeigte eine diesjährige Überblicksarbeit von Liu und Kollegen [109] zu den Gefahren und Abwehrtechniken beim Einsatz von maschinellen Lernalgorithmen. Interessierten stellen die Forscher der technischen Universität Dortmund einen stets aktualisierten Überblick zu solchen praxiserprobten Techniken und anderen bewährten Datenschutzmethoden bei Forschungsprojekten bereit [105].

Sogenannte nivelliert homomorphe Verschlüssellungsalgorithmen (engl.: Leveled Homomorphic Encryption, LHE) dagegen, die fast sämtliche Berechnungen auf verschlüsselten Daten zulassen, wird das größte Potential zugesprochen, zu einem wichtigen Baustein der IT Sicherheit während des Dataminings von Big Data durch eine fremde Instanz zu werden [10]. Solche Algorithmen zeichnen sich durch eine vom Anwender festgelegten Schaltungstiefe und damit einhergehender akzeptabler Laufzeit und Praxistauglichkeit aus.

Seitdem 2011 Naehrig, Lauter und Vaikuntanathan [121] die Frage stellten: "Can homomorphic encryption be practical?" hat sich diese Algorithmenklasse, LHE, rasant weiterentwickelt. Forscher [85] konzentrierten sich in der Anfangszeit dieser neuen Forschungslinie zunächst darauf, Algorithmen zu finden, bei denen das Training über verschlüsselte Daten durchgeführt werden kann. Sie waren gezwungen, solche Lernalgorithmen zu verwenden, bei denen der Trainingsalgorithmus als ein Polynom niedrigen Grades ausgedrückt werden kann. Infolgedessen waren die meisten früh vorgeschlagenen Algorithmen vom linearen Diskriminierungstyp. 2012 belegten erstmals Wu und Haven [160], dass die Berechnung von Kovarianzen und multiplen linearen Regressionen an über vier Millionen Daten in akzeptabler Laufzeit möglich ist. K-means Clusteranalysen [57, 164, 108] und andere maschinelle Lernmodelle [101, 40, 63] wurden auf mit LHE verschlüsselten großen Datenmengen bereits erfolgreich gezeigt. Sowohl kategoriale als auch ordinale und numerische Daten sind dafür geeignet [112]. Die noch vorhandenen praktischen Einschränkungen der heute verfügbaren Algorithmen erfordern lediglich maßgeschneiderte Ansätze, postulieren Aslett und Kollegen. Sie zeigen, wie die vorhandenen Lernalgorithmen umgeschrieben werden können, um

mit den modernen Algorithmen kompatibel zu sein [10]. Bei Bayes Klassifikatoren erwiesen sich die Leistungskosten in der Arbeit von Aslett und Kollegen [9] im Vergleich mit unverschlüsselten Daten sogar als wettbewerbsfähig. Die Forscher zeigten auch, dass sie ohne Mehrparteienberechnung oder Kommunikation auskommen können. LHE Systeme scheinen also heute schon arbeitserleichternd und komfortabel in verschiedenen Situationen einsetzbar zu sein [9, 10]. Weniger optimistisch benennen Wiese und Kollegen dagegen in ihrer aktuellen Anforderungsanalyse [159] zum Einsatz von LHE Algorithmen die Schwachpunkte dieser Verfahrensklasse. Beispielsweise ist die Wahl von geeigneten Parametern problematisch, mit denen die Verfahren sicher genug aber gleichzeitig noch effizient in der Praxis sind. Die Auswahl des passenden Algorithmus wird zudem durch fehlende Dokumentationen der Laufzeiten und dem Fehlen von Vergleichsarbeiten erschwert. Boxen und Kollegen [20] konnten 2016 nur sechs von 17 geplanten Implementierungen mithilfe ihres Testszenarios zur Lauffähigkeit bringen. Leider konnte ausgerechnet das als das sicherste und flexibel geltende Fan und Vercauteren (FV) LHE Schema [159, 67] aus der Simple Encrypted Arithmetic Library (kurz SEAL) nicht in der damaligen Version 2.0 beta implementiert werden. Die Autoren beider Arbeiten beklagen übereinstimmend, dass die Implementation eines LHE Systems heute noch explorativen Charakter hat. Erschwert wird potentiellen Anwendern die Auseinandersetzung mit den modernen LHEs auch deshalb, weil diese in Lehrbüchern und Enzyklopädien noch nicht und wenn, dann sehr stiefmütterlich behandelt werden. Etliche Reviews wenden sich an Experten oder Mathematiker [156, 75, 114], decken "nur" die Theorie der Systeme ab [132] oder fokussieren "nur" die Sicherheit, wobei eher weitreichende Grundkenntnisse voraus gesetzt werden [5]. Andere sind "nur" auf Spezialanwendungen wie beispielsweise das Cloudcomputing [5, 91] bezogen. Die Suche nach Informationen wird Interessierten durch eine uneinheitliche, verwirrende Terminologie erschwert [7, 8, 31] und ihre Implementierung wird nicht detailliert gezeigt.

Noch stellen sich also erwiesener Maßen den potentiellen Anwendern dieser vielversprechenden Algorithmenklasse einige Hindernisse bei der Implementierung in den Weg, die nur von ausgewiesenen Kryptologie Experten gemeistert werden können. Damit sich dies zukünftig verbessert, haben sich bereits in diesem Jahr zum zweiten Mal internationale Kryptologie Experten mit Vertretern von Universitäten, Wirtschaftspartnern und den US Behörden NIST und NIH zusammen gesetzt[4]. Sie wollen zusammen vorhandene Bibliotheken interoperabel gestalten und den potentiellen Nutzern Standardisierungen und APIs an die Hand geben. Erste Whitepaper sind frei zugänglich erschienen [31, 7, 26].

[4] `https://www.microsoft.com/en-us/research/blog/second-homomorphic-encryption-standardization-workshop-delivers-goods/`

1.2 Ziele der Untersuchung

Das Hauptanliegen dieser Bachelorarbeit ist es, die Einsatzfähigkeit des Fan Vercauteren (FV) Verschlüsselungssystems [67] in der Version v2.3.0-4 der SEAL Bibliothek [33] am Beispiel einer k-means Clusteranalyse zu dokumentieren. Damit soll einem potentiellem Anwender nicht nur der Einstieg in die moderne Kryptologie erleichtert werden, sondern gleichzeitig auch die von vorheriger Forschung geforderten Praxisdaten [159] hinsichtlich der Laufzeit und Parameterwahl bereitgestellt werden. Auch wenn die Implementierung auf Grundlage der SEAL Bibliothek damals nicht gelungen ist [20], glauben wir an eine Verbesserung in der aktuellen Version. Wir nehmen an, dass die SEAL Bibltiothek aufgrund der federführenden Rolle von Microsoft bei den aktuellen Standardisierungsbemühungen internationaler führender Kryptologieexperten zukünftig an Bedeutung gewinnen wird. An der Arbeit von Du, Gustafson, Huang und Peterson [63] orientieren sich dabei unsere Umsetzung des k-means Algorithmus, des Divisionsprotokolls und des Kommunikationsprotokolls.

Im Unterschied zu diesen Autoren haben wir eine andere Zielsetzung. Sie untersuchten den Einfluss der Eingabemenge und der Anzahl der Cluster auf die Laufzeit bei einer einzelnen FV Konfiguration. Wir dagegen betrachten den Einfluss verschiedener FV Parameter auf die Laufzeit. Dabei werden die Auswirkungen auf die Ergebnisgenauigkeit und die konkreten Laufzeiten dokumentiert. Damit kommen wir den Forderungen der Forschungscommunity nach konkreten Laufzeiten als Auswahlhilfe eines LHE Schemas nach [159]. Zudem geben wir Hinweise zur Parameterwahl und sichern durch Verwendung eines öffentlich verwendeten Datensatzes die Nachvollziehbarkeit.

Als Beispieldatensätze werden zum einen der für Klassifikationstests sehr beliebte und öffentlich zugängliche Datensatz der Schwertlilien verwandt. Zum anderen werden aus einem Forschungsprojekt [106] bereitgestellte, spatio-temporale Daten aus Dublin untersucht. Auf beiden Datenstätzen wird eine k-means Clusteranalyse durchgeführt. Damit soll die Nachvollziehbarkeit und Vergleichbarkeit sichergestellt werden. Zudem ist die Arbeit in der Forschungstradition des VaVel Projekts [105] der technischen Universität Dortmund angesiedelt. Es wird ein Client-Server Modell simuliert. Folgende übergeordneten explorative Forschungsfragen sollen mithilfe der Experimente beantwortet werden:

1. Wie aufwendig ist die Parameterbestimmung?
2. Wie hoch sind die Laufzeitkosten beim Einsatz einer FV Verschlüsselung?
3. Wie lässt sich die Ergebnisgenauigkeit einer k-means Clusternalyse auf FV verschlüsselten Daten beschreiben?
4. Unterscheiden sich die Ergebnisse von vorheriger Forschung und falls ja, wie?
5. Welche Vorschläge können für zukünftige Forschung gemacht werden?

1.3 Aufbau der Arbeit

Um die definierten Ziele zu erreichen, gliedert sich diese Untersuchung in zwei Teile. Der erste, theoretische Teil umfasst die Kapitel zwei bis fünf. Hier werden die wichtigsten Grundlagen beschrieben. Dabei liegt der Schwerpunkt auf der Kryptologie, damit die Experimente, die im zweiten praktischen Teil beschrieben sind, leicht nachvollzogen werden können. Nachdem in Kapitel zwei die kryptologischen Grundlagen auch für Laien verständlich erläutert wurden, befasst sich Kapitel drei mit dem Forschungsstand der modernen LHE Algorithmen. Es werden dabei tabellarische Überblicke bereitgestellt. Dem potentiellen Anwender wird so die Auswahl eines passenden Algorithmus erleichtert. Es werden allgemeingültige Hinweise zur Implementierung eines LHE Systems gegeben. Im vierten Kapitel wird das FV Verschlüsselungsschema mit seinen Parametern detailliert vorgestellt. Das fünfte Kapitel ist dem k-means Clusteralgorithmus als Beispiel eines maschinellen Lernalgorithus gewidmet. Mit dem sechsten Kapitel beginnt der praktische Teil dieser Untersuchung. Hier werden die verwendeten Arbeitsmittel, Daten und Methoden erläutert. Im siebten Kapitel werden die Ergebnisse präsentiert. Schließlich werden im achten Kapitel die Ergebnisse zusammengefasst, die Forschungsfragen beantwortet und diskutiert. Dabei wird auf Einschränkungen hingewiesen und es werden Vorschläge für zukünftige Forschung gemacht.

Kapitel 2

Grundlagen der Kryptologie

In diesem Kapitel wollen wir die Grundlagen der Kryptologie, der Wissenschaft der Verschlüsselung, erläutern. Die moderne Kryptologie basiert auf Erkenntnissen der Komplexitätstheorie, der Wahrscheinlichkeitstheorie und der Zahlentheorie, um nur einige zu nennen. Im beschränktem Rahmen dieser Bachelorarbeit müssen wir davon ausgehen, dass diese Voraussetzungen weitgehend bekannt sind. Wir werden daher nur die wichtigsten Zusammenhänge zum Verständnis und der Einordnung unseres Beispielalgorithmus, des Fan und Vercauteren (FV) Algorithmus, klären. Zur Auffrischung werden wir grundlegende Definitionen und Beispiele wiederholen. Interessierte finden ergänzende Informationen in den sorgsam ausgesuchten Quellen. Wir werden im Folgenden Fragen beantwortet wie:

- Was ist moderne Verschlüsselung?
- Wie lassen sich kryptografische Systeme unterscheiden?
- Welche funktionalen Qualitätseigenschaften besitzen Kryptosysteme?
- Warum werden kryptografische Algorithmen nicht geheim gehalten?
- Wie lässt sich die Sicherheit der Verfahren bestimmen?

2.1 Grundschema einer Verschlüsselung

Soll eine Nachricht m (engl.: message) über einen unsicheren Übertragungskanal von „A nach B“ übertragen werden, sprechen wir nach den Konventionen von A(lice) als Sender und Initiator des Prozesses und B(ob) dem Empfänger der Nachricht und ihrem Kommunikationspartner. Dabei spielt es keine Rolle, ob mit Alice und Bob reale Personen oder Systeme wie z. B. eine Cloud, ein einzelner Server, etc. gemeint sind. Die zu übermittelnde Nachricht m kann ebenfalls „alles“ sein, was sich als Zahl darstellen lässt. Früher waren es oft Briefe, heute sind meist Text-, Sprach-, Bild-, Filmdateien, etc. in Bits beliebiger Länge gemeint. Der ”unsichere Kommunikationskanal” ist heute z. B. das Internet oder die Wlan

Verbindung. Das Grundschema einer Verschlüsselung deckt jedes denkbare Szenarium ab, wenn es darum geht, eine Nachricht m mit mindestens einem Schlüssel k (engl.: key) zu verschlüsseln. Nur die berechtigten Instanzen besitzen den oder die Schlüssel, mit dem sie die verschlüsselte Nachricht, den Chiffretext (engl.: ciphertext, ct) wieder in die lesbare Form, den Klartext (engl.: plaintext, p), bringen können. Abbildung 2.1 zeigt das Grundschema einer symmetrischen Verschlüsselung zwischen Alice und Bob. Später in Kapitel 2.4.2 klären wir, was eine Symmetrische Verschlüsselung ist.

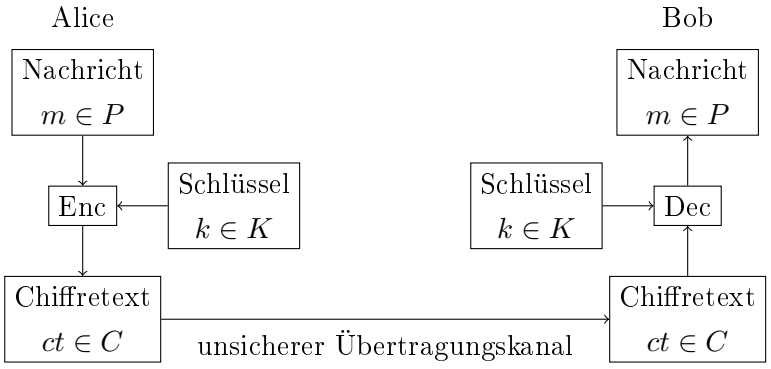


Abbildung 2.1: Grundschema einer symmetrischen Verschlüsselung bei der Kommunikation zwischen Alice und Bob. Enc: Verschlüsselung, Dec: Entschlüsselung.

2.1.1 Definition (Alphabet). Für die folgende Definition von kryptografischen Systemen benötigen wir Alphabete. Ein Alphabet A ist eine nichtleere Menge, deren Elemente w_i als Buchstaben bezeichnet werden. Aus dem Alphabet lassen sich Wörter w bilden, sodass gilt:

$$w = (w_1, \ldots, w_n) \in A^n \ .$$

Wir definieren:

$$A^* = \bigcup_{n \in \mathbb{N}_0} A^n \ .$$

A^* beinhaltet alle Wörter beliebiger Länge, die sich aus A bilden lassen.

2.1.2 Definition (kryptografisches System). Mathematisch betrachtet ist ein kryptografisches System ein Tupel aus fünf Elementen $(P \subset A_1^*, C \subset A_2^*, K, Enc, Dec)$:

- A_1 ist das Klartextalphabet und A_2 das Chiffretextalphabet.
- P ist der Klartextraum und C der Chiffretextraum. Sie sind die Menge aller möglichen Klartexte (engl.: plaintext), die das Kryptosystem verschlüsseln kann bzw. die Menge aller möglichen Chiffretexte (engl.: ciphertext), die das Ergebnis der Verschlüsselung sind.
- K ist die Menge aller möglichen Schlüssel (engl.: key) und heißt Schlüsselraum.

- $Enc = \{Enc_k : k \in K\}$ ist eine Familie von Funktionen $Enc_k : P \to C$, den Verschlüsselungsfunktionen (engl.: encryption).
- $Dec = \{Dec_k : k \in K\}$ ist eine Familie von Funktionen $Dec_k : C \to P$, den Entschlüsselungsfunktionen (engl.: decryption).

Es wird häufig $A := A_1 = A_2$ und $A^* := P := C$ gewählt. Beispielweise gilt $A = \{0, 1\}$ für die Binärschreibweise bei der Nutzung eines Computers. Das kryptografische System muss die Eigenschaft der Korrektheit (engl.: correctness) erfüllen. Diese besagt, dass es für eine Nachricht $m \in P$ und jeden Verschlüsselungsschlüssel $k \in K$ (engl.: encryption key) einen Entschlüsselungsschlüssel $k' \in K$ (engl.: decryption key) gibt, sodass für alle verschlüsselbaren Nachrichten die Gleichung $Dec_{k'}(Enc_k(m)) = m$ erfüllt ist [125].

2.2 Algebraische und komplexitätstheoretische Grundlagen

Die algebraische Struktur des Klartextraums P sowie des Chiffretextraums C liegen bei modernen Kryptosystemen oft in einer Gruppe oder in einem Ring. D.h., es gibt Verknüpfungen, sodass $(A^*, \circ)$ eine Gruppe oder $(A^*, +, \cdot)$ ein Ring ist.

2.2.1 Definition (Gruppe). Eine Gruppe ist eine Menge G mit einer Verknüpfung $\circ$, sodass die folgenden drei Bedingungen gelten:

- $\circ$ ist assoziativ (d.h., $(G, \circ)$ ist eine Halbgruppe).
- Es gibt ein neutrales Element $e \in G$.
- Für jedes Element aus G gibt es ein inverses Element.

Wenn ihre Verknüpfung kommutativ ist, wird eine Gruppe abelsche Gruppe genannt.

2.2.2 Definition (Ring). Ein Ring $(R, +, \cdot)$ ist eine Menge R mit zwei Verknüpfungen Addition $(+)$ und Multiplikation $(\cdot)$, sodass folgende Bedingungen erfüllt sind:

- $(R, +)$ ist eine abelsche Gruppe.
- $(R, \cdot)$ ist eine Halbgruppe.
- $(R, +, \cdot)$ ist assoziativ und es gibt ein neutrales Element der Multiplikation $e \in R$.
- Es gelten die Distributivgesetze.

Ist ein Ring kommutativ, so wird R ein kommutativer Ring genannt.

Moderne Kryptosysteme bestehen aus mathematischen Ver- und Entschlüsselungsfunktionen, die oft miteinander "verwandt" sind, d.h., die eine Funktion ist die Umkehrfunktion der anderen. Neben diesen Algorithmen besteht jedes Kryptosystem, wie Abbildung 2.1

visualisiert, noch aus dem Klartext, den Chiffretexten und mindestens einem Schlüssel. Wir benötigen, wie wir später noch genauer sehen werden, meist mehrere effiziente Algorithmen.

Ein Algorithmus wird als effizient bezeichnet, wenn der Ressourcenverbrauch (Zeit- und Speicherverbrauch) möglichst gering ist. Die Effizienz kryptografischer Verfahren wird in der $\mathcal{O}$-Notation angegeben, die es erlaubt, Algorithmen auf einer höheren Abstraktionsebene zu vergleichen, unabhängig von Implementierungsdetails, wie Programmiersprache, Compiler und Hardware Eigenschaften. $\mathcal{O}$ bezeichnet die obere Laufzeitschranke (höchstens), Ω die untere Laufzeitschranke (mindestens) und Θ die genaue Laufzeitschranke.

2.2.3 Definition ($\mathcal{O}$-Notation). Seien $f, g : \mathbb{N}_0^r \to \mathbb{R}$ mit $f(n), g(n) > 0$ zwei Funktionen, die die Laufzeit in Abhängigkeit der Eingabegröße n beschreiben, dann gilt:

- $\mathcal{O}(f(n)) = \{g(n) : \exists c > 0, n_0 > 0$, sodass für alle $n \geq n_0$ gilt $g(n) \leq c \cdot f(n)\}$.

Wir sagen, dass $g(n)$ durch $f(n)$ beschränkt ist und schreiben $g(n) \in \mathcal{O}(f(n))$. Dies bedeutet, dass $g(n)$ für $n \to \infty$ höchstens genauso stark wächst wie $f(n)$. Rechnerabhängige Konstanten werden hierbei ignoriert. Betrachtet wird das Laufzeitverhalten mit der jeweils zeitaufwändigsten Eingabe (dem Worst-Case). D.h., wir geben mithilfe der $\mathcal{O}$-Notation eine maximale Obergrenze der Laufzeit in Abhängigkeit der Eingabe n an.

Ist die Laufzeit das Kriterium, können wir also, wenn c eine Konstante ist, folgendes Laufzeitverhalten beschreiben:

- $\mathcal{O}(c)$: Die Laufzeit unseres Algorithmus ist konstant, d.h., unabhängig von der Eingabelänge.
- $\mathcal{O}(n)$: Die Laufzeit unseres Algorithmus ist linear.
- $\mathcal{O}(n^c)$: Ist $c \geq 2$, dann ist die Laufzeit unseres Algorithmus polynomiell.
- $\mathcal{O}(c^n)$: Ist $c \geq 2$, dann ist die Laufzeit unseres Algorithmus exponentiell.

Es gilt dabei die Hierarchie für ein $0 < \epsilon \leq \frac{1}{2}$ und ein $c \geq 2$:

$$\mathcal{O}(\log n) \subseteq \mathcal{O}(\log^2 n) \subseteq \mathcal{O}(\log^c n) \subseteq \mathcal{O}(n^\epsilon) \subseteq \mathcal{O}(\sqrt{n}) \subseteq \mathcal{O}(n) \subseteq \mathcal{O}(n^2) \subseteq \mathcal{O}(n^c) \subseteq \mathcal{O}(c^n) \ .$$

Algorithmen mit exponentieller Laufzeit werden als sehr ineffizient angesehen. Die Polynomialzeit wird in der Komplexitätstheorie als Grenze zwischen effektiv lösbaren und nicht effektiv lösbaren Problemen erachtet.

2.2.4 Definition (Polynomialzeit). Ein mathematisches Problem wird in polynomieller Zeit lösbar genannt, wenn es von einem Algorithmus gelöst wird, dessen benötigte Rechenzeit m höchstens polynomiell mit der Größe der Eingabe n des Problems wächst, d.h., es gibt eine natürliche konstante Zahl $k \geq 1$ mit $m(n) \in \mathcal{O}(n^k)$.

Kryptologische Verfahren rechnen meist mit sehr großen Zahlen. Wenn die Addition von zwei Bits die Zeit $\mathcal{O}(1)$ braucht, dann wird für zwei Zahlen $a, b \in \mathbb{Z}$ die folgende Laufzeit benötigt:

- Addition und Subtraktion: $\mathcal{O}(\max\{\log a, \log b\})$.
- Multiplikation: $\mathcal{O}((\log a) \cdot (\log b))$.
- Division mit Rest: $\mathcal{O}((\log b) \cdot (\log \frac{a}{b}))$.

$\log a$ ist hierbei der Logarithmus der Zahl a zur Basis 2 und gibt die ungefähre Bitlänge an. Es gibt Verfahren, die die Laufzeit verkürzen können. Werden bei Algorithmen Restklassen modulo n durch ihre kleinsten nicht negativen Vertreter der ganzen Zahlen $\mathbb{Z}_n$ repräsentiert (d.h., $a, b \in \mathbb{Z}_n$), dann werden nach [147] für die Rechenoperationen im Restklassenring die folgenden Laufzeiten benötigt:

- Addition und Subtraktion: $(\log n)$.
- Multiplikation und Division: $\mathcal{O}(\log^2 n)$.
- Exponentation ($a^b \mod n$): $\mathcal{O}((\log b)(\log^2 n))$.

2.2.1 Problemklassen

Die Algorithmen werden nach Erkenntnissen der Komplexitätstheorie entwickelt, die mathematische Probleme, also Aufgaben, die ein Computer (nicht) effizient lösen kann, in Komplexitätsklassen einteilt. Als Hilfsmodell wird dabei die theoretische Turingmaschine verwandt, die deterministisch, nichtdeterministisch und probabilistisch arbeiten kann. Für die Kryptologie sind die folgenden Komplexitätsklassen von Bedeutung:

1. **P Probleme** sind in polynomineller Zeit lösbar, z. B. Primzahltests.
2. **NP Probleme** können nur mit einer nichtdeterministischen Turingmaschine in polynomieller Zeit gelöst werden. Hierzu wird das Entscheidungsproblem positiv beantwortet, wenn eine Instanz der möglichen Berechnungen einer nichtdeterministischen Turing Maschine positiv antwortet. Anders ausgedrückt lässt sich hier nur in polynomieller Zeit testen, ob ein gegeber Lösungskandidat eine Lösung ist, z. B. Primfaktorzerlegung.
3. **NP schwere Probleme** sind mindestens so "schwer" zu lösen, wie alle Probleme in NP. D.h., jedes Problem in NP lässt sich auf ein NP schweres Problem polynomiell reduzieren. So ist die Lösung eines NP schweren Problems auch die Lösung jedes Problems, das in NP liegt.

4. **NP vollständige Probleme** sind alle Probleme, die sowohl NP schwierig sind, als auch selbst in NP liegen. Diese Klasse bildet so die schwersten Probleme aus NP.

Dabei ist zu beachten, dass alle Probleme aus P auch in NP liegen. Bei der Kryptoanalyse mit der Turingmaschine ist es nicht möglich, Verschlüsselungsalgorithmen schwieriger als NP zu testen. Bei der Exhaustion, dem Durchprobieren aller Schlüssel mit jeweiliger Probeverschlüsselung bei bekanntem Klartext, muss die Verschlüsselungsfunktion effizient sein und damit also in P liegen.

Kryptoanalytiker nutzen daher meist Orakel. Das sind simulierte Algorithmen, deren Funktionsweise unbekannt ist und die gar nicht existieren müssen, beispielsweise ein Orakel, das für jede Zahl $m \in \mathbb{Z}$ ihre Primteiler liefert. Hierbei ist die Funktionsweise uninteressant, es zählt nur das Ergebnis. Bei Orakeln gibt man oft eine Wahrscheinlichkeit an, mit der sie das richtige Ergebnis liefern. Für die Bestimmung eines bestimmten Sicherheitsniveaus eines Kryptoschemas nutzen Kryptoanalytiker Modelle, bei denen verschiedene Angreifertypen mit unterschiedlichen Ressourcen eine Rolle spielen. Darauf werden wir später genauer eingehen.

Bemerkung: Wie Gentry in [75] betont, ist die Frage, ob schnelle Prüfbarkeit der Lösung nicht immer eine schnelle Lösbarkeit des Problems bedeutet, auch wenn das entsprechende Lösungsverfahren, der Algorithmus, bis heute (noch) nicht gefunden wurde, wird in Kurzform als das $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ Problem bezeichnet. Es gehört zu den Millennium Problemen, für deren Lösung das Clay Mathematic Institute im Jahr 2000 eine Million Dollar ausgesetzt hat. Würde $\mathrm{P} = \mathrm{NP}$ gelten, dann wären beide Klassen gleich, was für zahlreiche Verschlüsselungsverfahren, die auf einem NP Problem basieren, bedeuten könnte, dass sie leicht geknackt werden können. Abbildung 2.2 visualisiert die Konsequenzen.
Sollte $\mathrm{P} \neq \mathrm{NP}$ gelten, dann stehen die Komplexitätsklassen in einer Beziehung zueinander, wie sie links skizziert ist. Sollte dagegen $\mathrm{P} = \mathrm{NP}$ sein, so wären alle Funktionen in $\mathrm{P} = \mathrm{NP}$ auch NP vollständig, wie rechts visualisiert.

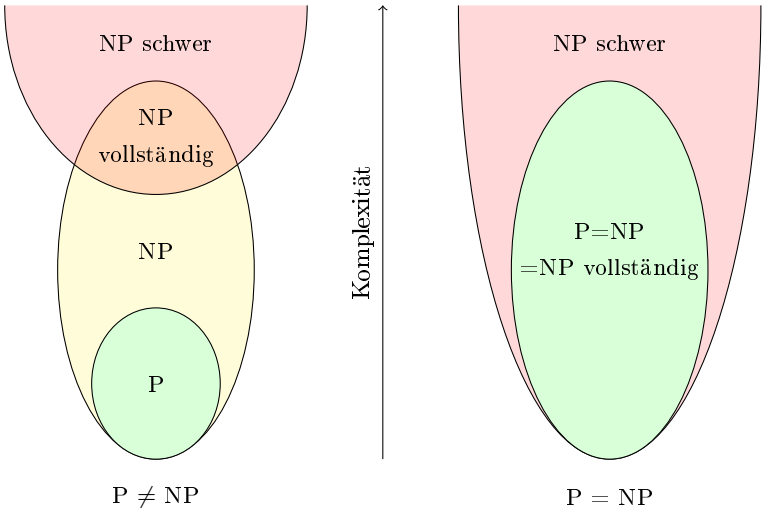


Abbildung 2.2: Problem P = NP. Quelle: eigene Darstellung nach [158].

2.2.2 Einwegfunktionen

Einwegfunktionen spielen eine zentrale Rolle bei der Konstruktion von Verschlüsselungen. Dabei handelt es sich um Funktionen, die leicht durchzuführen, aber sehr schwer (NP und höher) umzukehren sind.

2.2.5 Definition (Einwegfunktion). Eine Funktion $f : \{0,1\}^* \to \{0,1\}^*$ heißt Einwegfunktion, falls gilt:

1. Sie ist leicht zu berechnen:
   Es existiert ein deterministischer polynomialer Algorithmus, der bei Eingabe $x \in \{0,1\}^*$ den Wert $f(x)$ ausgibt.

2. Sie ist schwer zu invertieren:
   Für jeden probabilistischen polynomialen Algorithmus A, der die Umkehrfunktion von $f(x)$ berechnen soll, und jedes positive Polynom $P \in \mathbb{Z}[x]$ gilt für alle genügend großen n:

$$\text{(stark) } p(A(f(U_n), 1^n) \in f^{-1}(f(U_n))) < \frac{1}{P(n)} ,$$
$$\text{(schwach) } p(A(f(U_n), 1^n) \notin f^{-1}(f(U_n))) > \frac{1}{P(n)} .$$

   Wobei 1^n einen String aus n Einsen bezeichnet und eine Zufallszahl U_n, die gleichverteilt über $\{0,1\}^n$ ist. Bei starken Einwegfunktionen wird gefordert, dass jeder effiziente Algorithmus bei der Invertierung vernachlässigbaren Erfolg hat, wogegen bei der schwachen Einwegfunktion nur gefordert wird, dass er mit einer signifikanten Wahrscheinlichkeit fehlschlägt.

2.2.6 Definition (Vernachlässigbarkeit). Eine Funktion $f : \mathbb{N} \to \mathbb{R}$ ist vernachlässigbar (engl.: neglible) in k, wenn gilt:

$$\forall c \in \mathbb{N}_0 \exists k_0 \in \mathbb{N} \forall k \geq k_0 : |f(k)| \leq \frac{1}{k^c} \ .$$

D.h., dass eine Funktion $f(k)$ für jede beliebige Konstante c ab einem gewissen Grenzwert k_0 in ihrer Größe beschränkt ist durch $\frac{1}{k^c}$ und damit vernachlässigbar klein ist.

Um effizient ver- und entschlüsseln zu können, kommen oft Einwegfalltürfunktion (engl.: trapdoor one-way function oder kurz Trapdoor Funktionen) zum Einsatz. Solche Verfahren werden threshold Verfahren genannt. Dabei werden Einwegfunktionen mit einer geheim zu haltenden Zusatzinformation ausgestattet. Dieser Wert td (engl.: trapdoor) ermöglicht es dem berechtigten Empfänger der Nachricht, diese zu entschlüsseln, während es ohne diese Hintertür (die wörtliche Übersetzung von trapdoor ist Falltür) schwer ist, die Nachricht zu entschlüsseln. D.h., es gilt: $x \to f(x)$ leicht zu berechnen; $f(x) \to x$ schwer zu berechnen; $f_{td}(x) \to x$ leicht zu berechnen.

2.2.3 Kandidaten für Einwegfunktionen

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) [29, S. 21] empfiehlt für die praxiserprobten Verschlüsselungsverfahren in seinen Richtlinien als gleichwertige Kandidaten für Einwegfunktionen bei homomorphen Algorithmen das Faktorisierungsproblem und das Problem der Berechnung diskreter Logarithmen (engl.: discrete-logarithm problem, DLP) in geeigneten Körpern $\mathbb{F}_p$ (p prim) sowie deren Verallgemeinerungen:

- Faktorisierungsproblem:
  gegeben: n mit $n \in \mathbb{N}$,
  gesucht: Primfaktoren p und q, sodass gilt: $n = p \cdot q$.
- Diskretes Logarithmus Problem für Gruppen (DLP)
  gegeben: $x \in G$ für eine Gruppe $G = \{g^0, g^1, g^2, \ldots\}$ der Ordnung $n \in \mathbb{N}$,
  gesucht: $a \in \mathbb{N}$, dass gilt: $g^a = x$.

Aktuelle White Paper [31, 7] aus diesem und dem vergangenen Jahr sprechen einer Familie von Funktionen, die sich in modernen Kryptoschemata durchgesetzt haben besonderes Potential zu, nämlich:

- Learning With Errors (LWE) Problem [135],
- seiner effizienteren ringbasierten Variante dem Ring Learning With Errors (RLWE) Problem [113] sowie
- ihren den "deterministischen Fehler" betreffenden Entsprechungen, den Learning With Rounding (LWR) Problem und dem Ring Learning With Rounding (RLWR) Problem [11].

Grob beschrieben ist es bei allen Problemen dieser Klasse die Aufgabe mit einer Suchversion, ein geheimes Ringelement s zu finden, das mehrere zufällige "rauschende Ringprodukte" mit s enthält, während wir mit der Entscheidungsversion solche verrauschten Produkte von einheitlich zufälligen Ringelementen leicht unterscheiden können. Unser Verschlüsselungsalgorithmus im praktischen Teil basiert auf dem RLWE Problem, weswegen wir in Kapitel 4 genauer darauf eingehen werden.

2.3 Grundlagen der Kryptoanalyse

Innerhalb der Kryptologie können die Kryptographie (aus dem altgriechisch zusammengesetzt aus "$\kappa\rho\upsilon\pi\tau o\varsigma$" (kryptós) für geheim und "$\gamma\rho\alpha\varphi\epsilon\iota\nu$" (gráphein) für schreiben), die sich mit der Entwicklung von neuen und der Verbesserung bekannter kryptografischer Algorithmen befasst und die Kryptoanalyse (altgriechisch "$\alpha\nu\alpha\lambda\upsilon\sigma\iota\varsigma$" (análusis) für Auflösung) unterschieden werden. Letztere beschäftigt sich mit den Schwachstellen der verwendeten Algorithmen und damit, wie sie ausgenutzt werden können [142]. In diesem Kapitel werden wir die Grundlagen zur Bestimmung der Sicherheit eines Kryptosystems erläutern, damit wir die Qualität unterschiedlicher Kryptosysteme, die wir im nächsten Kapitel erörtern werden, leicht beurteilen können.

2.3.1 Kerckhoffs Prinzip

Ein wichtiger Baustein sicherer Kryptosysteme ist das bereits 1883 formulierte Kerckhoffs'sche Prinzip [99]. Es besagt, dass die Sicherheit der Verschlüsselung nicht davon abhängig sein darf, ob die Verschlüsselungsmethode bekannt oder unbekannt ist.

Dieses Prinzip mag uns auf den ersten Blick vielleicht widersinnig erscheinen. Allerdings halten wir im Alltag ja auch nicht den Mechanismus unseres Wohnung- oder Autoschlosses geheim, sondern passen auf die Schlüssel auf. Es hat sich stets gezeigt, dass Systeme, die hohe Sicherheit zu bieten scheinen, weil Details des Verfahrens geheim sind, leicht zu knacken sind. Wir sprechen dabei von Sicherheit durch Verschleierung (engl.: security by obscurity). Bei einem System, das davon abhängig ist, wie etwas verschlüsselt wird (z.B. mit einer Verschlüsselungsmaschine), sind alle so verschlüsselten Chiffretexte lesbar, wenn der Mechanismus bekannt wird (wenn die Verschlüsselungsmaschine in die Hände des Feindes gerät). Warnende Beispiele für die Folgen, wenn Kerckhoffs Prinzip nicht eingehalten wird, sind regelmäßig der Presse zu entnehmen, wie beispielsweise der Skandal 2008 um die millionenfach verwendeten kontaktlosen Bezahlkarten, der Mifare Chipkarten [125].

2.3.2 Bibliotheken

Kryptoalgorithmen werden daher heute in Bibliotheken frei zugänglich gemacht. Das hat den entscheidenden Vorteil, dass sie von einer breiten (Fach-) Öffentlichkeit getestet und diskutiert werden können. So werden Schwachstellen in den Systemen schnell erkannt. Fehler können beseitigt, Empfehlungen ausgesprochen und Standards etabliert werden. Bibliotheken, die Implementierungen für modernere nivelliert homomorphe Verschlüssellungssysteme (LHE) mit unterschiedlichem Umfang aufweisen, sind nach [159] beispielsweise:

- cuHE [53]
- HEAAN [37]
- HElib [88]
- NFLlib [128, 129]
- Palisade [141]
- SEAL [32]

Die genannten Bibliotheken werden von internationalen Forschergruppen betreut, die sich seit Mitte 2017 um eine Standardisierung und Interoperabilität der Bibliotheken bemühen. Alle sechs dieser Universalbibliotheken zeigen entweder für den FV oder den BGV Algorithmus, auf die wir später genauer eingehen werden, in unterschiedlicher Weise eine Auswahlmöglichkeit für den zugrunde liegenden Ring, die Fehlerverteilung und die Parameterauswahl. Die Dokumentationen sind unter folgenden Links abrufbar [31, 7]. Eine umfassendere Auflistung mit 11 weiteren Bibliotheken finden Interessierte bei Boxen und Kollegen [20].

Neben der sicherheitsfördernden Diskussion der Algorithmen sprechen auch praktische Gründe dafür, Kerckhoffs Prinzip anzuwenden: Wir können einen Schlüssel einfacher geheim halten als einen Algorithmus und einen kompromittierten Schlüssel leichter ersetzen, statt einen ganzen Algorithmus zu tauschen. Bei vielen Teilnehmerpaaren an einem Kommunikationsprozess, wie z. B. innerhalb einer Firma oder eines Forschungsprojektes, ist es um einiges einfacher, unterschiedliche Schlüssel zu verwenden, statt unterschiedliche Algorithmen für jede Kombination zu entwerfen. Da der Schlüssel keinen Teil des Algorithmus bzw. seiner Implementierung darstellt, ist er im Gegensatz zum Algorithmus nicht anfällig gegen Reverse-Engineering.

2.3.3 Drei Ansätze zur Beschreibung der Sicherheit in der Kryptologie

Wir kennen drei Ansätze, die Sicherheit kryptografischer Systeme zu beschreiben:

1. informationstheoretischer Ansatz,
2. systembasierter Ansatz,
3. komplexitätstheoretischer Ansatz.

1. Der informationstheoretische Ansatz strebt nach unbedingter Sicherheit. Per Definition heißt ein Verschlüsselungsverfahren absolut sicher, wenn es auch gegen Angreifer mit unbeschränkten Ressourcen, wie beispielsweise hinsichtlich der Zeit und Rechenleistung, sicher ist. 1949 wies der Mathematiker Claude Elwood Shannon [149] nach, dass eine Verschlüsselung mit absoluter Sicherheit ausschließlich dann gegeben ist, wenn der Schlüssel dieselbe Länge wie die zu verschlüsselnde Nachricht besitzt. Dazu muss ein Schlüssel zufällig (gleichverteilt) sein und darf nur ein Mal verwendet werden (one-time-pad).

Bedenken wir, dass heute große Datenmengen verschlüsselt werden müssen, wie z.B. Bilder und Videos von Spionagesatelliten oder Sprach- und Videotelefonie über das Internet und dass eine Verschlüsselung von 10 GB dann einen Schlüssel der Größe von 10 GB benötigt, dann wird schnell klar, dass absolut sichere Verfahren nur mit extrem hohem Aufwand herzustellen sind. Nur vereinzelt hat der informationstheoretische Ansatz zu unbedingt sicheren und praktikablen Kryptosystemen geführt. Meist jedoch führt dieser Ansatz zu unpraktikablen Systemen, weil die Laufzeit und der Speicherbedarf viel zu hoch werden. Zudem ist es wahrscheinlich, dass ein gedachter, fiktiver Angreifer, der mit unbegrenzten Rechenkapazitäten und unendlich viel Zeit ausgestattet ist, Algorithmen brechen kann, die für einen realen Angreifer allerdings praktisch unlösbar sind.

Bei dem **2. systembasierten Ansatz** wird pragmatisch vorgegangen. Bei der Entwicklung neuerer Kryptosysteme versucht man "nur" aus dem Scheitern älterer Systeme Lehren zu ziehen und beruft sich auf das "Versuch und Irrtum" Prinzip. Der Nachteil dieses Ansatzes liegt auf der Hand: Es wird nur auf Bekanntes reagiert, während mit den beiden anderen Sicherheitsansätzen auch neue noch unbekannte kryptoanalytische Methoden theoretisch mitbetrachtet werden.

Daher wird der **3. komplexitätsbasierte Ansatz** meist zur Beurteilung der Algorithmen verwendet. Ein Kryptosystem gilt als komplexitätstheoretisch sicher oder berechnungssicher (engl.: computationally secure), wenn es einem Angreifer nicht möglich ist, das Schema mit einem für ihn lohnenden Aufwand zu brechen, d.h., der Zeitaufwand und die Kosten eines erfolgreichen Angriffs übersteigen den potentiellen Nutzen. Hinsichtlich eines Rechenmodells (z.B. Maschinen oder Boolsche Schaltkreise) legen wir also die mindestens erforderlichen Ressourcen zum Lösen eines schweren mathematischen Problems im

Sinne der Komplexitätstheorie als Komplexität fest. Damit können solche Kryptosysteme entworfen werden, die praktisch unmöglich gebrochen werden können. Dazu wird ein Sicherheitsparameter λ definiert. Ein Kryptosystem gilt als praktisch sicher, wenn es nur mit überpolynomiellem Ressourceneinsatz geknackt werden kann.

Das Maß an Sicherheit der unterschiedlichen Kryptosysteme kann somit auch formal ausgedrückt werden, z. B. als Wahrscheinlichkeit, mit der ein fiktiver Angreifer einen Schlüssel in Polynomialzeit erraten kann, die in der Komplexitätstheorie als Grenze zwischen praktisch lösbaren und praktisch unlösbaren Problemen erachtet wird.

Ein Beispiel: Angenommen, wir betrachten ein Verfahren, für das der Schlüsselraum $\{0,1\}^\lambda$ verwendet wird und die Schlüssel gleichverteilt zufällig gezogen werden. Die Schlüssel sind also Bitfolgen der Länge λ und es existieren 2^λ mögliche Schlüssel. Wir nehmen weiter den einfachsten Angriff auf den geheimen Schlüssel, einen Brute-Force Angriff an, bei dem ein Angreifer iterativ alle Schlüssel durchprobiert. Somit muss ein Angreifer im Worst-Case bei der Brute-Force Methode 2^λ Schlüssel durchprobieren oder kann den korrekten Schlüssel durch (einmaliges) Raten mit Wahrscheinlichkeit $(\frac{1}{2})^\lambda$ bestimmen. Im Fall Sicherheitsparameter $\lambda = 128$ existieren $2^{128} > 3.4 \cdot 10^{38}$ mögliche Schlüssel. Im Fall $\lambda = 512$ existieren sogar $2^{512} > 1.3 \cdot 10^{154}$ mögliche Schlüssel, was die Anzahl an Atomen im sichtbaren Universum, die auf 10^{80} geschätzt wird, deutlich übersteigt.

Bevor wir weiter auf die Formalisierung der Sicherheit eingehen werfen wir einen Blick auf die Kryptoanalyse, das Teilgebiet der Kryptologie, indem sich Wissenschaftler damit befassen wie Kryptosysteme zu knacken sind.

2.3.4 Angriffsarten und Angreifer von IT Systemen

Typischerweise werden bei Sicherheitsanalysen aktive und passive sowie interne und externe Angriffsarten unterschieden. Bei einem passiven Angriff wird nicht in das IT System bzw. die Infrastruktur eingegriffen, weshalb solche Angriffe schwer zu erkennen sind. Schutz bietet hierbei nur eine durchgängige Verschlüsselung und Verifizierung. Solche Szenarien finden sich oft in der Realität, weil nicht alle verschlüsselten Nachrichten vertraulich sind oder nicht unbegrenzt lange geheim gehalten werden. Für Letzteres sind Quartalsumsatzzahlen von Unternehmen ein Beispiel, die nur bis zu ihrer Veröffentlichung vertraulich sind. Passive Angreifer können von den ihnen zugänglichen Chiffretexten und passenden Klartexten auf den dahinter liegenden Algorithmus schließen.

Bei einem aktiven Angriff dagegen greift der Angreifer auf das System zu, wobei Sicherheitslücken ausgenutzt werden. Klassische Cyberangriffe sind dadurch gekennzeichnet, dass sie sich über das Internet verbreiten und sich mit Schadstoffsoftware in Form von Viren, Würmern oder Trojanern Zugang zu den Systemen verschaffen. Sie hinterlassen Spuren, wodurch die Angreifer im besten Fall identifiziert werden können.

Externe Angriffe werden durch Personen oder Gruppen von außerhalb ohne Hilfe durch das bedrohte System durchgeführt. Interne Angriffe dagegen werden von Personen mit Bezug zum System gestartet, wie beispielsweise (ehemalige) interne und externe Mitarbeiter oder auch Zulieferer etc.. Beim Social-Engineering werden menschliche Beziehungen ausgenutzt, um z. B. durch Vortäuschen falscher Tatsachen an Passwörter zu kommen. Es ist auch denkbar, dass ein Mitarbeiter durch Erpressung dazu gebracht wird, Schlüssel zu verraten [125]. Im Gegensatz zu klassischen unspezifischen "hit-and-run" Angriffen werden unter Advanced Persistent Threats (APT) komplexe und zielgerichtete Bedrohungen verstanden, die sich gegen ein oder wenige Opfer richten. Diese Bedrohungen (engl.: threats) werden von den Angreifern aufwändig vorbereitet, sind hochentwickelt (engl.: advanced) und dauern lange an (engl.: persistent). Nach einem Arbeitspapier der Bundesakademie für Sicherheitspolitik[1] können die Bedrohungen in sechs Cyber Konfliktypen klassifiziert werden, nämlich Hacktivismus bzw. Cybervandalismus, Cyberkriminalität, Cyberspionage, Cybersabotage, Cyberterrorismus und Cyberkrieg. Diese sind nicht immer klar voneinander abgrenzbar, da die höheren Eskalationsstufe meist die niedrigeren umfassen.

2.3.5 Angriffsmodelle zur Bestimmung der Sicherheit

Kryptoanalysten entwickeln Modelle damit die Sicherheit von Kryptosystemen geprüft werden kann. Die Akteure bei Angriffsmodellen im Sinne der Spieltheorie werden als Herausforderer und / oder Gegner bezeichnet. Oft wird davon ausgegangen, dass der Gegner polynomiell beschränkt ist. Wir können verschiedene Modellszenarien unterscheiden, z. B. [28, 147]:

1. Nicht Unterscheidbarkeit (engl.: Indistinguishability, IND): Der Gegner kann zwei Klartextnachrichten m_0 und m_1 frei wählen und erhält für eine der beiden Nachrichten den passenden Chiffretext. Er muss erraten, zu welchem Klartext er passt.

2. Angriff mit wählbarem Klartext (engl.: Chosen Plaintext Attack, CPA): Der Herausforderer lässt den Gegner einen oder mehrere Klartexte und die entsprechenden Chiffretexte einsehen. Zusätzlich darf der Angreifer sogar einen Klartext wählen, der verschlüsselt wird. Das Ziel des Angreifers ist es, den Schlüssel oder einen Algorithmus zur Entschlüsselung herauszufinden.

3. Angriff mit wählbarem Chiffretext (engl.: Chosen Ciphertext Attack, CCA1): Der Angreifer kann verschiedene Chiffretexte wählen und hat Zugriff auf den entschlüsselten Klartext.

4. Angriff mit adaptiv wählbarem Chiffriertext (engl.: Adaptive Chosen Ciphertext Attack, CCA2): Bei diesem Sonderfall erhält der Angreifer sogar die Möglichkeit, dass

[1] `https://www.baks.bund.de/sites/baks010/files/baks_arbeitspapier_2_2014.pdf`

beliebig viele Chiffretexte seiner Wahl entschlüsselt werden, mit der Ausnahme eines anderen Chiffretextes den er selbst entschlüsseln soll [147].

Die ersten zwei Szenarien beziehen sich auf passive Angriffe, die letzten beiden dagegen auf aktive. Sie sind nach ihrer Schwere hierarchisch geordnet, wobei die Schwereren jeweils die weniger schweren umfassen. Damit ein kryptographisches Verfahren als sicher gelten kann, muss die Erfolgswahrscheinlichkeit eines Angreifers selbst bei mehreren Versuchen möglichst klein sein, wofür sich der Begriff der Vernachlässigbarkeit (Definition 2.2.6) durchgesetzt hat.

Ist es das primäre Sicherheitsziel, dass ein PPT (probabilistic polynomial time) Angreifer, womit ein Angreifer gemeint ist, dessen Angriff nicht nur in seiner Rechenzeit beschränkt ist, sondern dem auch erlaubt wird, probabilistische Algorithmen zum Lösen der Einwegfunktionen zu verwenden, durch den Chiffretext keinerlei Informationen über den Klartext erhält, dann entspricht dies dem Begriff der semantischen Sicherheit, der 1983 in einer Arbeit von Goldwasser und Micali [84] eingeführt wurde.

2.3.6 Semantische und IND-CPA Sicherheit

2.3.1 Definition (semantische Sicherheit). Ein Verschlüsselungsverfahren $Enc(k, m)$ einer konkreten Nachricht m mit dem Schlüssel k ist semantisch sicher (engl.: semantic security), wenn es für jede Verteilung von Nachrichten gleicher Länge, jede Funktion f, die beliebige Informationen aus dem Klartext extrahieren kann und jeden effizienten Algorithmus A einen effizienten Algorithmus B, (mit ϵ als trivialer Information wie beispielsweise die gleiche Länge) gibt, sodass die Wahrscheinlichkeiten

$$P[A^{Enc(k,\cdot)}(Enc(k,m)) = f(m)] - P[\mathcal{B}(\epsilon) = f(m)]$$

vernachlässigbar sind. Dabei bezieht sich die erste Wahrscheinlichkeit darauf, Informationen über f aus dem Chiffretext zu erhalten und die zweite Wahrscheinlichkeit darauf, Informationen quasi "aus dem Nichts" zu generieren. Diese Form der Sicherheit deckt jedoch nur passive Angriffe ab, womit z.B. Lauschangriffe gemeint sind und keine aktiven, böswilligen Angriffe.

Die Existenz von mehrfach benutzbaren, semantisch sicheren Verfahren impliziert, dass wir diese Verfahren als Einwegfunktion nutzen können. Allerdings soll an dieser Stelle noch einmal daran erinnert werden, dass wir bis heute nicht wissen, ob Einwegfunktionen oder andere schwere Probleme überhaupt existieren [75]. Die Definition der semantischen Sicherheit ist daher eher inoffizieller Art. Zudem enthält die Definition mehrere Quantitative, weshalb sie relativ schwer zu handhaben ist. Darum wurden äquivalente Begriff zur Bestimmung der Sicherheit eines Kryptosystems eingeführt: IND-CPA steht für indistinguishability under chosen plaintext attacks. Goldwasser und Micali [84] bezeichneten ein entsprechendes Verfahren als polynomial secure.

2.3.2 Theorem. *Ein Verfahren ist genau dann semantisch sicher, wenn es IND-CPA sicher ist.*

Den Beweis können wir bei Goldwasser und Micali [84] nachlesen.

2.3.3 Definition (IND-CPA Sicherheit). Ein Kryptosystem wird als "nicht vom Klartext unterscheidbar" oder IND-CPA sicher bezeichnet, wenn ein probabilistischer polynomiell beschränkter Gegner das folgende Spiel im Sinne der Spieltheorie mit einer Wahrscheinlichkeit größer als $1 + v(\epsilon)$ nicht gewinnen kann, wobei $\epsilon = 2$ als vordefinierter Sicherheitsparameter und v eine vernachlässigbare Funktion darstellen. Nach [134] wird bei dem Spiel wie folgt vorgegangen:

1. Der Herausforderer führt den Algorithmus zur Schlüsselerzeugung durch. Dieser Algorithmus soll als Eingabe einen Sicherheitsparameter ϵ enthalten und sendet den öffentlichen Schlüssel an den Gegner.

2. Der Gegner kann eine polynomiell begrenzte Anzahl von Verschlüsselungen oder andere Berechnungen an einem Orakel durchführen.

3. Der Gegner wählt zwei verschiedene Nachrichten $m_0, m_1 \in P$, die ungefähr die gleiche Länge haben, und sendet sie an den Herausforderer.

4. Der Herausforderer wählt zufällig ein Bit $b \in \{0, 1\}$ und sendet $Enc(k, m \cdot b)$ an den Gegner.

5. Der Gegner hat die Aufgabe dieses b zu erraten. Dazu darf er erneut mit dem Orakel eine polynomiell begrenzte Anzahl von Verschlüsselungen oder andere Berechnungen durchführen und antwortet dann mit $b' \in \{0, 1\}$, also entweder mit 0 oder mit 1.

6. Der Gegner gewinnt das Spiel, wenn sein gewähltes Bit den gleichen Wert hat, den der Herausforderer gewählt hat.

Semantischen Sicherheit, IND-CPA und polynomial secure sind als Bezeichnung äquivalent.

2.4 Unterscheidungsmerkmale von Kryptosystemen

Wir können kryptographische Systeme auf verschiedene Weise unterscheiden, beispielsweise in deterministische und probabilistische Verschlüsselungsverfahren. Deterministische Verfahren erzeugen stets für den gleichen Klartext und Schlüssel einen identischen Schlüsseltext. Dabei kann ein Angreifer leicht erkennen, ob die gleiche Nachricht zweimal versandt wurde und nach Berechnung des Chiffretexts Informationen über den Klartext enthalten. Es liegt auf der Hand, dass deterministische Verfahren nicht semantisch sicher bzw. IND-CPA sicher sein können und damit nicht den heutigen strengen Sicherheitsanforderungen genügen.

2.4.1 Probabilisische Verfahren

Probablistische Verfahren dagegen beinhalten einen Zufallsfaktor, das Rauschen r. So wird dafür gesorgt, dass bei gleichem Klartext und Schlüssel bei mehreren Verschlüsselungsvorgängen verschiedene Schlüsseltexte gebildet werden. Angreifer können gleiche, probabilistisch verschlüsselte Nachrichten ohne Entschlüsselung nicht dadurch erkennen, dass sie den gleichen Schlüsseltext besitzen.

Probabilistische Systeme führen durch das Rauschen der Zufallszahlen zwangsweise zu einer Nachrichtenerweiterung. Es existieren mehrere mögliche Chiffretexte für einen Klartext, weshalb die absolute Größe des Chiffretextes länger sein muss, als die absolute Größe des Klartextes. Das Verhältnis dieser Größen wird als Nachrichtenexpansion bezeichnet [84]. Die große Sicherheit der probilistischen Kryptosysteme geht damit zwangsweise mit einer typischen langen Laufzeit und einem erhöhten Speicherbedarf einher.

2.4.1 Definition (probabilistisches Kryptosystem). Nach [97, S. 9] ist ein probabilistisches Kryptosystem mit öffentlichem Schlüssel als ein Tupel mit sechs Elementen (P, C, K, Enc, Dec, R) definiert, wobei P, C, K, Enc und Dec wie in Definition 2.1.1, dem Grundschema der Kryptologie, definiert sind und mit R eine Menge von Zufallszahlen hinzukommt. Zusätzlich sollten die folgenden Eigenschaften erfüllt sein:

- Jede Verschlüsselungsregel $Enc_k : P \times R \to C$ und die entsprechende Entschlüsselungsregel $Dec_k : C \to P$ sind Funktionen, sodass $Dec_k(Enc_k(m, r)) = b$ für jeden Klartext $m \in P$ und jedes $r \in R$ gilt.

- Für jedes $k \in K$ und für jedes $x \in P$ wird eine Wahrscheinlichkeitsverteilung $p_{k,x}$ auf C definiert, wobei $p_{k,x}(y)$ die Wahrscheinlichkeit angibt, dass y der Chiffretext ist, vorausgesetzt, dass k der Schlüssel für alle Möglichkeiten von $r \in R$ ist. Unter der Vorraussetzung $x, x' \in P$, $x \neq x'$ und $k \in K$ sind alle Wahrscheinlichkeitsverteilungen $p_{k,x}$ und $p_{k,x'}$ in polynomieller Zeit nicht unterscheidbar.

2.4.2 Symmetrische Verfahren

Wir können auch zwischen symmetrischen und asymmetrischen Verschlüsselungsverfahren differenzieren. Bei symmetrischen Verfahren existiert nur ein Schlüssel k, der sowohl zur Verschlüsselung Dec als auch zur Entschlüsselung Enc einer Nachricht m verwendet wird. Der Schlüssel stellt ein gemeinsames Geheimnis zwischen Alice und Bob dar, wie in Abbildung 2.1 dargestellt. Es gilt:

$$Dec(Enc(m, k), k) = m \ .$$

Der Schlüssel muss zu Beginn des Kommunikationsprozesses über eine sichere Verbindung zwischen Alice und Bob übertragen werden, was den hauptsächlichen Schwachpunkt

dieser Verfahrensgruppe darstellt. Unter der Annahme, dass der Datenverkehr paarweise geheimgehalten werden soll und n Teilnehmer kommunizieren, muss jeder Teilnehmer $n-1$ Schlüssel bereithalten, was es für große Teilnehmerzahlen wie beispielsweise im Internet ungeeignet macht. Werden die Schlüssel über unsichere Informationskanäle verteilt, müssen zusätzliche Protokolle implementiert werden, wie das Diffie-Hellman Schlüsselaustauschverfahren (engl.: D-H attack goals key exchange) [59]. Eine alternative Möglichkeit des Schlüsseltausches besteht darin, die Schlüssel in einer Initialisierungsphase über Smartcards vom Distributor zu verteilen. Der große Vorteil symmetrischer Verfahren ist ihre in der Regel sehr gute Performanz.

Ein Beispiel für einen heutigen Standards entsprechenden, oft eingesetzten Algorithmus ist der im Jahr 2001 designte Advanced Encryption Standard (AES), der in drei Varianten Daten mit kryptografischen Schlüsseln in einer Länge und damit einer Sicherheit von 128, 192 und 256 Bit verarbeiten kann [52]. Er ist seit 2002 Standard der US Regierung und Teil des ISO / IEC 18033-3 Standards, in dem die Standards der Blockchiffren zum Schutz der Vertraulichkeit der Daten definiert sind und wird vom Bundesamt für Sicherheit in der Informationstechnik (BSI) empfohlen.

Die Sicherheit des symmetrischen Verfahrens hängt von der Schlüsselerzeugung ab. Dazu gibt es Zufallszahlengeneratoren (engl.: Random Number Generator, RNG). Wir unterscheiden:

- Echte Zufallzahlengeneratoren produzieren Zahlen, die nicht reproduziert oder vorhergesagt werden können.
- Pseudozufallzahlengeneratoren (engl.: Pseudorandom Number Generator, PRNG) haben einen Startwert, von dem aus eine Sequenz von Werten erzeugt wird. Sie sind deterministisch und besitzen gute statistische Eigenschaften. Sie können mit dem Chi Quadrat Test überprüft werden.
- Kryptografisch sichere Pseudozufallszahlengeneratoren (engl.: Cryptographically Secure Pseudo Random Number Generator, CSPRNG) sind PRNG, die die zusätzliche Eigenschaft "nicht vorhersagbar" besitzen [125].

2.4.3 Asymmetrische Verfahren

Asymmetrische Verfahren werden in der Literatur meist als Public-Key Verfahren bezeichnet. Dem schließen wir uns im Folgenden an. Zur Verbesserung der Sicherheit wurde erst 1977 das erste asymmetrische Verschlüsselungsverfahren von Ronald L. Rivest, Adi Shamir und Leonard M. Adleman entwickelt [139], das nach den Initialen der Autorennamen (RSA) benannt ist. Es ist heute noch das wichtigste Public-Key Kryptosystem.

Bei Public-Key Verfahren erzeugt jeder Teilnehmer ein Schlüsselpaar, nämlich einen privaten geheimen Schlüssel (engl.: Secret-Key, sk) zum Entschlüsseln und einen öffentli-

chen Schlüssel (engl.: Public-Key, pk) zum Verschlüsseln, der auf dem privaten Schlüssel basiert. Der Secret-Key lässt sich nicht (leicht) aus dem Public-Key berechnen. Es gilt [125]:

$$Dec(Enc(m, sk), pk) = m \ .$$

Der große Vorteil von Public-Key Verfahren ist der einfache Schlüsselaustausch. Der private Schlüssel muss und darf nicht verteilt werden, sondern verbleibt bei seinem jeweiligen Besitzer. Die zu übertragenden öffentlichen Schlüssel sind nicht geheim und können daher auch über unsichere Kanäle verteilt werden. Zudem ist die Schlüsselzahl linear zu den Teilnehmern. Public-Key Verfahren können sowohl zur Sicherung der Vertraulichkeit wie auch zur Authentifizierung des Absenders der Informationen verwendet werden. Dabei wird der öffentliche Schlüssel des Absenders verwendet, um seine Identität zu überprüfen. Zu den Nachteilen der Public-Key Verfahren zählt, dass diese Algorithmen ca. 10 000 Mal langsamer als symmetrische Verfahren arbeiten [125]. Sie benötigen eine große Schlüssellänge, was bei mehreren Empfängern einer verschlüsselten Nachricht zu Problemen führt, da jedes Mal die Nachricht für jeden Empfänger extra verschlüsselt werden muss.

Damit in der Praxis die Schnelligkeit der symmetrischen und die Sicherheit der Public-Key Verschlüsselung verbunden werden können, werden hybride Verfahren eingesetzt. Dabei wird beispielsweise wie folgt vorgegangen:

1. Zuerst generiert Alice einen zufälligen Sitzungsschlüssel (engl.: session key) für eine (schnelle) symmetrische Datenverschlüsselung mit einem Zufallsschlüsselgenerator.
2. Alice sendet diesen Sitzungsschlüssel mit dem öffentlichen Schlüssel (pk) von Bob (langsame aber sichere asymmetrische Verschlüsselung, z.B. mit RSA) an Bob.
3. Mit seinem privaten Schlüssel (sk) entschlüsselt Bob den Sitzungsschlüssel (Public-Key Verschlüsselung).
4. Danach können die Daten mit den Sitzungsschlüsseln verschlüsselt übertragen werden (symmetrische Verschlüsselung).

2.4.4 C Bewertungsschema

Damit ein Public-Key System die heute geforderten starken Sicherheitsanforderungen erfüllen kann (IND-CPA sicher sein kann vgl. Kapitel 2.3.6), besteht es nach [74] aus einem Quadruple mit vier effizienten Algorithmen, die zusammen als C Bewertungsschema bezeichnet werden [8]: den beiden probabilistischen Algorithmen KeyGen, der Schlüsselgenerierung und der Verschlüsselung Enc. Weiter benötigen wir Dec, die Entschlüsselung sowie $Eval$, den Auswertungsalgorithmus. Diesen im Vergleich zum Grundschema der Verschlüsselung zusätzlichen Auswertungsalgorithmus können wir uns wie einen in das Kryptosystem eingebauten Computer vorstellen.

Die Arbeitsweise der Algorithmen ist wie folgt:

- **KeyGen (1^λ):** Der Schlüsselerzeugungsalgorithmus nimmt einen Sicherheitsparameter λ als Eingabe und erzeugt einen geheimen Schlüssel sk und einen öffentlichen Schlüssel pk. λ bezeichnet das gewünschte Sicherheitsniveau des Schemas. Zum Beispiel 128 Bit Sicherheit ($\lambda = 128$) oder 192 Bit Sicherheit ($\lambda = 192$). 1^λ stellt eine Bitfolge der Länge λ dar.

- **Enc (pk, m):** Der Verschlüsselungsalgorithmus nimmt als Eingabe pk und eine Nachricht $m \in M$ und gibt den Chiffretext ct aus.

- **Eval $(pk, f, ct_1, \ldots, ct_t)$:** Der Auswertungsalgorithmus nimmt als Eingabe pk und eine arithmetische Schaltung $f : M^t \to M$ in einer Klasse F von erlaubten Schaltungen und t Chiffretexten $ct_1, \ldots, ct_t$ ohne die Nachrichten $m_1, \ldots, m_t$ lesen zu können. Er gibt einen ausgewerteten Chiffretext ct aus.

- **Dec (sk, C):** Nach Eingabe von sk und einem Chiffretext ct gibt der Entschlüsselungsalgorithmus eine Nachricht m aus.

2.4.5 Verformbarkeit

Alle Public-Key Verschlüsselungsverfahren weisen eine besondere Eigenschaft auf, sie sind verformbar.

2.4.2 Definition (verformbar (engl.: malleable)). Wir sagen, dass ein Kryptosystem verformbar ist, wenn es bei einem Chiffretext ct, der die Klartextnachricht m darstellt, möglich ist, Funktionen f und g zu berechnen, wobei f nicht die Identitätsfunktion ist, sodass $f(ct)$ zu $g(m)$ entschlüsselt wird. Wenn es nicht möglich ist, ein solches f und g zu berechnen, wird das Kryptosystem als nicht formbar bezeichnet.

Obwohl die Verformbarkeit die theoretische Sicherheit eines Kryptosystems einschränkt, ist diese Eigenschaft bei homomorphen Kryptosystemen erwünscht, die speziell dafür konstruiert werden, Berechnungen auf verschlüsselten Daten zu ermöglichen. Offensichtlich können verformbare Kryptosysteme, womit Public-Key Verfahren gemeint sind, (vgl. Kapitel 2.4.3) nicht IND-CCA2 sicher sein, da der Gegner im vierten Schritt einfach $f(m \cdot b)$ an den Herausforderer senden kann, um $g(m \cdot b)$ zu empfangen (vgl. Kapitel 2.3.5). Der Gegner kann dann $g^{-1}(g(m \cdot b)) = m \cdot b$ berechnen, um zu bestimmen, welchen Wert $b \in \{0, 1\}$ der Herausforderer gewählt hat. Verformbare Verfahren sind damit anfällig für einen aktiven Man-in-the-middle Angriff. Alle homomorphen Algorithmen, die wir uns im Folgenden näher ansehen, sind verformbar und können höchstens IND-CCA1 sicher sein. Der Grund, warum die IND-CCA1 Sicherheit für homomorphe Verschlüsselungsschemata immer noch erreicht werden kann, liegt darin, dass der Gegner in dem IND-CCA1 Spiel keinen Zugriff auf ein Entschlüsselungs Orakel mehr hat.

2.5 Erstes Zwischenfazit

Wir haben in diesem Kapitel die wichtigsten Grundlagen geklärt, damit auch Laien die späteren Experimente und Herausforderungen bei der Parameterbestimmung nachvollziehen können. Es bleibt festzuhalten, dass es viele Eigenschaften gibt, die ein Public-Key Verfahren erfüllen muss. Dazu zählen:

1. In der Kryptographie sollen die Chiffretexte von zufällig entstanden Texten nicht zu unterscheiden sein. Das Erzielen einer solchen Eigenschaft "Zufälligkeit" wird bei modernen Systemen dadurch erzielt, dass wir Rauschen hinzufügen, bis unsere Nachrichten und das Rauschen nicht mehr unterscheidbar sind.

2. Das Schlüsselpaar aus K und die Algorithmen Enc und Dec sollten effizient berechenbar sein.

3. Die Chiffretexte sollten keine wesentlich größere Länge aufweisen als die zugehörigen Klartexte. Je nach Sicherheitsbedürfnis kann die Schlüssellänge variieren, wobei es natürlich auf die Gesamtlänge der Nachricht m ankommt, die verschlüsselt werden soll.

4. Der Public-Key sollte nur von berechtigten Person verändert werden können, womit Personen gemeint sind, die in Besitz des Schlüssels sind.

5. Der geheime Schlüssel sk sollte nicht in vertretbarer Laufzeit aus dem öffentlichen pk zu berechnen sein.

6. Aus einem Chiffretext ct können ohne Kenntnis des geheimen Schlüssels sk nicht oder nur sehr schwer der zugehörige Klartext m oder Teile davon rekonstruiert oder diese sinnvoll verändert werden.

Im folgenden Kapitel betrachten wir homomorphe Systeme genauer und werden uns mit den besonderen Herausforderungen befassen, die eben aufgezählten Eigenschaften bei homomorphen Systemen zu gewährleisten.

Kapitel 3

Homomorphe Verschlüsselungssysteme

Die Kernidee aller homomorphen Verschlüsselungssysteme (HE) ist, dass das Ausführen einiger Operationen an verschlüsselten Daten nach der Entschlüsselung das gleiche Ergebnis liefert, als ob die Berechnung auf dem ursprünglichen Klartext durchgeführt worden wäre. Wir können sie damit unter anderem zur Sicherung der Privatsphäre bei ausgelagerten Berechnungen sowie bei Speicher- und Datenbankanforderungen nutzen. Aus Kapitel zwei wissen wir, dass HE probabilistische Publik-Key Systeme sein müssen, damit sie den strengen Sicherheitsanforderungen der semantischen Sicherheit bzw. der IND-CPA Sicherheit genügen können. Sie müssen korrekte Ver- und Entschlüsselungen gewährleisten und sind C Bewertungsschemata. Im Folgenden werden wir uns ansehen, wie HE klassifiziert werden. Wir werden zudem einen Forschungsüberblick bieten, damit unser Beispielalgorithmus im Praxisteil nach Fan and Vercauteren (FV) [67] besser eingeordnet werden kann. Zudem werden Hinweise zur Auswahl des passenden HE Systems gegeben.

3.1 Homomorphismus

Der Begriff homomorph leitet sich von den altgriechischen Wörtern "$o\mu\acute{o}\varsigma$" (homos, gleich) und "$\mu o\rho\varsigma\acute{\eta}$" (morphe, Form) ab. Er wird in verschiedenen Fachgebieten dazu verwandt, um eine "gleiche Struktur" zu beschreiben. In der Algebra beschreibt das Konzept des Homomorphismus eine parallele Verknüpfung zwischen Operationen an zwei Objektgruppen bzw. Mengen.

3.1.1 Definition (Homomorphismus). Sei $f : P \to C$ eine Funktion mit der Umkehrfunktion $f^{-1} : C \to P$, welche Abbildungen zwischen den Elementen des Klartextraums P und des Chiffrierttextraums C vornehmen. Die Addition wird mit $\oplus$ bezeichnet und die Multiplikation mit $\otimes$. Wenn bei einer Operation $\oplus$ gilt:

$$\forall x, y \in P : f^{-1}(f(x) \oplus f(y)) = x \oplus y$$

ist die Abbildung f ein additiver Homomorphismus. Von einem multiplikativen Homomorphismus spricht man, wenn für f gilt:

$$\forall x, y \in P : f^{-1}(f(x) \otimes f(y)) = x \otimes y \ .$$

Kann f sowohl ein additiver als auch ein multiplikativer Homomorphismus sein, spricht man von einem algebraischen Homomorphismus. Homomorphismus beschreibt also, dass sich die Elemente des Wertebereichs untereinander bezüglich der Operation genauso verhalten wie die Elemente des Definitionsbereichs.

3.1.2 Definition (Gruppenhomomorphismus). Seien $(G, \circ)$, $(H, \bullet)$ Gruppen. Eine Abbildung $\varphi : G \to H$ ist ein Gruppenhomomorphismus, wenn $\varphi(g_1 \circ g_2) = \varphi(g_1) \bullet \varphi(g_2)$ für alle $g_1, g_2 \in G$ gilt.

3.1.3 Definition (Ringhomomorphismus). Seien R und R' Ringe. Eine Abbildung $\varphi : R \to R'$ heißt Ringhomomorphismus, wenn für alle $r_1, r_2 \in R$ gilt:

- $\varphi(r_1 + r_2) = \varphi(r_1) + \varphi(r_2)$,
- $\varphi(r_1 \cdot r_2) = \varphi(r_1) \cdot \varphi(r_2)$,
- $\varphi(1) = 1$.

3.2 Klassifikation homomorpher Kryptosysteme

Um eine Einleitung in die Klassifikation der verschiedenen HE Systeme zu geben, die noch heute uneinheitlich in der wissenschaftlichen Literatur gehandhabt werden, stellen wir im Folgenden zwei Ansätze zur Vereinheitlichung der verschiedenen Klassifikationsbegriffe vor. Die Autoren beider Artikel [7, 8] wurden durch den verwirrenden uneinheitlichen Begriffsgebrauch zu den Standardisierungsbemühungen in ihren englischen Artikel motiviert. An dieser Stelle wollen wir auch auf die unterschiedlichen Übersetzungen ins Deutsche hinweisen, was zusätzlich verwirrend ist. In diesem jungen Forschungsfeld ist noch keine einheitliche deutsche Begriffsübersetzung zu beobachten. Sprachpuristen bitten wir um Verständnis, dass wir im Folgenden daher in solchen Fällen die englischen Fachbegriffe an erster Stelle nennen und ihre Akronyme verwenden werden. Wir verzichten in diesem Unterkapitel auf die mathematischen Definitionen, die in den Originalarbeiten [8, 7] nachgelesen werden können.

3.2.1 Klassifikation nach Albrecht et al., (2018)

Der Begriff homomorphe Verschlüsselung (engl.: Homomorphic Encryption, HE) wird zum einen, wie von uns bis hierher, für die gesamte Klasse homomorpher Algorithmen verwendet. Gleichzeitig kann damit aber auch eine bestimmte Algorithmenklasse der HE gemeint

sein. Albrecht und Kollegen [7] unterscheiden grob im Rahmen aktueller Standardisierungsbemühungen, die nur praxistaugliche Algorithmen betreffen, zwischen (1.) homomorpher Verschlüsselung (HE) und (2.) vollständig homomorpher Verschlüsselung (engl.: Fully Homomorphic Encryption, FHE).

1. HE unterstützt entweder die Addition oder die Multiplikation, aber nicht beide zusammen. Bekannte Beispiele dieser Klasse, auf denen zahlreiche Verbesserungen und Modifikationen beruhen, sind:

- Das ElGamal Verfahren [66]: $f(m_0, m_1) = m_0 \cdot m_1$
- Das Goldwasser-Micali Verfahren [84]: $f(m_0, m_1) = m_0 \oplus m_1$
- Das Paillier Verfahren [126]: $f(m_0, m_1) = m_0 + m_1$

Diese Algorithmenklasse steht zwar nicht im Mittelpunkt dieser Untersuchung, allerdings ist es unser Ziel, Kryptologielaien in die Materie einzuführen. Damit Interessierte den Forschungsfortschritt in dieser Klasse leicht nachvollziehen können und so Hilfe zur Auswahl eines HE Systems bekommen, listet Tabelle 3.1 diese Verfahren sowie ihre Verbesserungen und Modifikationen komprimiert auf. In der Spalte HE System finden sich die Quellenangaben zu den in den folgenden Spalten erläuterten Modifikationen. Eine Auflistung der möglichen Rechenoperationen ist in der Spalte Operationen zu finden. Die Addition wird wie bisher mit $\oplus$ und die Multiplikation mit $\otimes$ bezeichnet. Zusätzlich verwenden wir $\otimes c$ für Multiplikationen mit einer Konstante c sowie $\ominus$ für Subtraktionen. Erweiterung beinhaltet die jeweilige Erweiterungsrate des Chiffretextes durch Anwendung der jeweiligen Verschlüsselung. In der letzten Spalte werden die Verschlüsselungen und Modifikationen kurz erläutert und die herausstechenden Besonderheiten zusammengefasst. Die Tabelle ist zwar das Ergebnis einer umfassenden, systematischen Literaturrecherche unter Einbezug verschiedener Reviews, kann aber keinen Anspruch auf Vollständigkeit erheben. Einen wesentlich weiterführenden Einblick in die Anwendung der HE Systeme bietet die Übersichtsarbeit von Liebig [105].

2. FHE unterstützen dagegen sowohl die Addition als auch die Multiplikation. Gehen wir von einem C Bewertungsschema mit KeyGen, *Enc*, *Eval*, *Dec* aus, das wir im Kapitel 2.4.4 besprochen haben, dann können wir auch sagen, dass sich HE und FHE dahingehend unterscheiden, ob nur ein Auswertungsalgorithmus Eval_f für entweder die Addition oder die Multiplikation vor dem Entschlüsseln beliebig oft angewendet werden kann und sofort der Entschlüsselungsalgorithmus *Dec* zur Anwendung kommt wie bei HE oder sich zunächst weitere Schaltungen anschließen, mit denen weitere Funktionen berechnet werden können, wie bei FHE. Abbildung 3.1 visualisiert diesen Unterschied.

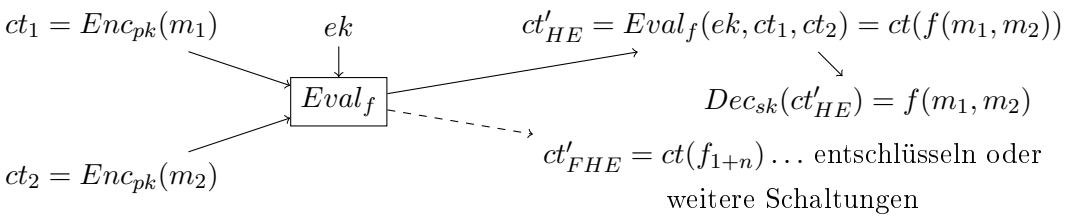


Abbildung 3.1: Unterschied FHE zu HE anhand des Auswertungsalgorithmus $Eval$. ct_1, ct_2: Chiffretexte, Enc_pk: Verschlüsselungsfunktion mit Public-Key pk, m_1, m_2: Klartext Nachrichten, ek: Evaluation Key, $Eval_f$: Auswertungsalgorithmus für Funktion f, ct'_{HE}: Ergebnis Berechnung mit HE, ct'_{FHE}: Ergebnis Berechnung mit FHE.

FHE eignen sich nach Albrecht et al. [7] besonders für mehrere Benutzer, die Berechnungen auf einem Server oder anderen verteilten Systemen mit ihren vertraulichen Daten ausführen möchten wie beispielsweise im Forschungskontext oder etwa in einem kommunalen Grid System. In einem solchen Fall können wir eine Sicherheits Nebeneigenschaft erwarten: Evaluation Privacy (Auswertungsprivatheit) besagt, dass der Chiffretext ct verbirgt, welche Berechnungen homomorph durchgeführt wurden um ct zu erhalten [7]. In anderer Literatur wird dafür der Begriff circuit privacy (Schaltungsprivatheit) genutzt:

$$Enc_{pk}(ct(m_1, m_2)) \approx Eval_f(ek, ct_1, ct_2)[143] \ .$$

Diese Eigenschaften sind nützlich, wenn die Berechnung privater Daten anderen überlassen wird wie beispielsweise beim Cloudcomputing oder im Client-Server (Klient-Server) Modell. Möglicherweise möchte der Server seinen proprietären Algorithmus vor dem Klienten verbergen. Evaluation Privacy ist zudem eine Voraussetzung, wenn wir FHE für eine sichere zwei Parteien Kommunikation mit geringer Interaktion verwenden wollen. Bei anderen Spezialfällen und gleichzeitiger Berücksichtigung bestimmter Angriffsszenarien können weitere passende Nebeneigenschaften definiert werden, deren Aufzählung an dieser Stelle den Rahmen sprengen würde. Interessierten empfehlen wir als Einstieg [7].

HE System	Operationen	Erweiterung	Erläuterungen und Besonderheiten
RSA, 1978 [138]	• $\otimes$	1	• In der Originalversion deterministisch und daher nicht IND-CPA sicher, es sei denn, Nachrichten werden zufällig aufgefüllt (padding). Wenn m_1 und m_2 zufällig als m_1' und m_2' aufgefüllt werden, ist aber die Wiederherstellung von $m_1 \cdot m_2$ von $m_1' \cdot m_2'$ unmöglich. Daher wird das RSA Kryptosystem nicht oft wegen seiner homomorphen Eigenschaften verwendet. • RSA kommt oft zur Sicherung von Internet-, Bank- und Kreditkartentransaktionen zum Einsatz, etwa zur Verschlüsselung, der Datenauthentisierung (Signaturverfahren) oder Instanzauthentisierung (Challenge-Response Verfahren).
Goldwasser Micali (GM) 1984 [84]	• $\otimes$ • modulo 2 • XOR	n	• Verschlüsselung ist einfach ($r = 2$). • Die Entschlüsselung erfordert nur eine einzige Berechnung und dauert nur $\mathcal{O}(\ell(p)^2)$. • GM unterstützt die Rerandomisierung von Nachrichten ohne Kenntnis des Klartextes. • Nachteil: Eingabe besteht nur aus einem einzigen Bit, d.h., die Verschlüsselung von k Bits kostet $\mathcal{O}(k \cdot \ell(p)^2)$, was nicht effizient ist (Chiffretextgröße: $k \cdot \log N$). • Aktuell sollte n mindestens 1024 Bit betragen, daher wachsen Nachrichten um einen Faktor von über 1000 und GM wird in Abhängigkeit der Anwendung evtl. unbrauchbar.
Joye-Libert Verfahren (GM Modifikation), 2013 [95]	• $\otimes$ • modulo 2 • XOR	≥ 4	• Basiert auf dem GM Verfahren, • hat eine höhere Effizienz bei der Entschlüsselung (ist schneller als das Naccache-Stern Schema), • Chiffretextgröße von $\log_2 N$, • benötigt weniger Speichers GM, • erstes, auf quadratischen Resten (QR) basierende Schema, das bei richtiger Abstimmung eine verbesserte lossy Trapdoor Funktion enthält.
ElGamal, 1984, [66]	• $\otimes$	2	• Das ElGamal Schema ist eine verbesserte Version des Diffie-Hellman Schlüsselaustausch Algorithmus [59]. • Es wird hauptsächlich in hybriden Verschlüsselungssystemen verwendet, um den geheimen Schlüssel eines symmetrischen Verschlüsselungssystems zu verschlüsseln.
ElGamal Varianten, 1997, [51]	• $\otimes$ bzw. • $\oplus$	2	Cramer, Gennaro und Schoenmakers stellen zwei Varianten vor: • eine threshold Variante (mit mehreren Kommunikationspartnern) des ElGamal Kryptosystems und • eine neue Variante von ElGamal Kryptosystems, die additiv homomorph anstelle von multiplikativ homomorph ist.
ElGamal Variante, 1999, [70]	• $\otimes$	2	• Eiichiro Fujisaki and Tatsuaki Okamoto zeigen, wie das ElGamal Schema auf IND-CCA2 Sicherheit erweitert werden kann. • Es verliert dabei die Formbarkeit.

Tabelle 3.1: Überblick (P)HE: Varianten und Modifikationen. $\oplus$: Addition, $\ominus$: Subtraktion, $\otimes$: Multiplikation, $\otimes c$: Multiplikation mit Konstante. Wenn nicht ausdrücklich anders vermerkt, sind alle HE probabilistisch und IND-CPA sicher. Fettgedruckte Namen sind für Originalversionen reserviert.

HE System	Operationen	Erweiterung	Erläuterungen und Besonderheiten
Benaloh, 1994, [14]	• $\oplus$ • $\ominus$ • $\otimes$ • $\otimes c$	n/r	• Benaloh schlug eine Erweiterung des GM Cryptosystems vor, in dem die Nachricht blockweise anstelle einzelner Bits verschlüsselt werden. Es ist einfach zu implementieren und hat eine verringerte Nachrichtenexpansion verglichen mit GM, was aber auf Kosten einer Zunahme der Entschlüsselungskomplexität geht, die auf $\mathcal{O}(\sqrt{r})$ erhöht wird. • Mögliche Operationen sind: Multiplikation von zwei verschlüsselten Nachrichten sowie Multiplikation mit einer bekannten Konstante und Exponentiation durch eine bekannte Konstante. • Es unterstützt die Rerandomisierung von Nachrichten ohne Kenntnis des Klartextes. • Es wurde ursprünglich von Cohen (Benaloh) und Fischer [43] im Kontext eines kryptographischen Wahlsystems vorgestellt.
Naccache-Stern, 1998 [120]	• $\oplus$ • $\ominus$ • $\otimes$ • $\otimes c$	≥ 4	• Sicherheit der Ursprungsversion wird vermutet, bietet aber keinen formalen Beweis. • Die semantische Sicherheit einer modifizierten probabilistischen Version [120] beruht auf der Schwierigkeit der π-ten Reste für jedes der π, dabei muss σ so gewählt werden, dass es keine Informationen über die Faktorisierung von n enthüllt nach der Hybridtechnik von [83]. • Während im Benaloh Kryptosystem Nachrichten durch eine kleine Primzahl p begrenzt sind, werden sie im Naccache-Stern Kryptosystem durch das Produkt vieler kleiner Primzahlen begrenzt. • Eine verschlüsselte Nachricht wird wiederhergestellt, indem sie von jeder der kleinen Primzahlen mithilfe des modulo operators entschlüsselt wird und dann die Nachricht unter Verwendung von chinesischem Restetheorem rekonstruiert wird. So wird eine kleinere Nachrichtenerweiterung möglich. • Nachrichten werden bis $\log_2 p$ Bits verschlüsselt, d.h., bei größeren Nachrichten ist es dem Benaloh-Fischer System vorzuziehen.
Sander, Young, and Yung, 1999, [144]	• AND	kn	• Autoren stellen (unter Verwendung des AND Satzes nach DeMorgan) ein Verfahren mit OR und NOT Gattern in einem Zwei Parteien Setting vor, wobei eine Partei Kenntnis von einer Schaltung hat, die eine geheime Funktion f berechnet, und die andere eine geheime Eingabe x für $f(x)$ besitzt. • Das gleichzeitig davon unabhängig vorgestellte HE Schema ist probabilistisch, IND-CPA sicher und basiert auf GM mit einer zusätzlichen Kodierungsstufe. • Um sicher zu sein, muss n mindestens 1024 Bit lang sein. • Die zusätzliche NC^1 Schaltung ist eine Protokolltiefenschaltung, die als ein Binärbaum visualisiert werden kann. Jeder Knoten des Baums funktioniert entweder als ein ODER Gatter oder ein NOT Gatter, wobei Eingaben zu jedem Gatter an der Unterseite des Baums bereitgestellt werden. Die Ausgabe der Schaltung ist der Wert, der von dem Gatter an der Wurzel des Baums ausgegeben wird.

Tabelle 3.1 (Fortsetzung): Überblick (P)HE: Varianten und Modifikationen. $\oplus$: Addition, $\ominus$: Subtraktion, $\otimes$: Multiplikation, $\otimes c$: Multiplikation mit Konstante. Wenn nicht ausdrücklich anders vermerkt, sind alle HE probabilistisch und IND-CPA sicher. Fettgedruckte Namen sind für Originalversionen reserviert.

HE System	Operationen	Erweiterung	Erläuterungen und Besonderheiten
Okamoto-Uchiyama, 1998, [122]	• $\oplus$ • $\ominus$ • $\otimes$ • $\otimes c$	3	• verwendet diskrete Falltür Logarithmen, • Nachteil: Nachrichten sind durch p begrenzt, aber arithmetisch ist modulo $n = p^2 \cdot q$, sodass ein Chiffretext etwa dreimal so groß ($\log_2 N$) wie der entsprechende Klartext ist, was allerdings eine Verbesserung gegenüber den früheren Systemen darstellt. • Der gebundene Nachrichtenbereich kann ein Problem darstellen, wenn das Verschlüsselungssystem speziell wegen seiner homomorphen Eigenschaften verwendet wird. Da p ein Teil des privaten Schlüssels ist, kann ein Benutzer nicht genau angeben, was die gebundenen Nachrichten sind, was problematisch sein kann, wenn viele verschlüsselte Nachrichten summiert oder multipliziert werden sollen. • Subtraktion wird durch Multiplizieren von $c_0 \cdot c_1^{-1}$ erreicht. • Addition und Subtraktion min einer Konstante können durchgeführt werden, indem die Konstante mit dem öffentlichen Schlüssel verschlüsselt wird und eine der zuvor definierten Operationen verwendet wird.
M-Okamoto-Uchiyama (modifiziertes Schema, 1999 und 2001)	• $\oplus$ • $\ominus$ • $\otimes$ • $\otimes c$	3	• Coron, Naccache und Paillier (1999) [46] sowie Choi, Choi und Won (2001) [65] haben jeweils eine effizientere Variante des Okamoto-Uchiyama Kryptosystems vorgeschlagen, die die Komplexität der Entschlüsselung bei gleicher Sicherheit reduziert: wähle g so, dass gilt: $g\lambda = 1 + n \mod n^2$, dann kann die Entschlüsselung mithilfe einer speziellen Logarithmusfunktion $L(x)$ auf $m = L(c\lambda \mod n^2) \mod n$ vereinfacht werden. • Okamoto und Uchiyama haben selbst versucht, ihr Kryptosystem über eine elliptische Kurve mit p Punkten zu definieren, wobei diskrete Logarithmen effizient zu berechnen sind [70]. Sie konstatieren jedoch, dass eine sichere Konstruktion mit ihrem Ansatz nicht möglich ist.

Tabelle 3.1 (Fortsetzung): Überblick (P)HE: Varianten und Modifikationen. $\oplus$: Addition, $\ominus$: Subtraktion, $\otimes$: Multiplikation, $\otimes c$: Multiplikation mit Konstante. Wenn nicht ausdrücklich anders vermerkt, sind alle HE probabilistisch und IND-CPA sicher. Fettgedruckte Namen sind für Originalversionen reserviert.

HE System	Operationen	Erweiterung	Erläuterungen und Besonderheiten
Paillier, 1999, [126]	• $\oplus$ • $\ominus$ • $\otimes$ • $\otimes c$	2	• Neben der Addition sind unter Verwendung der verschlüsselten Klartexte $E(m_1)$, $E(m_2)$ und des öffentlichen Schlüsselpaares (n, g) zusätzliche Grundoperationen auf den Klartexten $m_1, m_2 \in \mathbb{Z}^*_{n^2}$ möglich: $$E(m_1) \cdot E(m_2)(\mod n^2) = E(m_1 + m_2(\mod n))$$ $$E(m_1) \cdot gm^2(\mod n^2) = E(m_1 + m_2(\mod n))$$ $$E(m_1) \cdot m_2(\mod n^2) = E(m_1 m_2(\mod n)) .$$ Die Gleichungen zeigen, wie sich die auf verschlüsselten Daten berechneten Operationen auf die Klartexte auswirken. • Paillier verwendet diskrete Falltür Logarithmen. • Das Entschlüsseln eines Geheimtextes im Paillier Kryptosystem erfordert die Verwendung einer Logarithmusfunktion L ähnlich dem Okamoto-Uchiyama Kryptosystem. • Zusätzlich wird (ähnlich wie bei der Erweiterung des GM zum Benaloh Kryptosystem) hier nun das Benaloh System erweitert, indem es auf der Schwierigkeit der Berechnung bestimmter Reste höherer Ordnung anstelle quadratischer Reste modulo a prim p basiert. • Die Chiffretext Erweiterung wird auf nur 2 reduziert, was bedeutet, dass die Größe eines Chiffretextes doppelt so groß ist wie die des Klartextes. • Die Verschlüsselungskosten sind nicht zu hoch. Die Entschlüsselung benötigt eine Potenzierung modulo n^2 zur Potenz $\lambda(n)$ und eine Multiplikation modulo n. • Paillier zeigte in seiner Arbeit, wie die Entschlüsselung effizient durch das chinesische Resttheorem gehandhabt werden kann. • Mit geringerer Expansion und geringeren Kosten im Vergleich zu früheren Systemen ist Pailler attraktiv und wird allgemein als das praktischste HE angesehen. • Typische Einsatzgebiete sind e-Wahlen und als threshold Schema.
Paillier (Schnelleres Entschlüsseln) [126, 130]	• $\oplus$ • $\ominus$ • $\otimes$ • $\otimes c$	2	• Neben dem grundlegenden Kryptosystem schlug Paillier auch eine effizientere Variante, die jedoch eine andere Sicherheitsannahme erfordert sowie eine formbare Trapdoor-Permutation Version vor, die jedoch veränderte homomorphe Operationen wie das grundlegende Paillier Kryptosystem enthält.

Tabelle 3.1 (Fortsetzung): Überblick (P)HE: Varianten und Modifikationen. $\oplus$: Addition, $\ominus$: Subtraktion, $\otimes$: Multiplikation, $\otimes c$: Multiplikation mit Konstante. Wenn nicht ausdrücklich anders vermerkt, sind alle HE probabilistisch und IND-CPA sicher. Fettgedruckte Namen sind für Originalversionen reserviert.

HE System	Operationen	Erweiterung	Erläuterungen und Besonderheiten
M-Paillier (modifiziertes) Paillier [65]	• $\oplus$ • $\ominus$ • $\otimes$ • $\otimes c$	2	• Wie das modifizierte Okamoto-Uchiyama Kryptosystem basiert die M-Paillier Variante (Choi, Choi, and Won, 2001) auf der Idee, g so zu wählen, dass $L(g\lambda \mod n^2) \approx 1 \mod n$ ist, sodass die Entschlüsselung auf $m = L(c\lambda \mod n^2) \mod n$ vereinfacht werden kann. • Dies wird erreicht, indem g so gewählt wird, dass $g\lambda = 1 + n \mod n^2$ gilt. • Die Modifikation kann sowohl auf die grundlegenden als auch auf die schnellen Entschlüsselungsvarianten des Paillier Kryptosystems angewendet werden.
Paillier Schnidt-Samoa- Takagi [146]	• $\oplus$ • $\ominus$ • $\otimes$ • $\otimes c$	3	• Das grundlegende Paillier Kryptosystem ist über $\mathbb{Z}_{n^2}$ definiert, das isomorph zu $\mathbb{Z}_n^* \times \mathbb{Z}_n$ ist, wenn $n = p \cdot q$. • Die vorgeschlagene Variante ist über $\mathbb{Z}_n^* \times \mathbb{Z}_{p \cdot q}$ definiert, wenn $n = p^2 \cdot q$. • Zudem basiert es auf einer anderen Sicherheitsbedingung. • Die Verschlüsselung ist effektiver, leider Chosen Chiffretext Attack anfällig.
Paillier Elliptische Kurve, 2000, [127]	• $\oplus$ • $\ominus$ • $\otimes$ • $\otimes c$	2	• Galbraith [71] zeigte, dass das Pailliers Elliptic Curve Cryptosystem nicht sicher ist, da es unter dem gleichen Problem leidet, auf das Okamoto und Uchiyama beim Versuch, eine elliptische Kurvenvariante ihres Kryptosystems zu erstellen, gestoßen sind [122]. • Nämlich, dass die Verwendung von elliptischen Kurven, bei denen diskrete Logarithmen einfach sind, sogenannte anomale Kurven, es ermöglicht, dass der private Schlüssel aus der notwendigen öffentlichen Information wiederhergestellt wird. • Als Antwort hat Galbraith einige Ideen geliefert, wie man ein Paillier ähnliches Kryptosystem über einer elliptischen Kurve aufbauen kann, mit einem konkreten Beispiel.
Damgård-Jurik 2001 [55]	• $\oplus$ • $\ominus$ • $\otimes$ • $\otimes c$	$(s+1)/s$	• Erweiterung des Paillier Kryptosystems, indem Nachrichten von $\mathbb{Z}_n$ genommen und auf Elemente von $\mathbb{Z}_{n^{s+1}}^*$ abgebildet werden, womit eine Chiffretext Erweiterungsrate von $(s+1)/s$ erreicht wird. • Dies hat den negativen Aspekt zur Folge, dass die Größe des Modulus erhöht wird, wodurch die modulare Arithmetik verlangsamt wird und die minimale Nachrichtengröße erhöht wird. • Das Paillier Kryptosystem nutzte die Tatsache, dass $\mathbb{Z}^*$ isomorph $\mathbb{Z} \times \mathbb{Z}^*$ ist. • Das Damgård-Jurik Kryptosystem erweitert es, indem gilt: $\mathbb{Z}_{n^{s+1}}^*$ ist isomorph zu $G \times H$ mit $G = \mathbb{Z}_{n^s}$ und $H = \mathbb{Z}_n^*$. Somit ist die Faktorgruppe $\bar{G} = \mathbb{Z}_{n^{s+1}}/H$ isomorph zu Z_{n^s}.

Tabelle 3.1 (Fortsetzung): Überblick (P)HE: Varianten und Modifikationen. $\oplus$: Addition, $\ominus$: Subtraktion, $\otimes$: Multiplikation, $\otimes c$: Multiplikation mit Konstante. Wenn nicht ausdrücklich anders vermerkt, sind alle HE probabilistisch und IND-CPA sicher. Fettgedruckte Namen sind für Originalversionen reserviert.

HE System	Operationen	Erweiterung	Erläuterungen und Besonderheiten
Damgård-Jurik Varianten, 2003, [56]	• AND	$(s+1)/s$	• Eine Variante modifiziert das Kryptosystem mit flexiblen Längen derart, dass mehrere Benutzer denselben öffentlichen Modulus verwenden können, während sie die homomorphen Eigenschaften des ursprünglichen Paillier Kryptosystems beibehalten. • Dies wird erreicht, indem ein privater Schlüssel verwendet wird, der nicht auf der Faktorisierung des Moduls beruht. • Wenn s zunimmt, wird die Arithmetik langsamer. • Es gibt eine effiziente Thresholdvariante, wobei der private Schlüssel in einer Gruppe an t Personen einer Untergruppe verteilt werden kann. Soll eine Nachricht entschlüsselt werden, müssen mindestens t Personen zusammenarbeiten

Tabelle 3.1 (Fortsetzung): Überblick (P)HE: Varianten und Modifikationen. $\oplus$: Addition, $\ominus$: Subtraktion, $\otimes$: Multiplikation, $\otimes c$: Multiplikation mit Konstante. Wenn nicht ausdrücklich anders vermerkt, sind alle HE probabilistisch und IND-CPA sicher. Fettgedruckte Namen sind für Originalversionen reserviert.

3.2.2 Klassifikation nach Armknecht et al., (2015)

Armknecht und Kollegen [8] unterscheiden feiner, weil sie auch solche Systeme mit beachten, die damals und heute noch nicht praxistauglich waren bzw. sind. Sie unterscheiden vier verschiedene Klassen.

1. Teilweise homomorphe Verschlüsselung (engl.: Partial Homomorphic Encryption, **PHE**). Diese Gruppe entspricht den HE nach Albrecht et al., die in Kapitel 3.2.1 erläutert wurde.

2. Einigermaßen homomorphe Verschlüsselung (engl.: Somewhat Homomorphic Encryption, **SHE**). SHE Systeme unterstützen sowohl die Addition als auch die Multiplikation. Die Anzahl der Additionen ist in der Regel weniger beschränkt als die Anzahl der Multiplikationen. Das ist darin begründet, dass das der Sicherheit dienende Rauschen mit jeder Addition deutlich weniger zunimmt, als mit jeder Multiplikation. Wird das Rauschen im Ciffretext zu groß (engl.: overflow), ist eine korrekte Entschlüsselung nicht mehr möglich. Den Wert, um den das Rauschen noch wachsen kann, sodass die Entschlüsselung zu korrekten Ergebnissen führt, nennen wir Rausch Budget. Durch bestimmte Verfahren können die mathematischen Fähigkeiten erweitert werden, womit SHE Systeme zu FHE bzw. LHE gemacht werden können, die weiter unten erklärt werden.

2005 publizierten Dan Bonet, Eu-Jin Goh und Kobbi Nissim [16] das erste SHE. Sie definierten auf Basis des Subgruppen Entscheidungsproblems ein Kryptosystem mit einer Erweiterung von $\frac{n}{r}$ über einer elliptischen Kurvengruppe G. Dabei unterstützt eine bilineare Paarung (Weil Paarung [119]) die isomorphe Gruppe G_1. Punkte der elliptischen Kurve werden dabei in endliche Körper (engl.: Finite Fields, FFs) umgewandelt. Somit kann eine homomorphe Multiplikation zweier verschlüsselter Nachrichten unter Erhaltung der additiven homomorphen Eigenschaften des Kryptosystems berechnet werden. Da keine bilinearen Paarungen bekannt sind, die auf FFs angewendet werden können, sind keine weiteren Multiplikationen möglich. Somit kann bei diesem Schema eine unbegrenzte Anzahl von homomorphen Additionen durchgeführt werden, gefolgt von einer einzelnen Multiplikation, gefolgt von einer unbegrenzten Anzahl von Additionen. Weitere Beispiele finden sich bei [16, 79, 116, 143].

3. Niviliert homomorphe Verschlüsselung (engl.: Leveled Homomorphic Encryption, **LHE**). Damit ist die praxistaugliche Version der SHE bzw. FHE gemeint. FHE werden wir weiter unten erklären. Solche Algorithmen basieren, anders als PHE, meist auf Ringen und gelten als post quantum sicher. Additionen und Multiplikationen sind zwar wie bei SHE begrenzt, doch legt der Anwender selbst durch die Wahl der Parameter fest, wie viele Multiplikationen berechnet werden können. Das schränkt zwar einerseits die mathe-

matischen Fähigkeiten im Vergleich zu den FHE Systemen ein, hält aber andererseits die typischen rauschbedingten hohen Leistungskosten in einem vertretbarem Rahmen. Daher müssen LHE neben den Grundeigenschaften zusätzlich die Eigenschaft der Kompaktheit aufweisen:

- Die Ausgabe von *Eval* ist auf höchstens $p(\lambda)$ Bits eingeschränkt, wobei λ der Sicherheitsparameter ist.
- die Länge der Ausgabe von *Eval* darf nicht von einem Hilfswert abhängen.

LHE können durch einen zusätzlichen Bootstrapping Schritt zu FHE gemacht werden. Unser Beispielalgorithmus FV [67] zählt zu dieser Klasse. In der Definition von Albrecht und Kollegen zählen LHE zur Klasse der FHE.

4. Vollständige homomorphe Verschlüsselung (engl.: Fully Homomorphic Encryption, **FHE**). Diese Verfahrensgruppe ist durch einen Bootstrapping Schritt gekennzeichnet, durch den Schaltungen unbegrenzter Tiefer ermöglicht werden und damit beliebige Berechnungen durchgeführt werden können. Als zusätzliche Eigenschaft zu der Kompaktheit definiert Armknecht [8] die i-hop Korrektheit. Damit meint er die Korrektheit bei einer Auswertung in mehreren Stufen (hop). Weiter unterscheided er, je nach Stufenanzahl, "multi-", "poly-" oder "∞-"hop. Es gibt derzeit noch kein praxistaugliches FHE Schema, weil der Bootstrapping Schritt die ohnehin beträchtlichen Leistungskosten der LHE enorm vergrößert.

Bemerkung 1: Es finden sich auch weitere Bezeichnungen wie S/FHE in der Literatur. Immer dann, wenn ein FHE praxisbezogen eingesetzt wird, muss es sich genau genommen nach Albrecht und Kollegen [7] um ein LHE handeln. Nur LHE und die PHE werden heute in der Praxis tatsächlich als Problemlöser genutzt, was die grobe Einteilung in der praxisbezogenen Arbeit von Albrecht und Kollegen 2018 erklärt.

Abbildung 3.2 visualisiert die erörterte Systematik. Wir sehen, in jedem FHE stecken auch LHE, SHE und PHE. Sie zeigt auch, dass die Effizienz im Sinne von Laufzeit in der Regel bei den klassischen PHE am besten ist. Sie wird (in Abhängigkeit des Schemas) über SHE und LHE zu FHE hin schlechter. Bei der Flexibilität in Hinblick auf mögliche Rechenoperationen auf verschlüsselten Daten ist es umgekehrt. Mit anderen Worten: Flexibilität kostet Laufzeit.

Bemerkung 2: Zur Abgrenzung wollen wir auf die funktionellen Verschlüsselungsschemata (Functionell Encryption, FE) hinweisen. Dabei werden statt nur einem Public-Key mehre öffentliche Schlüssel für bestimmte Merkmale erzeugt, sodass später getrennt bzw. selektiv, im Sinne von funktional, nur die interessierenden Merkmale betrachtet und berechnet werden können. Allerdings kann nur der Besitzer des Hauptschlüssels die verschlüsselten

Daten lesen. Es kann leicht zu Verwechselungen kommen, wenn in Arbeiten, die sich mit dem Datamining bei gleichzeitigem Schutz die Privatsphäre befassen, die Autoren Yang, Zhong und Wright [162] zitiert werden, die diese Methode anwenden. Allerdings können FE Systeme durch Anpassungen zu FHE werden [8].

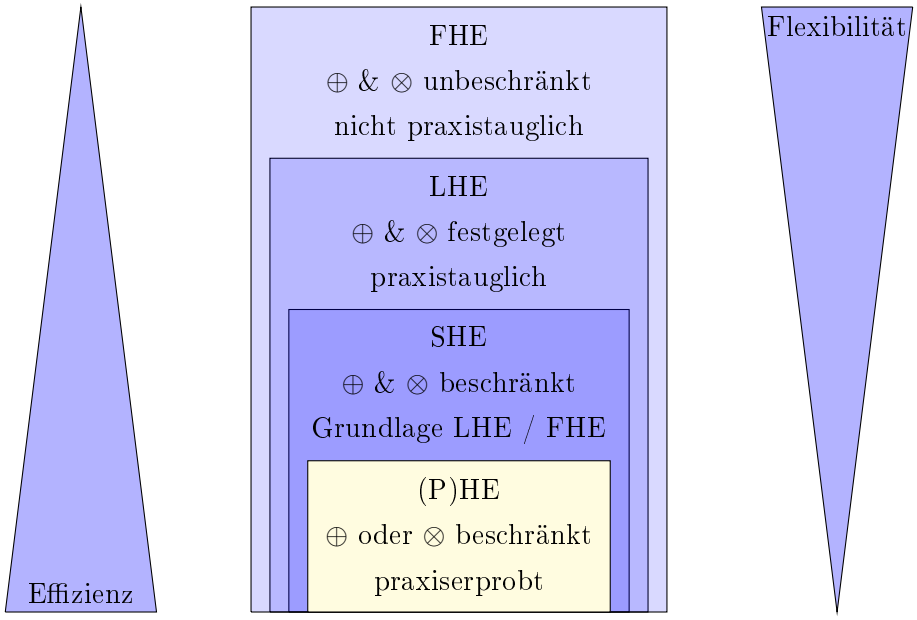


Abbildung 3.2: Überblick der Klassifikationen homomorpher Verschlüsselungssysteme. ⊕: Addition, ⊗: Multiplikation (mit einer Konstanten oder mit Chiffretexten), PHE: teilweise homomorphe Verschlüsselung, SHE: Einigermaßen homomorphe Verschlüsselung, LHE: Nivilierte homomorphe Verschlüsselung, FHE: Vollständig homomorphe Verschlüsselung.

3.3 Kurze Geschichte und Forschungsstand FHE

Im Folgenden gehen wir auf die kurze Geschichte der FHEs ein, sodass der rasante Forschungsfortschritt der letzten Jahre deutlich wird. Bis 1978 wurde mit verschlüsselten Daten nur eins gemacht: sie wurden entschlüsselt. Die beiden "Väter" des RSA, Ron Rivest, Len Adleman und ihr Kollege Michael L. Dertouzos erkannten das Potential der Verformbarkeit "ihres" Public-Key Verfahrens und schlugen nur wenige Monate nach Veröffentlichung des RSA vor, auch auf verschlüsselten Daten zu rechnen. Ihr algebraisch homomorphes Verschlüssellungskonzept, bei dem beliebige Berechnungen auf verschlüsselten Daten vorgenommen werden können, nannten sie "Privacy Homomorphism" [137]. Leider wiesen Brickell und Yacobi [27] Anfälligkeiten für Chosen Plaintext (CPA) und Chosen Ciphertext Attacken (CCA) nach. Seitdem wurde nun von vielen Wissenschaftlern intensiv an der Möglichkeit geforscht, auf verschlüsselten Daten rechnen zu können und f zu erweitern. Es dauerte allerdings noch über 30 Jahre, bis Craig Gentry 2009 in seiner Dissertation [72]

und anschließend auf einem Krytokongress [73] zeigte, dass FHE theoretisch möglich sind. Seinen "Haupttrick" um aus einem SHE ein FHE zu konstruieren, nannte er Bootstrapping. Damit kann auch heute noch ein SHE und LHE zu einem FHE Schema erweitert werden. Darum stellen wir es an dieser Stelle kurz vor.

3.4 Das Gentry Schema

Das Gentry Schema besteht aus drei "Zutaten":

1. Gentry stellte zunächst ein dem Kryptoschema von Goldreich, Goldwasser und Halevi (GGH Kryptosystem) [82] ähnliches SHE vor, das aber auf Idealen basiert. Mit diesem SHE wird eine begrenzte Anzahl von Additionen und Multiplikationen, aber keine Funktionen mit hohem Polynom Grad unterstützt, denn es muss immer unser Ziel sein, das Rauschen so gering wie möglich zu halten (vgl. Kapitel 3.2.2).

2. Gentry reduzierte durch ein Squash Verfahren der Entschlüsselungsschaltung den Grad des Entschlüsselungspolynoms. Dieses Verfahren ist in der Folgezeit häufig kritisiert worden und wird bei späteren Verfahren nicht mehr eingesetzt, weshalb wir auf eine genauere Erläuterung verzichten.

3. Der Bootstrapping Algorithmus ist Gentrys Haupttrick, um das Rauschen auf dem Chiffretext zu reduzieren. So können erneut, als hätte es das Rauschen nicht gegeben, weitere Additionen und Multiplikationen ausgeführt werden.

Der Begriff Bootstrapping wird auch in anderen statistischen Verfahren, wie beispielsweise bei multiplen Regressionen, dafür genutzt, bei scheinbar ausweglosen Situationen eine Lösung zu finden. Er wurde ursprünglich als Übersetzung des Ausdrucks "sich mit den eigenen Haaren aus Sumpf ziehen" aus den Geschichten um den Lügenbaron von Münchhausen verwendet.

Gentry's Bootstrapping Methode basiert auf einer simplen Idee: Er überlegte sich, dass uns theoretisch nichts daran hindert, bereits verschlüsselte Daten ct_1 mit einem neuen Schlüssel sk_2 wieder zu verschlüsseln (ct'_1) und auch den alten Schlüssel mit diesem neuen zu verschlüsseln sk'_1. Wird nun ct'_1 mit sk'_1 verschlüsselt, erhalten wir einen Chiffretext ct_2. Dieser verschlüsselt dieselbe Nachricht mit reduziertem Rauschen. Es gilt:

$$\begin{aligned} ct'_1 &= Enc(ct_1, sk_2) \ , \\ sk'_1 &= Enc(sk_1, sk_2) \ , \\ ct_2 &= Dec(ct'_1, sk'_1) \ . \end{aligned}$$

An Abbildung 3.3 lässt sich gut erkennen, dass nach dem Bootstrapping der neu entstandene Ciphertext ct_2 sich mit dem neuen Schlüssel sk_2 entschlüsseln lässt. Dabei wird das

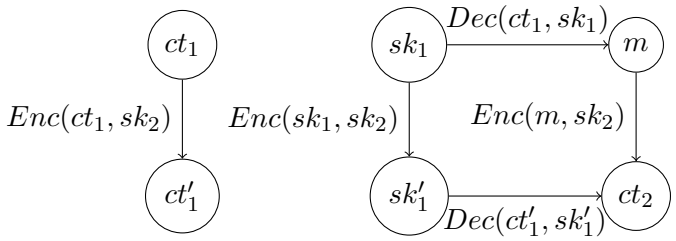


Abbildung 3.3: Bootstrapping. *Enc*: Verschlüsselung, *Dec*: Entschlüsselung, ct_1, ct_2: verschlüsselte Nachricht m, sk_1, sk_2: Secret-Key, sk'_1: verschlüsselter Secret-Key, ct'_1: erneut verschlüsselter Chiffretext. Quelle: Eigene Darstellung nach [12].

Rauschen jedoch nicht vollständig entfernt. Wir müssen bedenken, dass, obwohl die Entschlüsselungsoperation das Rauschen des verschlüsselten Textes in Bezug auf den ersten öffentlichen Schlüssel eliminiert, dennoch Rauschen erzeugt wird. Dies geschieht immer, wenn die Schaltung in Hinblick auf den zweiten öffentlichen Schlüssel evaluiert wird. Um eine Verbesserung zu erzielen, sollte das endgültige Rauschen also kleiner als das Anfangsrauschen sein. Zudem sollte es möglich sein, mindestens eine weitere Operation durchzuführen. Sind diese Bedingungen erfüllt und wenden wir Bootstrapping iterativ an, erhalten wir ein FHE Schema.

Mit anderen Worten, wir betrachten ein Schema dann als bootstrappbar, wenn es seine eigene Entschlüsselungsschaltung homomorph bewerten kann. Somit muss das Schema bezüglich der Gatter, die in der Entschlüsselungsschaltung erscheinen, homomorph sein. Gleichzeitig muss es in der Lage sein, die gesamte Entschlüsselungsschaltung auszuwerten, ohne einen Überschuss an Rauschen zu erzeugen. Die Hauptnachteile des Bootstrapping sind:

- erneute Entschlüsselungen haben eine große multiplikative Tiefe, womit die Anzahl hintereinander ausgeführter Multiplikationen gemeint ist.

- Der Speicherverbrauch der Verschlüsselung ist enorm, denn die Verschlüsselung einer bereits verschlüsselten Nachricht erzeugt einen quadratischen Overhead.

3.5 Drei Generationen FHE

Mit der theoretischen Vorstellung von Genty's FHE beginnt die Ära der modernen Kryptoschemata [72, 73]. Die Forschung zu FHE hat zwei Hauptziele: Zum einen sollen der rauschinduzierte Overhead reduziert und die mögliche multiplikative Tiefe erhöht werden, um Bootstrapping zu ermöglichen. Zum anderen wird nach neuen Wegen gesucht, die rauschbedingten Fehler der Chiffretexte zu reduzieren, was als Chiffriertextaktualisierung (engl.: refresh) bezeichnet wird.

3.5.1 Erste Generation FHE

Nach 2009 wurde in der ersten Generation zunächst daran gearbeitet, vollhomomorphe Systeme zu implementieren, die alle auf Gitteridealen beruhen, wie es von Gentry [73] vorgeschlagen wurde. 2010 präsentierten Smart und Vercauteren den ersten Versuch, Gentrys Schema [151] zu implementieren, wobei ihnen jedoch der Bootstrapp Schritt nicht wie vorgesehen gelang. Damals waren alle verfügbaren FHE Schemata so ineffizient, dass sie einfach auf keiner Hardware liefen. Tabelle 3.7 listet die FHE Schemata der ersten Generation auf. Seitdem sind große Fortschritte erzielt worden.

Problem	2009	2010
• Gitter • Subsummen	• Gentry stellt erste FHE theoretisch vor [72] • Erster Implementierungsversuch bei dem Bootstrapping misslingt [151]	
• DLP		• AGCD basiertes Schema vorgeschlagen [157]

Tabelle 3.7: Erste Generation FHE (2009-2010). DLP: diskretes Logarithmus Problem.

Gentry präsentierte mit Halevi einen neuen Ansatz, bei dem das oft kritisierte Squash Verfahren durch das modifizierte Diff-Hellmann Verfahren von ElGamal ersetzt wurde [76]. 2011 optimierten Gentry und Halevi das Ursprungsschema [77] mit einigen Ideen von Smart und Vercauteren, womit die erste Implementierung gelang. Auf der Crypto Konferenz 2012 zeigten Gentry, Halevi und Smart, dass es "nur" 36 Stunden dauert, um AES zu verschlüsseln [79]. Nur ein Jahr später war dies bereits deutlich schneller möglich [1]. Der Sicherheitsparameter $\lambda = 72$ Bit, einer öffentliche Schlüsselgröße von 2.3 GB und einer benötigtem refresh Dauer von 30 Minuten, die später als Bootstrapping bezeichnet wurde, zeigten jedoch, dass die hohen Leistungskosten einem Praxiseinsatz immer noch im Wege standen.

3.5.2 Zweite Generation FHE

FHE Systeme der zweiten Generation sind dadurch gekennzeichnet, dass sie auf den Problemen learning with Errors (LWE) oder Ring learning with Errors (RLWE) basieren, die wir in Kapitel 2.2.3 bereits angerissen haben. Die Vektoren können dabei als Polynome betrachtet werden. Berechnungen mit Polynomen haben meist eine bessere Komplexität verglichen mit Vektoren. Die ersten FHE Systeme dieser Art, die (noch) unter Nutzung Gentry's bootstrapping Methode implementiert wurden, publizierten Brakerski und Vaikuntanathans [25, 24]. Zudem führten die Autoren eine Key-Switching Technik ein, die die Größe der Chiffretexte verringert. Sie zeigten auch, wie der zugrunde liegende Modulus des Rings geändert werden kann. Damit war es nicht länger notwendig, die Entschlüsselungsschaltung wie zuvor zu "quetschen", denn die Dimension des Chiffretextes ist reduziert, so-

dass das zugrunde liegende SHE Schema den Grad des neuen Entschlüsselungsalgorithmus homomorph bewerten kann. Mit dieser Methode können wir also auch nicht bootstrapbare SHE als Grundlage eines FHE verwenden.

Mehrere Modifikationen wurden vorgenommen, um solche Systeme zu verbessern, einschließlich der maßstabsfreien Variante von Brakerski [21] und der NTRU Verschlüsselung [111]. Für das Akronym NTRU finden sich in der Literatur verschiedene Erklärungen: N-th degree TRUncated polynomial ring, Number Theory Research Unit oder Number Theorists aRe Us. In [78, 22] werden etliche Schemata mit Chargenfähigkeit vorgestellt. Ihre Implementierung wird in [79] beschrieben, wobei sich die Autoren auf Systeme von [23, 78] stützen. Eine Version mit Bit Vektoren präsentierten Brakerski, Gentry und Halevi in [22].

Besondere Bekanntheit erlangte das von Brakerski, Gentry und Vaikuntanathan vorgeschlagene BGV Schema. Diese vollständig homomorphe Verschlüsselungsmethode kommt ohne Gentrys Bootstrapping Verfahren aus [23]. Die Autoren nutzen dazu die Techniken von Brakerski und Vaikuntanathan zur Schlüssel- und Modulusschaltung, um ein besseres Rauschmamagement zu erzielen. Im BGV Schema wird die Größe des öffentlichen Schlüssels gemäß der Tiefe der Schaltungen, die homomorph verarbeitet werden soll, linear erhöht, um die Verwendung von Bootstrapping zu vermeiden. Wir können dann für eine gegebene Schaltung von angemessener Tiefe die Parameter des Schemas auswählen, um die Schaltung in einer akzeptablen Zeit homomorph zu bewerten. Sie führten dazu den Begriff der nivelliert homomorphen Verschlüsselung (LHE) ein. Wir können die Autoren als die Väter der praxistauglichen FHE betrachten (vgl. Kapitel 3.2.2).

Das BGV Schema wurde anschließend implementiert und eine homomorphe Auswertung der AES Schaltung [79] durchgeführt. Damit eine Schaltung mit multiplikativer Tiefe d unter Verwendung der Modulusschalttechnik homomorph ausgewertet werden kann, muss der öffentliche Schlüssel (leider) d verschiedene Versionen des Bewertungsschlüssels (eng.: Evaluation-Key) enthalten. Dies macht einen sehr großen Speicher von 256 GB RAM erforderlich, damit der öffentliche Schlüssel gespeichert werden kann.

Auf der Crypto 2012 schlug Brakerski den neuen Begriff der Skaleninvarianz [21] für LHE vor. Im Gegensatz zu einem Schema, das Modulus-Switching verwendet, behalten die Chiffretexte eines skaleninvarianten Schemas während der gesamten homomorphen Auswertung den gleichen Modulus bei. Nur eine Kopie des skaleninvarianten Bewertungsschlüssels muss gespeichert werden, womit die Speicherkosten sinken. Diese Technik übertrugen Fan und Vercauteren [67] auf das BGV Schema sowie Bos, Lauter, Loftus und Naehrig [19] auf das López-Alt, Tromer und Vaikuntanathans Schema. Die daraus resultierenden Schemata werden FV bzw. YASHE genannt. Tabelle 3.8 fasst die Entwicklung der Schemata der zweiten Generation zusammen.

Problem	2011	2012	2013
• Gitter • Subsummen	• Erste gelungene Implementierung des Gentry Schema [77] • Squashverfahren ersetzt [118]	-	-
• DLP	• Verringerung Publik Key Größe • erste Implementation [45]	• Weitere Schlüsselgrößeeduktion • modulus switching vorgestellt [47]	• Batching eingeführt [39] • Entwicklung Permutationen [35]
• (R)LWE	• Auf LWE und RLWE basierende FHEs • Schlüssel und Moduls Switching [152]	• Rauschen wird veringert • Skaleninvariantes Schemata eingeführt [111] • Brakerski [21] • BGV [76] • FV Schema [67]	• Key und Moduls switsching verworfen [81]
• NTRU	• von RLWE zu NTRU [152]	• NTRU basiertes Schema mit key und Moduls switching [111]	• NTRU basiertes skaleninvariantes Schema [19] • YASHE [19]

Tabelle 3.8: Zweite Generation FHE (2011-2013). DLP: diskretes Logarithmus Problem, (R)LWE: (Ring) learning with Errors, NTRU: N-th degree TRUncated polynomial ring.

3.5.3 Dritte Generation FHE

Obwohl bereits 2010 Van Dijk, Gentry, Halevi und Vaikuntanathan [157] vorgeschlagen hatten, Ganzzahlen als Basis eines FHE, wie bereits bei der FV Verschlüsselung zu verwenden, ist dies erst ab ca. 2014 ein Kennzeichen aller Systeme der dritten Generation. Hierdurch wurde die Effizienz erneut erhöht und die öffentlichen Schlüsselgrößen reduziert [45, 47]. Zudem verbessert die Batch Technik das Leistungsniveau [35, 39]. Auch wurden weitere skalierungsfreie, ganzzahlen FHE Schemata [44] nach den Vorschlägen in [21] publiziert. Wie bereits bei der Klasse der HE Schemata zu beobachten war, bestehen auch die Schemata der späteren FHE Generationen jeweils aus Verbesserungen der früheren. Miglore et al. ordnen SHIELD [118] der dritten Generation zu. Sie basieren auf unserem bereits auf Ganzzahlen basierenden Beispielsalgorithmus FV [67] und YASHE' [19], die die Autoren der zweiten Generation zuordnen. Mit größeren Kosten für die ersten homomorphen Multiplikationen haben Schemata der dritten Generation ein viel besseres asymptotisches Verhalten, postulieren Bonnoron und Kollegen in [17].

Jüngst ist die neueste Variante des FV Schema erschienen. Chen und Kollegen [34] erweitern den Klartextraum auf $\frac{\mathbb{Z}}{(b^n+1)\cdot\mathbb{Z}}$. Bei einem Vergleich erlaubt das neue Schema die Auswertung von Schaltungen Tiefe 9 mit 32 Bit an ganzzahligen Eingängen, während das ursprüngliche FV Schema mit den gleichen Parametereinstellungen nur eine Tiefe von 2 erreicht. Ebenso ist der BGV Algorithmus verbessert worden. Halevi und Shop zeigten experimentell, dass sich die Laufzeit bei "typischen" Parametern um den Faktor 30 bis 75 reduziert werden kann [87]. Dies gelang durch eine Reduktion des Speicherbedarfs der

Evaluation-Keys für die "typischen" Parameter. Damit wird auf eine weitere Herausforderung bei kryptologischen Verfahren hingewiesen, auf die wir später eingehen werden: die Parameterbestimmung. Zum Zeitpunkt dieser Arbeit sind weder der oben erwähnte verbesserte FV Algorithmus noch die soeben erwähnte Variante des BGV Algorithmus in den entsprechenden Bibliotheken (SEAL bzw. HElib) öffentlich verfügbar. Die Arbeiten [34, 87] verdeutlichen aber die unverminderte und rasante Entwicklung auf diesem Gebiet. In Tabelle 3.9 kann der Forschungsfortschritt der dritten Generation bis heute nachgelesen werden.

Problem	2014	2015	2016	2017	2018
• Gitter • Subsummen			• Polinominal zeit Quantum Attacke [50]		
• DLP	• skaleninvariantes Schema eingeführt [44]	• Reduktion von LWE zu AGCD [41]			
• (R)LWE	• RLWE scheint sich als Basis durchzusetzen [100]	• Bootstrapping in weniger als 1 Sek [64]	• Bootstrapping in weniger als 0.1 Sekunde [42] • Vergleichsarbeiten: YASHE vs. FV [103], YASHE vs FV vs SHIELD vs FNTRU [118]	• Neuskalierungsprozedur und neue Batching Technik [38] • Arbeit zur postquantum Sicherheit: LWE besser als RLWE [18]	• multiplikative Tiefe von 9 mit 32-Bit erreicht durch verbessertes FV Schema [34] • BGV 30-75x schneller für "typische" Parameter [87]
• NTRU			• von Gentry (2013) zu NTRU [62] • verschiedene Angriffe auf YASHE [6, 36, 102]		

Tabelle 3.9: Dritte Generation FHE (2014-2018). DLP: diskretes Logarithmus Problem, (R)LWE: (Ring) learning with Errors, NTRU: N-th degree TRUncated polynomial ring.

3.6 Auswahl des passenden HE Schemas

Die Frage, welches Anwendungsschema für welches Anwendungsszenarium geeignet ist, können wir heute nach einer Betrachtung des Forschungsstands und einer systematischen Literaturanalyse nur schwer beantworten. Bis heute gibt es keinen Benchmark, um einen fairen und vollständigen Überblick über die Effizienz all dieser Systeme zu geben. Es fehlen Untersuchungen, die mehrere Verfahren miteinander vergleichen.

Im Laufe unserer Literaturrecherche konnten wir nur diese vier Vergleichsarbeiten finden:

- Lepoint und Naehrig [103]: Vergleich von FV und YASHE
- Costache und Smart [48]: Vergleich von FV, YASHE, NTRU und BGV
- Kim und Lauter [101]: Vergleich von BGV und YASHE
- Migliore, Bonnoron und Fontaine [118]: Vergleich von FV, SHIELD, und F-NTRU

Eine Beurteilung ist daher sehr schwer möglich. Kryptographie ist zudem nicht die einzige Herausforderung. Das Rechnen mit verschlüsselten Daten beinhaltet eigene Implementierungsprobleme. Dazu müssen wir erwähnen, dass Angriffe auf YASHE erfolgreich waren, sodass als Folge Microsoft in ihrer SEAL Bibliothek YASHE durch FV ersetzte.

Als Entscheidungshilfe fassen wir in Tabelle 3.10 die in der Fachliteratur dokumentierten Laufzeiten und Besonderheiten, die wir in unserer Literaturrecherche finden konnten, zusammen. In der Spalte Implementierung befinden sich die Literaturquellen, in denen die jeweiligen Implementierungen und Testergebnisse dokumentiert sind. Wie wir wissen, hängt die angegebene Laufzeit von der verwendeten Hardware, dem Sicherheitsparameter λ, den Berechnungen innerhalb der Schaltung und der aus der Schaltung resultierenden Tiefe an AND Gattern (und damit der multiplikativen Tiefe auf Bit Niveau) ab. Diese Angaben befinden sich in den folgenden Spalten. Besonderheiten beschreiben wir in der letzten Spalte.

Da die angegebenen Laufzeiten durch die Berechnung auf unterschiedlich gut ausgestatteten Computern erfolgten, ist an dieser Stelle keine direkte Vergleichbarkeit der Laufzeitmessungen gegeben. Dies zeigt, dass nur sehr wenig praxisrelevante Dokumentationen über die reellen Laufzeiten auf heutigen Heimrechnern existieren, und es wird mit einem Blick deutlich, dass selbst durch den rasanten Forschungsfortschritt noch heute wann immer es möglich ist, die PHE Systeme den LHE Schemata vorzuziehen sind.

Basis Schema	Implement.	Plattform	Schaltung	λ	AND Tiefe	Gesamtzeit	Anzahl paral. Enc	Relative Zeit pro Block	Besonderheit
BGV, Brakerski, Vaikuntanathan, 2011[24]	Naehrig et al., 2011 [121]	• Intel duo 2,2GHz • 1GB RAM	• Mittelwert von n=1024 nur per Addition	-	-	20ms	-	-	• ohne Bootstrapping • MAGMA Arithmetik • Diagramme mit weiteren Beispielwerten • Keygen = 250ms • Enc = 24ms • Dec = 15-26ms $<$ 1 ms
			• Division nach Enc Varianz von n=100 • 1 Multiplikation nötig • Zahlen mit 128-bits	-	-	6s	-	-	
	Gentry et al., 2012a [79]	• Intel Xeon 2,0GHz • 256GB RAM	• AES-128 • byte Slice • 10 Durchgänge	80	40	$<$ 48h	54	37min	• Leveled Variante BGV ohne Bootstrapping • Batchtechnik
			• AES-128 • bit Slice • 10 Durchgänge	80	40	65h	720	5min	
DGHV, Van Dijk et al., 2010 [157]	Cheon et al., 2013 [35]	• Intel i7 3,4Ghz • 32GB RAM	• AES-128 • byte wise	72	40	18,3h	33	33min	• Batch FHE mit Bootstrapping • vergleichbare Zeiten mit Gentry, (2012) • Desktop mit weniger Speicher
			• AES-128 • state wise	72	40	113h	531	12min 46s	
	Coron et al., 2014 [44]	• Intel Xeon E5 2690 2,9GHz	• AES-128 • byte wise	72	40	3h 35min	569	23s	• Lineare multiplakative Tiefe (statt exponentieller)
			• AES-128 • state wise	80	40	102h	1875	195s	

Tabelle 3.10: Überblick FHE: Implementierungen, Laufzeiten und Besonderheiten. Basis Schema: implementiertes Schema, Implement.: Implementation und Testdurchführung, Plattform: Daten des verwendeten Computers, λ: Sicherheitsparameter, AND Tiefe: multiplikative Tiefe auf Bit Niveau, Gesamtzeit: Ergebnis der Messung, Anzahl paral. Enc: Anzahl paralleler Verschlüsselungen, Relative Zeit pro Block: benötigte Laufzeit pro Block einer Blockchiffre, Besonderheit: Besonderheiten der Implementierung. In Anlehnung an Quelle: [114].

Basis Schema	Implement.	Plattform	Schaltung	λ	AND Tiefe	Gesamtzeit	Anzahl paral. Enc	Relative Zeit pro Block	Besonderheit
FV, Fan ,Vercauteren, 2012 [67]	Wu, Haven, 2012 [160]	• AMD Opteron	• Lineare Regression N=1024 • Kovarianz N=4096 • 2 (3) Dimensionen	80	-	4,17min (57,66min)	-	-	• Fazit: leveled HE ohne Bootstrapping mit batching ist für die Analyse großer Daten z. B. Sensordaten geeignet
			• 4 (8) Dimensionen	1024	-	19,84min (159,97min)	-	-	
			• Lineare Regression N=4194304 • Kovarianz N=1048576 • 2 (3) Dimensionen	80	-	15,85min (47,33min)	-	-	
			• 4 (8) Dimensionen	1024	-	115,53min (284,57min)	-	-	
	Lepoint, Naehrig, 2014 [103]	• Intel i7 2600 3,4Ghz	SIMON-32/64	128	34	3h 27min	2048	6,06s	
Brakerski, 2012 [21]	Cheon et al., 2016 [38]	• Intel i5	FFT-Hadamard Produkt-IFFT (N=16)	-	-	3,21s	-	0,78ms	-
	Costache et al., 2016 [49]	• Intel Xeon E5	FFT-Hadamard Produkt-IFFT (N=16)	-	-	0,46s	-	-	-
Khedr et al, 2014, SHIELD, [100]	Gentry et al., 2015 [80]	• Intel i5 3320M	AES Verschlüsselung	-	-	4min 5s 6min 34s 17min 30s 27min 11s	-	2s / Block 3,3s/ Block 17,5s/ Block 27,1s/ Block	← Ohne Bootstrapping ← Ohne Bootstrapping ← mit Bootstrapping ← mit Bootstrapping

Tabelle 3.10 (Fortsetzung): Überblick FHE: Implementierungen, Laufzeiten und Besonderheiten. Basis Schema: implementiertes Schema, Implement.: Implementation und Testdurchführung, Plattform: Daten des verwendeten Computers, λ: Sicherheitsparameter, AND Tiefe: multiplikative Tiefe auf Bit Niveau, Gesamtzeit: Ergebnis der Messung, Anzahl paral. Enc: Anzahl paralleler Verschlüsselungen, Relative Zeit pro Block: benötigte Laufzeit pro Block einer Blockchiffre, Besonderheit: Besonderheiten der Implementierung. In Anlehnung an Quelle: [114].

Basis Schema	Implement.	Plattform	Schaltung	λ	AND Tiefe	Gesamtzeit	Anzahl paral. Enc	Relative Zeit pro Block	Besonderheit
López-Alt et al., 2012 [111]	Dorös et al., 2014 [61]	• Intel Xenon 2,9GHz	AES Verschlüsselung	80	40	27h 31h	2048	54s/ Block 55s/ Block	 ← Höheres Sicherheitsniveau
		• Intel i7 3770K 3,5GHz • 32GB Ram	Prince	130	30	57min	1024	3,3s	
	Dai et al., 2014 [54]	• Nvidia GTX690	• AES Verschlüsselung • Prince	-	-	4,15h 22min	-	7,3s/ Block 1,28s/ Block	-
	Öztürk et al., 2015 [124]	• Virtex 7	AES Encrypt	-	-	15min	-	439ms/ Block	-
YASHE, Bos et al., 2013 [19]	Lepoint, Naehrig, 2014 [103]	• Intel i7 2400	Simon-32/64	128	34	3min 20s 14min 9s	2048	200s/ Block 0,57s/ Block	← Verringerte Latenz ← Maximierte Datenrate

Tabelle 3.10 (Fortsetzung): Überblick FHE: Implementierungen, Laufzeiten und Besonderheiten. Basis Schema: implementiertes Schema, Implement.: Implementation und Testdurchführung, Plattform: Daten des verwendeten Computers, λ: Sicherheitsparameter, AND Tiefe: multiplikative Tiefe auf Bit Niveau, Gesamtzeit: Ergebnis der Messung, Anzahl paral. Enc: Anzahl paralleler Verschlüsselungen, Relative Zeit pro Block: benötigte Laufzeit pro Block einer Blockchiffre, Besonderheit: Besonderheiten der Implementierung. In Anlehnung an Quelle: [114].

3.7 Zweites Zwischenfazit

Wir haben uns in diesem Kapitel die Klasse der homomorphen Kryptosysteme genauer angesehen. Ihre Klassifikiationen sind sowohl in der englischen als auch in der deutschen Literatur uneinheitlich und daher verwirrend. Mithilfe von zwei vorgestellten Arbeiten zur Klassifikation haben wir das Begriffswirrwarr geordnet. Im Forschungsbereich ist eher die Klassifikation nach Armknecht et al. [8] verbreitet. Die Klassifikation nach Albrecht et al. [7] bezieht sich auf die praxistauglichen Kryptosysteme. Wir werden daher in den folgenden Kapiteln dieser wissenschaftlichen Arbeit die Klassifikation nach Armknecht et al. nutzen. Unser Beispielalgorithmus im praktischen Teil dieser Untersuchung ist ein LHE. Er kann durch Bootstrapping zu einem FHE werden und zählt zur Gruppe der LHE Algorithmen der zweiten Generation. Bislang ist kein erfolgreicher Angriff auf FV bekannt geworden. Als einer der wenigen Algorithmen ist er mit anderen LHEs verglichen worden. Wiese und Kollegen bezeichnen FV als einen der sichersten Algorithmen [159].

Bei unserer Betrachtung haben wir herausgearbeitet, dass das Ausmaß der Flexibilität dieser Systeme negativ mit ihren Leistungskosten korreliert. Damit sind die typischen langen Laufzeiten und hohen Speicherkosten gemeint. Ein kurzer Abriss der jungen Geschichte der HE zeigte rasante Verbesserungen. Zur Erleichterung der Auswahl eines passenden Systems präsentieren wir potentiellen Anwendern tabellarische Zusammenfassungen. Ihnen zu entnehmen ist auch, dass heute Anwendern noch empfohlen werden muss, möglichst ein PHE Algorithmus zu verwenden. Auch wenn sich die Laufzeitkosten der LHE und FHE im Zuge verbesserter Technik und Algorithmen stetig verringern, scheinen LHE nur in Ausnahmefällen für die praktische Arbeit geeignet. Angesprochen haben wir zudem, dass neben den kryptografischen Herausforderungen die Parameterwahl den Anwendern einige schwierige Entscheidungen abverlangt. Diese werden wir im Praxisteil an passender Stelle erläutern, nachdem wir im folgenden Kapitel unseren Beispielalgorithmus FV [67] detailliert betrachtet haben.

Kapitel 4

Fan und Vercauteren (FV) Verschlüsselung

Im Folgenden werden wir zuerst, basierend auf dem Grundlagenartikel, die Fan und Vercauteren (FV) Verschlüsselung [67] vorstellen, die im Jahr 2012 veröffentlicht wurde. Formeln, Beweisschemata und Erläuterungen wurden aus dieser Grundlagenarbeit übernommen und teilweise von uns ergänzt, da wir in unserer Literaturrecherche keine weiteren Arbeiten finden konnten, die den Algorithmus besser beschreiben. Bei der FV Verschlüsselung handelt es sich um ein nivelliert homomorphes Verschlüsselungsschema (LHE) das per Bootstrapping zu einem vollständig homomorphen Verschlüsselungsschema (FHE) zu Lasten der Laufzeit, erweitert werden kann (vgl. Kapitel 3.4). Die Zahlen, die verschlüsselt werden müssen, liegen als Polynome vor. Die Chiffretexte werden ebenfalls als Polynome repräsentiert. Zuerst werden wir die wichtigsten Grundlagen der Basisnotation, der benötigten Wahrscheinlichkeitsverteilung und des zugrunde liegenden RWLE Problems erklären. Basierend auf dem RLWE Problem, das wir in Kapitel 2.2.3 eingeführt haben, werden wir die FV Verschlüsselung herleiten.

4.1 (Basis) Notation

Basis der FV Verschlüsselung ist der Polynomring R, auf dem alle Rechenoperationen stattfinden.

$$R = \frac{\mathbb{Z}[x]}{f(x)} \ .$$

Dieser wird begrenzt durch ein normiertes, irreduzibles Polynom $f(x) \in \mathbb{Z}[x]$ mit Grad n. Das Polynom $f(x)$ heißt normiert, wenn für den höchsten Koeffizienten a_n (Leitkoeffizient) $a_n = 1$ gilt. Es ist irreduzibel, wenn sich das Polynom nicht als Produkt zweier nicht invertierbarer Polynome aufteilen lässt. D.h., das Polynom zerfällt nicht in einfachere Polynome.

Eine komplexe Zahl ζ ist eine m-te Einheitswurzel, wenn $\zeta^m = 1$ gilt und ist primitiv, wenn alle m-ten Einheitswurzeln als Potenz von ζ darstellbar sind. Für $1 \leq k \leq m$ ist ζ_m^k eine primitive m-te Einheitswurzel genau dann, wenn m und k teilerfremd sind. D.h., $ggT(m,k) = 1$.

$$\zeta_m^k = e^{\frac{2k\pi i}{m}} \ .$$

In der Praxis wird typischerweise ein Kreisteilungs- bzw. zyklotomisches Polynom $\Phi_m(x)$ als Begrenzung des Ringes genutzt. Dabei handelt es sich um das minimale Polynom der m-ten Einheitswurzel ζ.

$$\Phi_m(x) = \prod_{\substack{\zeta^m=1 \\ \zeta \text{ primitiv}}} (x - \zeta) = \prod_{\substack{1 \leq k \leq m \\ ggT(k,m)=1}} (x - e^{\frac{2k\pi i}{m}}) \ .$$

Alle Koeffizienten von $\Phi_m(x)$ sind ganzzahlig, sodass es sich um ein ganzzahliges, normiertes, irreduzibles Polynom handelt. Die bekannteste Wahl ist in Gleichung 4.1 zu sehen. n ist dabei eine Zweierpotenz.

$$\Phi_m(x) = x^n + 1 \ . \tag{4.1}$$

Der Grad von $\Phi_m(x)$ ist die Eulersche Phi Funktion $\varphi(m)$.

$$\varphi(m) = |\{a \in \mathbb{N} | 1 \leq a \leq m \wedge ggT(a,m) = 1\}| \ .$$

D.h., die Anzahl der zu m teilerfremden Zahlen, die kleiner sind als m. Die Elemente $\underline{\boldsymbol{a}}$ aus R sind Polynome mit Grad $n-1$ und werden im Folgenden zur besseren Lesbarkeit unterstrichen und fett gedruckt.

$$\underline{\boldsymbol{a}} = \sum_{i=0}^{n-1} a_i \cdot x^i \ .$$

Die Koeffizienten von $\underline{\boldsymbol{a}}$ bezeichnen wir als a_i. Sie sind, wie in der Definition unseres Ringes R beschrieben, ganzzahlig und werden durch den Grad $\varphi(m)$ des Kreisteilungspolynoms $\Phi_m(x)$ begrenzt. Die Unendlichkeitsnorm $\| \underline{\boldsymbol{a}} \|$ von $\underline{\boldsymbol{a}}$ ist der größte Koeffizient des Polynoms.

$$\| \underline{\boldsymbol{a}} \| = max_i |a_i| = max\{|a_i| : 0 \leq i \leq n-1\} \ .$$

Darauf aufbauend definieren wir den Ausdehnungsfaktor δ_R des Ringes R

$$\delta_R = max\{\frac{\| \underline{\boldsymbol{a}} \cdot \underline{\boldsymbol{b}} \|}{\| \underline{\boldsymbol{a}} \| \cdot \| \underline{\boldsymbol{b}} \|} : \underline{\boldsymbol{a}}, \underline{\boldsymbol{b}} \in R\} \ .$$

Der Ausdehnungsfaktor gibt an, wie stark sich die Unendlichkeitsnorm bzw. der größte Koeffizient bei einer Multiplikation maximal vergrößert. So gibt das Maximum der Menge über alle möglichen Polynome des Ringes die maximale Zuwachsrate an.

Mit $\mathbb{Z}_q$ beschreiben wir die Menge an ganzen Zahlen im Intervall $(-\frac{q}{2}, \frac{q}{2}]$, für $q > 1$. Diese Menge ist nicht zu verwechseln mit dem Ring $\frac{\mathbb{Z}}{q\mathbb{Z}}$. Aufbauend darauf definieren wir die Eingrenzung des Polynomrings R auf die Menge $\mathbb{Z}_q$. R_q beschreibt die Menge an Polynomen aus R mit Koeffizienten in $\mathbb{Z}_q$. In der FV Verteilung werden Polynome aus der Menge R_2 sowie R_q gezogen. $[a]_q$ ist eine eindeutige Zahl aus der Menge $\mathbb{Z}_q$, die über einem ungeraden Modulus gebildet wird, sodass für ein $a \in \mathbb{Z}$ durch $a \mod q$ eine Reduktion in das Intervall $(-\frac{q}{2}, \frac{q}{2}]$ stattfindet. $[\underline{\boldsymbol{a}}]_q$ ist die analoge Definition auf Polynomen. $[.]_q$ wird dabei auf die einzelnen Koeffizienten von $\underline{\boldsymbol{a}}$ angewendet und reduziert das Polynom in die Menge R_q. $r_q(a) = a \mod q$ bezeichnet die normale modulo Operation und damit eine Reduktion in das Intervall $[0, q)$.

Wir verwenden die gängigen Rundungsnotationen einer Zahl x für Aufrunden ($\lceil x \rceil$), Abrunden ($\lfloor x \rfloor$) und Runden zur nächstgelegenen Zahl ($\lfloor x \rceil$). $size(n)$ ist die Bit Größe einer ganzen Zahl n.

$$size(n) = \lceil log_2(n + 0.5) \rceil \ .$$

$n[i]$ ist das i-te Bit der Bit Schreibweise von $|n|$. Das erste Bit von n ist $n[0]$.

4.2 Fehlerverteilung

Wir schreiben $\underline{\boldsymbol{x}} \leftarrow \mathcal{D}$ oder $\underline{\boldsymbol{x}} \leftarrow S$, wenn ein Polynom $\underline{\boldsymbol{x}}$ aus einer Wahrscheinlichkeitsverteilung $\mathcal{D}$ oder gleichverteilt aus einer Menge S gezogen wird. Ein Polynom $\underline{\boldsymbol{p}} \in R$ ist beschränkt durch B, wenn der größte Koeffizient aus $\underline{\boldsymbol{p}}$ durch B beschränkt ist ($\| \underline{\boldsymbol{p}} \| \leq B$). Eine Verteilung χ ist durch B beschränkt, wenn alle Polynome $\underline{\boldsymbol{p}}$, die aus χ gezogen werden können, durch B beschränkt sind. D.h., jedes Polynom, das aus χ gezogen werden kann, hat Koeffizienten in $[-B, B]$.

Wir benötigen die diskrete Gauß Verteilung über ganze Zahlen $\mathcal{D}_{\mathbb{Z},\sigma}$. $\mathcal{D}_{\mathbb{Z},\sigma}$ weist jedem $x \in \mathbb{Z}$ eine Wahrscheinlichkeit proportional zu $exp(\frac{-\pi|x|^2}{\sigma^2})$ zu. Sie entspricht der diskreten Normalverteilung $\mathcal{N}(0, \sigma^2)$ mit Erwarungswert $\mu = 0$ und der Standardabweichung σ. Sei der Grundraum $\Omega = \mathbb{Z}$ und $\Sigma = \mathcal{P}(\mathbb{Z})$ die Menge der Ereignisse aller Potenzmengen auf den ganzen Zahlen. Dann ist die Wahrscheinlichkeit ein $x \in \mathbb{Z}$ zu ziehen gegeben durch das Wahrscheinlichkeitsmaß $P(x)$.

$$P(x) = \frac{p_\sigma(x)}{S_\sigma}$$

mit $p_\sigma(x) = exp(\frac{-(x-\mu)^2}{2 \cdot \sigma^2}) = exp(\frac{-x^2}{2 \cdot \sigma^2})$ und $S_\sigma = p_\sigma(\mathbb{Z}) = \sum_{k=-\infty}^{\infty} p_\sigma(k) \approx \sqrt{2\pi} \cdot \sigma$. Die Verteilung ist beschränkt durch B, wenn B groß genug gewählt wird. In der Praxis ist nach [67] z.B. $B = 10 \cdot \sigma$ eine gute Wahl. Die Standardabweichung σ ist einer der Parameter der FV Verschlüsselung. Die Verteilung χ, aus der wir unsere Polynome ziehen, wird nun über $\mathcal{D}_{\mathbb{Z},\sigma}$ definiert. In dem Fall der FV Verschlüsselung mit $\Phi_m(x) = x^n + 1$, wobei n eine

Zweierpotenz sein muss, werden die n Koeffizienten des zu ziehenden Polynoms aus $\mathcal{D}_{\mathbb{Z},\sigma}$ gezogen und es gilt:

$$\chi = \mathcal{D}^n_{\mathbb{Z},\sigma} \ .$$

Entsprechend der $\mathcal{D}_{\mathbb{Z},\sigma}$ Verteilung ist auch χ durch B begrenzt. Der Fall für allgemeinere Kreisteilungspolynome, der für die von uns genutzte FV Verschlüsselung unnötig ist, wird in [67] erläutert. Die Polynome, die aus der Verteilung gezogen werden, werden in FV als Fehler bzw. Rauschen in die Verschlüsselung und den Public-Key eingerechnet.

4.3 Ring learning with Errors (RLWE)

Das RLWE Problem ist die Abbildung des LWE Problems auf Ringe und bildet die Basis für die FV Verschlüsselung. Sei $R = \dfrac{\mathbb{Z}[x]}{\Phi_m(x)}$ der Polynomring und $\Phi_m(x)$ das Kreisteilungspolynom mit Grad $\varphi(m)$, welcher abhängt von dem Sicherheitsparameter λ. Sei $q = q(\lambda) \geq 2$ eine ganze Zahl und $\chi = \chi(\lambda)$ die Verteilung aus der wir Polynome ziehen können. Sei $\underline{\boldsymbol{s}}, \underline{\boldsymbol{a}} \in R_q$ gleichverteilt aus R_q gezogen und $\underline{\boldsymbol{e}} \sim \chi$ ein Polynom aus der χ Verteilung, dann ist die Einwegfunktion des RLWE Problems die Ausgabe:

$$(\underline{\boldsymbol{a}}, [\underline{\boldsymbol{a}} \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{e}}]_q) \ . \tag{4.2}$$

Statistisch ist die Verteilung von $(\underline{\boldsymbol{a}}, [\underline{\boldsymbol{a}} \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{e}}]_q)$ nicht auseinanderzuhalten von der Verteilung von $(\underline{\boldsymbol{a}}, \underline{\boldsymbol{b}})$, bei der beide Polynome gleichverteilt gezogen wurden. Anstatt $\underline{\boldsymbol{s}}$ gleichverteilt aus R_q zu ziehen, kann es nach [67] aus χ gezogen werden, ohne Sicherheitskonsequenzen in Kauf nehmen zu müssen. Die Schwierigkeit des Problems ist unabhängig von q.

4.4 FV

Im Folgenden werden wir das Verschlüsselungsschema, das 2012 von Fan und Vercauteren vorgestellt wurde [67], erläutern. Die Operationen der Schlüsselerzeugung wie auch der Ver- und Entschlüsselung wurden in der optimierten Version aus der LPR Verschlüsselung [113] übernommen. Die Optimierung besteht darin, $\underline{\boldsymbol{s}}$ und $\underline{\boldsymbol{u}}$ gleichverteilt aus R_2 zu ziehen, anstatt aus der χ Verteilung. Hinzu kommt die Erzeugung der Relinearisation-Keys, die wir später im Zuge der Relinearisierung nach einer Multiplikation erläutert werden. Des Weiteren beschreiben wir die Erweiterung um die Addition und Multiplikation auf den verschlüsselten Daten, basierend auf RLWE. Das FV Verschlüsselungsschema portiert das vollständig homomorphe (auf LWE basierende) Verschlüsselungsschema von Brakerski [21] auf das RLWE Problem, wie wir in Kapitel 3.5.2 bereits erläutert haben.

4.4.1 Secret-Key

Algorithmus 4.1 bekommt zur Erzeugung des Secret-Keys eine Bitfolge der Länge λ (1^λ) übergeben und zieht den Secret-Key gleichverteilt aus R_2.
Eingabe:1^λ
Ausgabe:sk (Secret-Key)

$\underline{\boldsymbol{s}} \leftarrow R_2$
return $sk = \underline{\boldsymbol{s}}$

Algorithmus 4.1: Secret-Key: SecretKeyGen(1^λ)

4.4.2 Public-Key

Der Public-Key Algorithmus wurde aus der LPR Verschlüsselung übernommen und besteht aus den zwei Polynomen des RLWE Problems (Gleichung 4.2). Hierzu werden in Algorithmus 4.2 zwei Polynome $\underline{\boldsymbol{a}} \in R_q$ und $\underline{\boldsymbol{e}} \sim \chi$ mit dem Secret-Key $\underline{\boldsymbol{s}} \sim \chi$, basierend auf dem RLWE Problem, verrechnet. Im Gegensatz zur Beschreibung in [113] wurde an dieser Stelle das erste Polynom negiert. Dies sorgt dafür, dass wir uns in dem Beweis des Entschlüsselungsalgorithmus nur noch Addition und Multiplikation und nicht mehr die Subtraktion betrachten müssen. Ein künstliches Rauschen $\underline{\boldsymbol{e}}$ wird auf $\underline{\boldsymbol{a}} \cdot \underline{\boldsymbol{s}}$ addiert, um die Rückschlüsse auf den Secret-Key $\underline{\boldsymbol{s}}$ zu verhindern, da $\underline{\boldsymbol{a}}$ aus dem zweiten Teil des Public-Keys bekannt ist.

Eingabe:sk (Sectret Key)
Ausgabe:pk (Public-Key)

$\underline{\boldsymbol{s}} = sk$
$\underline{\boldsymbol{a}} \leftarrow R_q$
$\underline{\boldsymbol{e}} \leftarrow \chi$
return $pk = ([-(\underline{\boldsymbol{a}} \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{e}})]_q, \underline{\boldsymbol{a}})$

Algorithmus 4.2: Public-Key: PublicKeyGen(sk)

4.4.3 Verschlüsselung

Sei R_t der Klartextraum mit $t \in \mathbb{Z}$ und $t > 1$. Der Klartextraum bestimmt, welche Form und Größe die zu verschlüsselnden Nachrichten (der Klartext) haben können. In diesem Fall handelt es sich um Polynome mit Koeffizienten im Intervall $(-\frac{t}{2}, \frac{t}{2}]$. Die Ver- und Entschlüsselung basiert auf dem Konzept, dass sich eine Zahl q darstellen lässt als:

$$q = \Delta \cdot t + r_t(q) \ .$$

Hierbei ist $\Delta = \lfloor \frac{q}{t} \rfloor$ die ganzzahligen Division und $r_t(q) = q \bmod t$ der Rest der ganzzahligen Division. q und t müssen dabei weder Primzahlen noch coprime sein.

Eingabe:pk (Public-Key), $\underline{\boldsymbol{m}} \in R_t$ (Klartext)
Ausgabe:ct (Chiffretext)

$\underline{\boldsymbol{p}}_0 = pk[0] = [-(\underline{\boldsymbol{a}} \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{e}})]_q$
$\underline{\boldsymbol{p}}_1 = pk[1] = \underline{\boldsymbol{a}}$
$\underline{\boldsymbol{u}} \leftarrow R_2$
$\underline{\boldsymbol{e}}_1, \underline{\boldsymbol{e}}_2 \leftarrow \chi$
$ct = ([\underline{\boldsymbol{p}}_0 \cdot \underline{\boldsymbol{u}} + \underline{\boldsymbol{e}}_1 + \Delta \cdot \underline{\boldsymbol{m}}]_q, [\underline{\boldsymbol{p}}_1 \cdot \underline{\boldsymbol{u}} + \underline{\boldsymbol{e}}_2]_q)$
return ct

Algorithmus 4.3: Verschlüsseln: Encrypt(pk,$\underline{\boldsymbol{m}}$)

Algorithmus 4.3 verschlüsselt eine übergebene Nachricht $\underline{\boldsymbol{m}}$. Der erzeugte Chiffretext ct besteht aus zwei Polynomen, wobei $\underline{\boldsymbol{m}}$ nur im ersten Polynom verrechnet wurde. Das zweite Polynom wird zur Entschlüsselung in Algorithmus 4.4 benötigt. Die beiden Polynome des Public-Keys $\underline{\boldsymbol{p}}_0, \underline{\boldsymbol{p}}_1 \in R_q$ werden jeweils mit einem Polynom $\underline{\boldsymbol{u}} \sim \chi$ multipliziert und ein kleines Rauschen $\underline{\boldsymbol{e}}_1, \underline{\boldsymbol{e}}_2 \sim \chi$ zur Vermeidung von Rückschlüssen aufaddiert. Die Nachricht $\underline{\boldsymbol{m}}$ wird nun über $\Delta \cdot \underline{\boldsymbol{m}} = \lfloor \frac{q}{t} \rfloor \cdot \underline{\boldsymbol{m}}$ auf das erste Polynom aufaddiert. Der Chiffretext kann als ein lineares Polynom, dessen Koeffizienten wieder Polynome sind, gesehen werden. Sprechen wir in den folgenden Kapiteln von Polynomen in Chiffretexten, sind damit die Koeffizienten des Chiffretextes, seine Polynome gemeint.

4.4.4 Entschlüsselung

In Algorithmus 4.4 wird nun der Chiffretext ct so mit dem Secret-Key $\underline{\boldsymbol{s}}$ verrechnet, dass bei korrektem Runden die Nachricht $\underline{\boldsymbol{m}}$ herauskommt.

Eingabe:sk (Secret-Key), ct (Chiffretext)
Ausgabe:$\underline{\boldsymbol{m}}$ (Klartext)

$\underline{\boldsymbol{s}} = sk$
$\underline{\boldsymbol{c}}_0 = ct[0]$
$\underline{\boldsymbol{c}}_1 = ct[1]$
return $\underline{\boldsymbol{m}} = [\lfloor \frac{t \cdot [\underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}}]_q}{q} \rceil]_t$

Algorithmus 4.4: Entschlüsseln: Decrypt(sk,ct)

Wenn wir die Polynome eines Chiffretextes ct als Koeffizienten eines linearen Polynoms $ct(x)$ interpretieren und dieses Polynom an der Stelle $\underline{\boldsymbol{s}}$ auswerten, erhalten wir unsere Hauptinvariante:

$$[ct(\underline{\boldsymbol{s}})]_q = [\underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}}]_q = \Delta \cdot \underline{\boldsymbol{m}} + \underline{\boldsymbol{v}} \; . \tag{4.3}$$

Das Polynom $\underline{\boldsymbol{v}}$ wird Rauschen genannt. Dieses Rauschen wird, wenn es klein genug ist, durch die Rundungsoperation $\lfloor . \rceil$ im Entschlüsselungsschritt wegfallen. Ist das Rauschen zu groß, schlägt das Runden fehl (vgl. Kapitel 2.4.4) und es ist nicht mehr möglich, den Chiffretext sinnvoll zu entschlüsseln.

Zum Beweis der Invariante in Gleichung 4.3 erhalten wir durch Einsetzen der Definition von $\underline{\boldsymbol{c}}_0 = [\underline{\boldsymbol{p}}_0 \cdot \underline{\boldsymbol{u}} + \underline{\boldsymbol{e}}_1 + \Delta \cdot \underline{\boldsymbol{m}}]_q$ und $\underline{\boldsymbol{c}}_1 = [\underline{\boldsymbol{p}}_1 \cdot \underline{\boldsymbol{u}} + \underline{\boldsymbol{e}}_2]_q$ aus Algorithmus 4.3:

$$\begin{aligned}[\underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}}]_q &= \underline{\boldsymbol{p}}_0 \cdot \underline{\boldsymbol{u}} + \underline{\boldsymbol{e}}_1 + \Delta \cdot \underline{\boldsymbol{m}} + \underline{\boldsymbol{p}}_1 \cdot \underline{\boldsymbol{u}} \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{e}}_2 \cdot \underline{\boldsymbol{s}} \mod q \\ &= \Delta \cdot \underline{\boldsymbol{m}} + \underline{\boldsymbol{p}}_0 \cdot \underline{\boldsymbol{u}} + \underline{\boldsymbol{p}}_1 \cdot \underline{\boldsymbol{u}} \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{e}}_1 + \underline{\boldsymbol{e}}_2 \cdot \underline{\boldsymbol{s}} \mod q \ .\end{aligned}$$

Betrachten wir kurz in einer Zwischenrechnung den Term $\underline{\boldsymbol{p}}_0 \cdot \underline{\boldsymbol{u}} + \underline{\boldsymbol{p}}_1 \cdot \underline{\boldsymbol{u}} \cdot \underline{\boldsymbol{s}}$, so erhalten wir durch das Einsetzen der Definition von $\underline{\boldsymbol{p}}_0 = [-(\underline{\boldsymbol{a}} \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{e}})]_q$ und $\underline{\boldsymbol{p}}_1 = \underline{\boldsymbol{a}}$ aus Algorithmus 4.2:

$$\begin{aligned}\underline{\boldsymbol{p}}_0 \cdot \underline{\boldsymbol{u}} + \underline{\boldsymbol{p}}_1 \cdot \underline{\boldsymbol{u}} \cdot \underline{\boldsymbol{s}} &= -(\underline{\boldsymbol{a}} \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{e}}) \cdot \underline{\boldsymbol{u}} + (\underline{\boldsymbol{a}}) \cdot \underline{\boldsymbol{u}} \cdot \underline{\boldsymbol{s}} \mod q \\ &= \underline{\boldsymbol{a}} \cdot \underline{\boldsymbol{u}} \cdot \underline{\boldsymbol{s}} - \underline{\boldsymbol{a}} \cdot \underline{\boldsymbol{s}} \cdot \underline{\boldsymbol{u}} - \underline{\boldsymbol{e}} \cdot \underline{\boldsymbol{u}} \mod q \\ &= -\underline{\boldsymbol{e}} \cdot \underline{\boldsymbol{u}} \mod q \ .\end{aligned}$$

D.h., es gilt:

$$[\underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}}]_q = \Delta \cdot \underline{\boldsymbol{m}} - \underline{\boldsymbol{e}} \cdot \underline{\boldsymbol{u}} + \underline{\boldsymbol{e}}_1 + \underline{\boldsymbol{e}}_2 \cdot \underline{\boldsymbol{s}} \mod q \ .$$

Um unsere Invariante aus Gleichung 4.3 aus dieser Gleichung herzuleiten, definieren wir $\boldsymbol{v}$ als

$$\underline{\boldsymbol{v}} = -\underline{\boldsymbol{e}} \cdot \underline{\boldsymbol{u}} + \underline{\boldsymbol{e}}_1 + \underline{\boldsymbol{e}}_2 \cdot \underline{\boldsymbol{s}}$$

womit direkt Gleichung 4.3 folgt:

$$[\underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}}]_q = \Delta \cdot \underline{\boldsymbol{m}} + \underline{\boldsymbol{v}} \mod q \ .$$

Damit haben wir die Hauptinvariante bewiesen.

■

Um nun zu zeigen, wann $\underline{\boldsymbol{m}} = [\lfloor \frac{t \cdot [\underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}}]_q}{q} \rceil]_t$ gilt und damit der Algorithmus den Chiffretext korrekt entschlüsselt, betrachten wir die jeweiligen Schranken für die Größe der Elemente des Rauschterms $\underline{\boldsymbol{v}}$. Da χ durch B begrenzt ist und $\underline{\boldsymbol{e}}, \underline{\boldsymbol{e}}_1, \underline{\boldsymbol{e}}_2 \sim \chi$, sind diese nach Definition durch B begrenzt und da $\|\underline{\boldsymbol{s}}\| = \|\underline{\boldsymbol{u}}\| = 1$ gilt, folgt als Begrenzung für $\underline{\boldsymbol{v}}$:

$$\|\underline{\boldsymbol{v}}\| \leq B \cdot (2 \cdot \delta_R + 1) \ .$$

Für jeweils eine Multiplikation lässt sich $B \cdot \delta_R$ als Begrenzung festlegen, da der Ausdehnungsfaktor δ_R bei jeder Multiplikation mitbetrachtet werden muss. Wegen der Addition von $\underline{\boldsymbol{e}}_1$ muss noch ein weiteres B zu der Begrenzung aufaddiert werden. In dem Grundlagenartikel [67] wird diese Begrenzung basierend auf dem fälschlicherweise hergeleiteten

Term $\underline{\boldsymbol{v}} = \underline{\boldsymbol{e}} \cdot \underline{\boldsymbol{u}} + \underline{\boldsymbol{e}}_1 + \underline{\boldsymbol{e}}_2 \cdot \underline{\boldsymbol{s}}$ gebildet. Die hergeleitete Begrenzung gilt dennoch für den durch uns korrekt ermittelten Term $\underline{\boldsymbol{v}} = -\underline{\boldsymbol{e}} \cdot \underline{\boldsymbol{u}} + \underline{\boldsymbol{e}}_1 + \underline{\boldsymbol{e}}_2 \cdot \underline{\boldsymbol{s}}$, da durch eine Subtraktion die Unendlichkeitsnorm $\|.\|$ nicht zunehmen kann.

Für ein beliebiges $\underline{\boldsymbol{r}} \in R_q$ gilt, da Vielfache von q durch modulo q wegfallen:

$$\underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}} = \Delta \cdot \underline{\boldsymbol{m}} + \underline{\boldsymbol{v}} + q \cdot \underline{\boldsymbol{r}} \ . \tag{4.4}$$

Um nun zu zeigen, wann bei der Entschlüsselung korrekt gerundet wird, multiplizieren wir Gleichung 4.4 analog zum Entschlüsselungsalgorithmus mit $\frac{t}{q}$ und erhalten durch Einsetzen der Definition von $\Delta = \lfloor \frac{q}{t} \rfloor$:

$$\begin{aligned} \frac{t}{q} \cdot (\underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}}) &= \frac{t}{q} \cdot (\lfloor \frac{q}{t} \rfloor \cdot \underline{\boldsymbol{m}} + \underline{\boldsymbol{v}} + q \cdot \underline{\boldsymbol{r}}) \\ &= \frac{t}{q} \cdot (\frac{q}{t} \cdot \underline{\boldsymbol{m}} - \epsilon \cdot \underline{\boldsymbol{m}} + \underline{\boldsymbol{v}} + q \cdot \underline{\boldsymbol{r}}) \\ &= \underline{\boldsymbol{m}} + \frac{t}{q} \cdot (\underline{\boldsymbol{v}} - \epsilon \cdot \underline{\boldsymbol{m}}) + t \cdot \underline{\boldsymbol{r}} \ . \end{aligned} \tag{4.5}$$

$\epsilon \cdot \underline{\boldsymbol{m}}$ ist der Rest der ganzzahligen Division $\Delta \cdot \underline{\boldsymbol{m}}$, der übrig bleibt, wenn wir Δ bei der Multiplikation mit $\frac{t}{q}$ wegkürzen. Es gilt:

$$\epsilon = \frac{q}{t} - \Delta = \frac{q}{t} - \lfloor \frac{q}{t} \rfloor = q \mod t = r_t(q) \ . \tag{4.6}$$

Aus Gleichung 4.6 folgt direkt $\epsilon < 1$. Damit der Term $\frac{t}{q} \cdot (\underline{\boldsymbol{v}} - \epsilon \cdot \underline{\boldsymbol{m}})$ aus Gleichung 4.5 bei der Rundungsoperation wegfällt und somit korrekt gerundet wird, muss nach [67] gelten:

$$\frac{t}{q} \cdot \|\underline{\boldsymbol{v}} - \epsilon \cdot \underline{\boldsymbol{m}}\| < \frac{1}{2} \ .$$

An dieser Begrenzung lässt sich gut erkennen, dass sich ein Chiffretext bei zu großem Rauschen nicht mehr sinnvoll entschlüsseln lässt, da das Rauschen nicht mehr herausgekürzt werden kann. Zuletzt wird bei der Entschlüsselung der gekürzte Term Modulo t berechnet, wodurch $t \cdot \underline{\boldsymbol{r}}$ aus Gleichung 4.5 wegfällt. $\underline{\boldsymbol{m}}$ bleibt dabei vollständig erhalten, da $\underline{\boldsymbol{m}} \in R_t$ gilt. Womit bewiesen ist, dass bei einem Rauschen, das klein genug ist, die Nachricht $\underline{\boldsymbol{m}}$ vollständig entschlüsselt werden kann.

4.4.5 Addition

Seien ct_i für $i = 1, 2$ zwei Chiffretexte mit $[ct_i(\underline{\boldsymbol{s}})]_q = \Delta \cdot \underline{\boldsymbol{m}}_i + \underline{\boldsymbol{v}}_i$, dann gilt:

$$[ct_1(s) + ct_2(s)]_q = \Delta \cdot \underline{\boldsymbol{m}}_1 + \Delta \cdot \underline{\boldsymbol{m}}_2 + \underline{\boldsymbol{v}}_1 + \underline{\boldsymbol{v}}_2 \mod q \ .$$

Da $\underline{\boldsymbol{m}}_1 + \underline{\boldsymbol{m}}_2$ weiterhin in dem Klartextraum R_t liegen muss, ist es nötig, die Addition modulo t einzuschränken, sodass gilt:

$$\underline{\boldsymbol{m}}_1 + \underline{\boldsymbol{m}}_2 = [\underline{\boldsymbol{m}}_1 + \underline{\boldsymbol{m}}_2]_t + t \cdot \underline{\boldsymbol{r}} \ .$$

Analog zur Herleitung im vorherigen Abschnitt, erhalten wir durch Einsetzen der Definition von Δ und Umstellen:

$$\begin{aligned}[ct_1(s) + ct_2(s)]_q &= \Delta \cdot ([\underline{\boldsymbol{m}}_1 + \underline{\boldsymbol{m}}_2]_t + t \cdot \underline{\boldsymbol{r}}) + \underline{\boldsymbol{v}}_1 + \underline{\boldsymbol{v}}_2 \mod q \\ &= \Delta \cdot [\underline{\boldsymbol{m}}_1 + \underline{\boldsymbol{m}}_2]_t + \Delta \cdot t \cdot \underline{\boldsymbol{r}} + \underline{\boldsymbol{v}}_1 + \underline{\boldsymbol{v}}_2 \mod q \\ &= \Delta \cdot [\underline{\boldsymbol{m}}_1 + \underline{\boldsymbol{m}}_2]_t + \frac{q}{t} \cdot t \cdot \underline{\boldsymbol{r}} - \epsilon \cdot t \cdot \underline{\boldsymbol{r}} + \underline{\boldsymbol{v}}_1 + \underline{\boldsymbol{v}}_2 \mod q \\ &= \Delta \cdot [\underline{\boldsymbol{m}}_1 + \underline{\boldsymbol{m}}_2]_t + q \cdot \underline{\boldsymbol{r}} - \epsilon \cdot t \cdot \underline{\boldsymbol{r}} + \underline{\boldsymbol{v}}_1 + \underline{\boldsymbol{v}}_2 \mod q \ .\end{aligned}$$

Durch die Reduktion modulo q fällt an dieser Stelle $q \cdot \underline{\boldsymbol{r}}$ weg und es gilt:

$$[ct_1(s) + ct_2(s)]_q = \Delta \cdot [\underline{\boldsymbol{m}}_1 + \underline{\boldsymbol{m}}_2]_t + \underline{\boldsymbol{v}}_1 + \underline{\boldsymbol{v}}_2 - \epsilon \cdot t \cdot \underline{\boldsymbol{r}}$$

mit $\epsilon = \frac{q}{t} - \Delta = r_t(q) < 1$. Das Rauschen steigt bei der Addition additiv, um maximal t an, da $\|\underline{\boldsymbol{r}}\| \leq 1$ gilt.

Nach [67] können wir daher die Addition einfach wie in Algorithmus 4.5 definieren.

Eingabe:ct_1, ct_2 (Chiffretext)
Ausgabe:ct (Verschlüsseltes Ergebnis der Addition)
$ct = ([ct_1[0] + ct_2[0]]_q, [ct_1[1] + ct_2[1]]_q)$
return ct

Algorithmus 4.5: Addition: Add(ct_1,ct_2)

4.4.6 Multiplikation

Die Multiplikation in der FV Verschlüsselung teilt sich in zwei Teile. Die Basis Multiplikation, bei der sich der Chiffretext auf ein Polynom mit Grad 2 verlängert und die anschließende Relinearisierung, die das Polynom wieder auf Grad 1 und damit zu einem linearen Polynom reduziert. Für die Herleitung führen wir an dieser Stelle den Algorithmus für die Multiplikation in Algorithmus 4.6 ein. Wir können gut erkennen, dass der entstandene Chiffretext nach der Multiplikation aus drei Polynomen besteht.

Ausgehend von unserer Invariante in Gleichung 4.3 gilt für einen Chiffretext ohne modulo Operator:

$$ct_i(\underline{\boldsymbol{s}}) = \Delta \underline{\boldsymbol{m}}_i + \underline{\boldsymbol{v}}_i + q \cdot \underline{\boldsymbol{r}}_i \ . \tag{4.7}$$

Nach [67] gilt $\|\underline{\boldsymbol{r}}_i\| = \delta_R \cdot \|\underline{\boldsymbol{s}}\|$. Durch Einsetzen und Ausmultiplizieren der Invariante in Gleichung 4.7 erhalten wir:

$$\begin{aligned}(ct_1 \cdot ct_2)(\underline{\boldsymbol{s}}) =& \Delta^2 \cdot \underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{m}}_2 + \Delta \cdot (\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{v}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{v}}_1) + q \cdot (\underline{\boldsymbol{v}}_1 \cdot \underline{\boldsymbol{r}}_2 + \underline{\boldsymbol{v}}_2 \cdot \underline{\boldsymbol{r}}_1) \\ &+ \underline{\boldsymbol{v}}_1 \cdot \underline{\boldsymbol{v}}_2 + q \cdot \Delta \cdot (\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{r}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{r}}_1) + q^2 \cdot \underline{\boldsymbol{r}}_1 \cdot \underline{\boldsymbol{r}}_2 \ .\end{aligned} \tag{4.8}$$

Damit dieser Term entschlüsselt werden kann, müsste er mit $\frac{1}{\Delta}$ skaliert werden. Da Δ nicht zwingend q teilt, könnte dies durch den hohen Rundungsfehler im letzten Term zu einem

Eingabe:ct_1, ct_2 (Chiffretext), rlk (Relinearisation Key)
Ausgabe:ct (Verschlüsseltes Ergebnis Multiplikation)

$$\underline{\boldsymbol{c}}_0 = [\lfloor \frac{t \cdot (ct_1[0] \cdot ct_2[0])}{q} \rceil]_q$$
$$\underline{\boldsymbol{c}}_1 = [\lfloor \frac{t \cdot (ct_1[0] \cdot ct_2[1] + ct_1[1] \cdot ct_2[0])}{q} \rceil]_q$$
$$\underline{\boldsymbol{c}}_2 = [\lfloor \frac{t \cdot (ct_1[1] \cdot ct_2[1])}{q} \rceil]_q$$
$$ct = [\underline{\boldsymbol{c}}_0, \underline{\boldsymbol{c}}_1, \underline{\boldsymbol{c}}_2]$$
return ct

Algorithmus 4.6: Multiplikation: Mul(ct_1,ct_2, rlk)

starken Rauschen führen. Stattdessen skalieren wir mit $\frac{t}{q}$, was die Rundungsproblematik beseitigt. Sei $ct_1(x) \cdot ct_2(x) = \underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot x + \underline{\boldsymbol{c}}_2 \cdot x^2$, dann nutzen wir für die Skalierung folgende Approximation:

$$\frac{t}{q} \cdot (ct_1 \cdot ct_2)(\underline{\boldsymbol{s}}) = \lfloor t \cdot \frac{\underline{\boldsymbol{c}}_0}{q} \rceil + \lfloor t \cdot \frac{\underline{\boldsymbol{c}}_1}{q} \rceil \cdot \underline{\boldsymbol{s}} + \lfloor t \cdot \frac{\underline{\boldsymbol{c}}_2}{q} \rceil \cdot \underline{\boldsymbol{s}}^2 + \underline{\boldsymbol{r}}_a \ . \tag{4.9}$$

Neu hinzu kommt der Approximationsfehler $\underline{\boldsymbol{r}}_a$, für den nach [67] $\|\underline{\boldsymbol{r}}_a\| < \frac{(\delta_R \cdot \|\underline{\boldsymbol{s}}\| + 1)^2}{2}$ gilt. Wir schreiben analog zum vorherigen Abschnitt:

$$\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{m}}_2 = [\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{m}}_2]_t + t \cdot \underline{\boldsymbol{r}}_m$$
$$\underline{\boldsymbol{v}}_1 \cdot \underline{\boldsymbol{v}}_2 = [\underline{\boldsymbol{v}}_1 \cdot \underline{\boldsymbol{v}}_2]_\Delta + \Delta \cdot \underline{\boldsymbol{r}}_v \ .$$

Der Approximationsfehler ist hierbei nach [67] $\underline{\boldsymbol{r}}_m < \frac{t \cdot \delta_R}{4}$ und $\underline{\boldsymbol{r}}_v < \frac{E^2 \cdot \delta_R}{\Delta}$, wobei E eine Begrenzung des originalen Rauschterms $\underline{\boldsymbol{v}}_i$ ist mit $\|\underline{\boldsymbol{v}}_i\| < E$. Nach dieser Abschätzung des Rausch- und Fehlerverhaltens erhalten wir durch Einsetzen in Gleichung 4.8:

$$\begin{aligned}(ct_1 \cdot ct_2)(\underline{\boldsymbol{s}}) =& \Delta^2 \cdot ([\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{m}}_2]_t + t \cdot \underline{\boldsymbol{r}}_m) + \Delta \cdot (\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{v}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{v}}_1) + q \cdot (\underline{\boldsymbol{v}}_1 \cdot \underline{\boldsymbol{r}}_2 + \underline{\boldsymbol{v}}_2 \cdot \underline{\boldsymbol{r}}_1) \\ &+ [\underline{\boldsymbol{v}}_1 \cdot \underline{\boldsymbol{v}}_2]_\Delta + \Delta \cdot \underline{\boldsymbol{r}}_v + q \cdot \Delta \cdot (\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{r}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{r}}_1) + q^2 \cdot \underline{\boldsymbol{r}}_1 \cdot \underline{\boldsymbol{r}}_2 \ .\end{aligned} \tag{4.10}$$

Multiplizieren wir Gleichung 4.10 nun mit $\frac{t}{q}$, um den Entschlüsselungsschritt nachzuvollziehen, erhalten wir durch Umstellen der Terme und Ausnutzen von $t \cdot \Delta = q - r_t(q)$ Gleichung 4.11.

$$\begin{aligned}\frac{t \cdot (ct_1 \cdot ct_2)(\underline{\boldsymbol{s}})}{q} =& (\frac{q - r_t(q)}{q}) \cdot \Delta \cdot ([\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{m}}_2]_t + t \cdot \underline{\boldsymbol{r}}_m) + (\frac{q - r_t(q)}{q}) \cdot (\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{v}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{v}}_1) \\ &+ t \cdot (\underline{\boldsymbol{v}}_1 \cdot \underline{\boldsymbol{r}}_2 + \underline{\boldsymbol{v}}_2 \cdot \underline{\boldsymbol{r}}_1) + \frac{t}{q} \cdot [\underline{\boldsymbol{v}}_1 \cdot \underline{\boldsymbol{v}}_2]_\Delta + (\frac{q - r_t(q)}{q}) \cdot \underline{\boldsymbol{r}}_v \\ &+ (q - r_t(q)) \cdot (\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{r}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{r}}_1) + t \cdot q \cdot \underline{\boldsymbol{r}}_1 \cdot \underline{\boldsymbol{r}}_2 \ .\end{aligned} \tag{4.11}$$

Durch Ausnutzen von $\frac{q-r_t(q)}{q} = 1 - \frac{r_t(q)}{q}$ und Umstellen erhalten wir:

$$\begin{aligned}\frac{t \cdot (ct_1 \cdot ct_2)(\underline{\boldsymbol{s}})}{q} =& \Delta \cdot [\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{m}}_2]_t + (\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{v}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{v}}_1) + t \cdot (\underline{\boldsymbol{v}}_1 \cdot \underline{\boldsymbol{r}}_2 + \underline{\boldsymbol{v}}_2 \cdot \underline{\boldsymbol{r}}_1) \\ &+ \underline{\boldsymbol{r}}_v + (q - r_t(q)) \cdot (\underline{\boldsymbol{r}}_m + \underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{r}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{r}}_1) + q \cdot t \cdot \underline{\boldsymbol{r}}_1 \cdot \underline{\boldsymbol{r}}_2 \\ &+ \frac{t}{q} \cdot [\underline{\boldsymbol{v}}_1 \cdot \underline{\boldsymbol{v}}_2]_\Delta - \frac{r_t(q)}{q} \cdot (\Delta \cdot [\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{m}}_2]_t + (\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{v}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{v}}_1) + \underline{\boldsymbol{r}}_v) \ .\end{aligned} \tag{4.12}$$

So ist gut erkennbar, was nach Reduktion modulo q verschwindet und was vom Runden betroffen ist. Das Runden betrifft nur die letzte Zeile, da alle anderen Terme der Gleichung ganzzahlig sind. Diese Zeile bezeichnen wir im Folgenden als $\underline{\boldsymbol{r}}_r$ und es gilt nach [67]:

$$\|\underline{\boldsymbol{r}}_r\| < \delta_R \cdot (t + \frac{1}{2})^2 + \frac{1}{2} \ .$$

Nach einer Reduktion modulo q fallen die Terme $q \cdot (\underline{\boldsymbol{r}}_m + \underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{r}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{r}}_1)$ und $q \cdot t \cdot \underline{\boldsymbol{r}}_1 \cdot \underline{\boldsymbol{r}}_2$ als Vielfache von q weg. Durch die Reduktion modulo q und Substitution $\underline{\boldsymbol{r}}_r$ der restlichen Terme in der letzten Zeile aus Gleichung 4.12 erhalten wir durch Einsetzen in Gleichung 4.9:

$$\begin{aligned}[\lfloor t \cdot \frac{\underline{\boldsymbol{c}}_0}{q} \rceil + \lfloor t \cdot \frac{\underline{\boldsymbol{c}}_1}{q} \rceil \cdot \underline{\boldsymbol{s}} + \lfloor t \cdot \frac{\underline{\boldsymbol{c}}_2}{q} \rceil \cdot \underline{\boldsymbol{s}}^2]_q =& \Delta \cdot [\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{m}}_2]_t + (\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{v}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{v}}_1) \\ &+ t \cdot (\underline{\boldsymbol{v}}_1 \cdot \underline{\boldsymbol{r}}_2 + \underline{\boldsymbol{v}}_2 \cdot \underline{\boldsymbol{r}}_1) + \underline{\boldsymbol{r}}_v \\ &- r_t(q) \cdot (\underline{\boldsymbol{r}}_m + \underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{r}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{r}}_1) + \lfloor \underline{\boldsymbol{r}}_r - \underline{\boldsymbol{r}}_a \rceil \ .\end{aligned}$$

Setzen wir $\underline{\boldsymbol{v}}_3 = (\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{v}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{v}}_1) + t \cdot (\underline{\boldsymbol{v}}_1 \cdot \underline{\boldsymbol{r}}_2 + \underline{\boldsymbol{v}}_2 \cdot \underline{\boldsymbol{r}}_1) + \underline{\boldsymbol{r}}_v - r_t(q) \cdot (\underline{\boldsymbol{r}}_m + \underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{r}}_2 + \underline{\boldsymbol{m}}_2 \cdot \underline{\boldsymbol{r}}_1) + \lfloor \underline{\boldsymbol{r}}_r - \underline{\boldsymbol{r}}_a \rceil$, so erhalten wir Gleichung 4.13.

$$[\lfloor t \cdot \frac{\underline{\boldsymbol{c}}_0}{q} \rceil + \lfloor t \cdot \frac{\underline{\boldsymbol{c}}_1}{q} \rceil \cdot \underline{\boldsymbol{s}} + \lfloor t \cdot \frac{\underline{\boldsymbol{c}}_2}{q} \rceil \cdot \underline{\boldsymbol{s}}^2]_q = \Delta \cdot [\underline{\boldsymbol{m}}_1 \cdot \underline{\boldsymbol{m}}_2]_t + \underline{\boldsymbol{v}}_3 \tag{4.13}$$

mit einer Begrenzung nach [67]:

$$\|\underline{\boldsymbol{v}}_3\| < 2 \cdot \delta_R \cdot t \cdot E \cdot (\delta_R \cdot \|\underline{\boldsymbol{s}}\| + 1) + 2 \cdot t^2 \cdot \delta_R^2 \cdot (\|\underline{\boldsymbol{s}}\| + 1)^2 \ .$$

Diese Begrenzung zeigt nach [67], dass das Rauschen durch die Multiplikation nicht quadratisch zunimmt, sondern etwa um den Faktor $2 \cdot t \cdot \delta_R^2 \cdot \|\underline{\boldsymbol{s}}\|$. Es lässt sich gut erkennen, dass nicht nur t, sondern insbesondere auch die Norm des Secret-Keys einen Einfluss auf die Rauschzunahme haben. Da durch unsere Optimierung $\|\underline{\boldsymbol{s}}\| = 1$ gewählt wurde, wird der Rauschzuwachs durch diese Optimierung limitiert.

4.4.7 Relinearisierung

Als Resultat der Multiplikationen wird der Chiffretext ct um ein Polynom verlängert. Die Relinearisierung korrigiert dieses Problem und bildet den Chiffretext $ct = [\underline{\boldsymbol{c}}_0, \underline{\boldsymbol{c}}_1, \underline{\boldsymbol{c}}_2]$ mit Grad 2 wieder auf einen Chiffretext $ct' = (\underline{\boldsymbol{c}}'_0, \underline{\boldsymbol{c}}'_1)$ mit Grad 1 ab, sodass gilt:

$$[\underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{c}}_2 \cdot \underline{\boldsymbol{s}}^2]_q = [\underline{\boldsymbol{c}}'_0 + \underline{\boldsymbol{c}}'_1 \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{r}}]_q \ . \tag{4.14}$$

$\|\underline{\boldsymbol{r}}\|$ ist hierbei klein. Zu diesem Zweck stellen wir an dieser Stelle den Relinearisation-Key (rlk) vor. Da $\underline{\boldsymbol{s}}^2$ unbekannt ist, wäre die erste Idee, $\underline{\boldsymbol{s}}^2$ analog zu dem Public-Key mithilfe eines Rauschens unkenntlich zu machen und als Relinearisation-Key zur Verfügung zu stellen.

Eingabe:sk (Sectret Key)
Ausgabe:rlk

$\underline{\boldsymbol{a}}_0 \leftarrow R_q$
$\underline{\boldsymbol{e}}_0 \leftarrow \chi$
$rlk = [([-(\underline{\boldsymbol{a}}_0 \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{e}}_0) + \underline{\boldsymbol{s}}^2]_q, \underline{\boldsymbol{a}}_0)$
return rlk

Algorithmus 4.7: Erste Idee: EvaluateKeyGen(sk)

Es gilt bei diesem Ansatz $rlk[0] + rlk[1] \cdot \underline{\boldsymbol{s}} = \underline{\boldsymbol{s}}^2 + \underline{\boldsymbol{e}}_0$. Da aber $\underline{\boldsymbol{c}}_2$ ein zufälliges Element aus R_q ist, würde das Rauschen $\underline{\boldsymbol{e}}_0$ zu sehr vergrößert, was zu einer schlechten Approximation von $\underline{\boldsymbol{c}}_2 \cdot \underline{\boldsymbol{s}}$ und so auch zu einem hohen Fehler $\underline{\boldsymbol{r}}$ führt. Um dieses Problem zu umgehen, wurden in der FV Verschlüsselung zwei verschiedene Herangehensweisen vorgestellt. In der ersten Version der Relinearisierung wird $\underline{\boldsymbol{c}}_2$ in Teile kleinerer Norm unterteilt. Die zweite Version ähnelt einer Form des "modulus switching" (vgl. Kapitel 3.5.2).

Version 1: Um $\underline{\boldsymbol{c}}_2$ in Teile kleiner Norm zu unterteilen, wird eine Basis T gewählt und $\underline{\boldsymbol{c}}_2$ als Summe dieser Unterteilung in Abhängigkeit von T auszudrücken, sodass gilt:

$$\underline{\boldsymbol{c}}_2 = \sum_{i=0}^{l} T^i \cdot \underline{\boldsymbol{c}}_2^{(i)} \mod q \ . \tag{4.15}$$

Die Summe ist nach [67] begrenzt durch $l = \lfloor \log_T(q) \rfloor$ und die Koeffizienten von $\underline{\boldsymbol{c}}_2^{(i)}$ sind in R_T, d.h., in $\{-\frac{T}{2}, \frac{T}{2}\}$. Zwischen der Basis T und dem Klartext Modulus t besteht trotz der Begriffsähnlichkeit keinerlei Abhängigkeit. Basierend auf Algorithmus 4.8 wird nun $T^i \cdot \underline{\boldsymbol{s}}^2$ für $i = 0, .., l$ im Relinearisation-Key verrechnet und damit unkenntlich gemacht. Die Länge des Relinearisation-Keys ist nach [67] gegeben durch $l + 1 \simeq \log_T(q)$.

Da $T^i \cdot \underline{\boldsymbol{s}}^2$ weder als verschlüsselte Form im Relinearisation-Key noch als wirkliche Stichprobe aus der RLWE Verteilung vorliegt, müssen wir zusätzlich annehmen, dass das Verschlüsselungsschema sicher ist, wenn der Relinearisation-Key bekannt ist. D.h., dass durch den Relinearisation-Key keine Rückschlüsse auf die verschlüsselten Inhalte oder den Secret-Key möglich sind.

Die Relinearisierung selbst wird durch Algorithmus 4.9 durchgeführt. Die Anzahl an Multiplikationen bei der Relinearisierung liegt nach [67] bei $2 \cdot l \simeq 2 \cdot \log_T(q)$.

Eingabe:sk (Sectret Key), T
Ausgabe:rlk

$l = \lfloor \log_T(q) \rfloor$
for $i = 0, .., l$ **do**
$\underline{\boldsymbol{a}}_i \leftarrow R_q$
$\underline{\boldsymbol{e}}_i \leftarrow \chi$
end for
$rlk = [([-(\underline{\boldsymbol{a}}_i \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{e}}_i) + T^i \cdot \underline{\boldsymbol{s}}^2]_q, \underline{\boldsymbol{a}}_i) : i \in [0..l]]$
return rlk

Algorithmus 4.8: Relinearisation-Key v1: EvaluateKeyGen(sk, T)

Eingabe: $ct = [\underline{\boldsymbol{c}}_0, \underline{\boldsymbol{c}}_1, \underline{\boldsymbol{c}}_2]$ (Chiffretext), rlk (Relinearisation-Key)
Ausgabe:ct

$\underline{\boldsymbol{c}}'_0 = [\underline{\boldsymbol{c}}_0 + \sum_{i=0}^{l} rlk[i][0] \cdot \underline{\boldsymbol{c}}_2^{(i)}]_q$
$\underline{\boldsymbol{c}}'_1 = [\underline{\boldsymbol{c}}_1 + \sum_{i=0}^{l} rlk[i][1] \cdot \underline{\boldsymbol{c}}_2^{(i)}]_q$
$ct = (\underline{\boldsymbol{c}}'_0, \underline{\boldsymbol{c}}'_1)$
return ct

Algorithmus 4.9: Relinearisierung v1: FV.SH.Relin(ct, rlk)

Um zu beweisen, dass die Relinearisierung das geforderte Ergebnis liefert, setzen wir die Definition des entstandenen Chiffretextes aus Algorithmus 4.8 in Gleichung 4.14 ein.

$$\underline{\boldsymbol{c}}'_0 + \underline{\boldsymbol{c}}'_1 \cdot \underline{\boldsymbol{s}} = (\underline{\boldsymbol{c}}_0 + \sum_{i=0}^{l} rlk[i][0] \cdot \underline{\boldsymbol{c}}_2^{(i)}) + (\underline{\boldsymbol{c}}_1 + \sum_{i=0}^{l} rlk[i][1] \cdot \underline{\boldsymbol{c}}_2^{(i)}) \cdot \underline{\boldsymbol{s}} \mod q \ .$$

Durch Umstellen der Gleichung und Einsetzen der Definition des Relinearisation-Keys rlk $= [([-(\underline{\boldsymbol{a}}_i \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{e}}_i) + T^i \cdot \underline{\boldsymbol{s}}^2]_q, \underline{\boldsymbol{a}}_i) : i \in [0..l]]$ erhalten wir:

$$\begin{aligned}
\underline{\boldsymbol{c}}'_0 + \underline{\boldsymbol{c}}'_1 \cdot \underline{\boldsymbol{s}} &= \underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}} + \sum_{i=0}^{l} (-(\underline{\boldsymbol{a}}_i \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{e}}_i) + T^i \cdot \underline{\boldsymbol{s}}^2) \cdot \underline{\boldsymbol{c}}_2^{(i)} + \sum_{i=0}^{l} \underline{\boldsymbol{a}}_i \cdot \underline{\boldsymbol{c}}_2^{(i)} \cdot \underline{\boldsymbol{s}} \mod q \\
&= \underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}} + \sum_{i=0}^{l} -\underline{\boldsymbol{a}}_i \cdot \underline{\boldsymbol{s}} \cdot \underline{\boldsymbol{c}}_2^{(i)} - \underline{\boldsymbol{e}}_i \cdot \underline{\boldsymbol{c}}_2^{(i)} + T^i \cdot \underline{\boldsymbol{s}}^2 \cdot \underline{\boldsymbol{c}}_2^{(i)} + \sum_{i=0}^{l} \underline{\boldsymbol{a}}_i \cdot \underline{\boldsymbol{c}}_2^{(i)} \cdot \underline{\boldsymbol{s}} \mod q \\
&= \underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}} + \sum_{i=0}^{l} \underline{\boldsymbol{a}}_i \cdot \underline{\boldsymbol{c}}_2^{(i)} \cdot \underline{\boldsymbol{s}} - \sum_{i=0}^{l} \underline{\boldsymbol{a}}_i \cdot \underline{\boldsymbol{c}}_2^{(i)} \cdot \underline{\boldsymbol{s}} - \sum_{i=0}^{l} \underline{\boldsymbol{e}}_i \cdot \underline{\boldsymbol{c}}_2^{(i)} + \sum_{i=0}^{l} T^i \cdot \underline{\boldsymbol{s}}^2 \cdot \underline{\boldsymbol{c}}_2^{(i)} \mod q \\
&= \underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}} - \sum_{i=0}^{l} \underline{\boldsymbol{e}}_i \cdot \underline{\boldsymbol{c}}_2^{(i)} + \sum_{i=0}^{l} T^i \cdot \underline{\boldsymbol{s}}^2 \cdot \underline{\boldsymbol{c}}_2^{(i)} \mod q \\
&= \underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}} - \sum_{i=0}^{l} \underline{\boldsymbol{e}}_i \cdot \underline{\boldsymbol{c}}_2^{(i)} + \underline{\boldsymbol{s}}^2 \cdot \sum_{i=0}^{l} T^i \cdot \underline{\boldsymbol{c}}_2^{(i)} \mod q \ .
\end{aligned}$$

Da nach Gleichung 4.15 gilt $\underline{\boldsymbol{c}}_2 = \sum_{i=0}^{l} T^i \cdot \underline{\boldsymbol{c}}_2^{(i)} \mod q$ folgt:

$$= \underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}} - \sum_{i=0}^{l} \underline{\boldsymbol{e}}_i \cdot \underline{\boldsymbol{c}}_2^{(i)} + \underline{\boldsymbol{s}}^2 \cdot \underline{\boldsymbol{c}}_2 \mod q$$

$$= \underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_1 \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{c}}_2 \cdot \underline{\boldsymbol{s}}^2 - \sum_{i=0}^{l} \underline{\boldsymbol{e}}_i \cdot \underline{\boldsymbol{c}}_2^{(i)} \mod q \ .$$

Durch Anwenden von $[.]_q$ auf beiden Seiten können wir $\underline{\boldsymbol{r}} = \sum_{i=0}^{l} \underline{\boldsymbol{e}}_i \cdot \underline{\boldsymbol{c}}_2^{(i)}$ setzen. Es folgt direkt die Korrektheit von Gleichung 4.14.

■

Das Rauschen, das in der Relinearisierung entsteht, ist unabhängig von dem Rauschen des Chiffretextes, der relinearisiert wird. Nach [67] wird er durch $\|\underline{\boldsymbol{r}}\| < \frac{(l+1)\cdot B\cdot T\cdot \delta_R}{2}$ begrenzt. Die Auswirkungen der Wahl des Parameters T werden später in Kapitel 4.6 erläutert.

Version 2: Die alternative Herangehensweise der Relinearisierung wird in den bekannten Implementierungen, wie z. B. die von uns genutzte Bibliothek SEAL, nicht umgesetzt. Daher wird sie im Folgenden nur sehr knapp vorgestellt. Tiefergreifende Informationen und Beweise befinden sich in [67].

Zur Erinnerung: Unser Ziel ist es, $\underline{\boldsymbol{s}}^2$ im Relinearisation-Key unterzubringen, ohne Rückschlüsse auf $\underline{\boldsymbol{s}}^2$ bei bekanntem Relinearisation-Key zu erlauben und ohne dass der Fehler Term $\underline{\boldsymbol{r}}$ zu groß wird. Dazu wird $\underline{\boldsymbol{s}}^2$ mit einer beliebigen ganzen Zahl p unkenntlich gemacht. Damit das Rauschen bei der Relinearisierung verschwindet, muss der modulo Operator auf $p\cdot q$ angepasst werden und entsprechend $\underline{\boldsymbol{a}}$ aus $R_{p\cdot q}$ gezogen werden. Des Weiteren muss bei der Wahl der Varianz der Fehlerverteilung χ' sehr vorsichtig vorgegangen werden, da nach [67] ansonsten das Sicherheitslevel stark beeinträchtigt wird. In Algorithmus 4.11 wird die Relinearisierung auf Basis des oben vorgestellten Relinearisation-Keys durchgeführt. Mit der Division durch p wird p herausgerechnet und anschließend modulo q berechnet. Das Ergebnis entspricht nach [67] der Invariante:

Eingabe:sk (Sectret Key), p
Ausgabe:rlk

$\underline{\boldsymbol{a}} \leftarrow R_{p\cdot q}$
$\underline{\boldsymbol{e}} \leftarrow \chi'$
$rlk = ([-(\underline{\boldsymbol{a}} \cdot \underline{\boldsymbol{s}} + \underline{\boldsymbol{e}}) + p \cdot \underline{\boldsymbol{s}}^2]_{p\cdot q}, \underline{\boldsymbol{a}})$
return rlk

Algorithmus 4.10: Relinearisation-Key v2: EvaluateKeyGen(sk, p)

$$\underline{\boldsymbol{c}}_{2,0} + \underline{\boldsymbol{c}}_{2,1} \cdot \underline{\boldsymbol{s}} = \underline{\boldsymbol{c}}_2 \cdot \underline{\boldsymbol{s}}^2 + \underline{\boldsymbol{r}}$$

mit $\|\underline{\boldsymbol{r}}\| < \frac{q\cdot B_k\cdot \delta_R}{p} + \frac{\delta_R\cdot \|\underline{\boldsymbol{s}}\|+1}{2}$.

Eingabe:$ct = [\underline{\boldsymbol{c}}_0, \underline{\boldsymbol{c}}_1, \underline{\boldsymbol{c}}_2]$ (Chiffretext), rlk (Relinearisation-Key)
Ausgabe:ct

$(\underline{\boldsymbol{c}}_{2,0}, \underline{\boldsymbol{c}}_{2,1}) = ([\lfloor \frac{\underline{\boldsymbol{c}}_2 \cdot rlk[0]}{p} \rceil]_q, [\lfloor \frac{\underline{\boldsymbol{c}}_2 \cdot rlk[1]}{p} \rceil]_q)$
$ct = ([\underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{c}}_{2,0}]_q, [\underline{\boldsymbol{c}}_1 + \underline{\boldsymbol{c}}_{2,1}]_q)$
return ct

Algorithmus 4.11: Relinearisierung v2: FV.SH.Relin(ct, rlk)

4.5 FV in SEAL v2.3.0-4

In unseren Experimenten nutzen wir die Implementierung von FV aus der Bibliothek SEAL (simple encrypted arithmetic library) in der Version v2.3.0-4 von Microsoft. Dort wurde die FV Verschlüsselung leicht verändert bzw. generalisiert. Daher wollen wir kurz auf diese Änderungen eingehen. Hierbei orientieren wir uns an den Erläuterungen und Formeln, die in [33] genutzt wurden.

Der Klartextraum bleibt derselbe, sodass alle Elemente Polynome aus R_t sind. Die Chiffretexte hingegen sind in ihrer Länge nicht mehr auf lineare Polynome begrenzt. D.h., ein Chiffretext $ct = (\underline{\boldsymbol{c}}_0, \underline{\boldsymbol{c}}_1, \ldots, \underline{\boldsymbol{c}}_k)$ besteht aus $k+1$ Polynomen aus dem R_q, wobei $k \geq 1$ gilt. Damit lassen sich die Chiffretexte als Polynom k-ten Grades ansehen. Das i-te Polynom ist dabei zugehörig zur i-ten Potenz des Secret-Keys $\underline{\boldsymbol{s}}$. Wegen dieser Verallgemeinerung auf längere Chiffretexte wurde es nötig, die verschiedenen Operationen ebenfalls für längere Chiffretexte anzupassen.

4.5.1 Entschlüsselung

Die Entschlüsselung entspricht der Verallgemeinerung der Entschlüsselung der FV Verschlüsselung in Algorithmus 4.4.

$$[\lfloor \frac{t}{q} \cdot [ct(\underline{\boldsymbol{s}})]_q \rceil]_t = [\lfloor \frac{t}{q} \cdot [\underline{\boldsymbol{c}}_0 + \cdots + \underline{\boldsymbol{c}}_k \underline{\boldsymbol{s}}^k]_q \rceil]_t \ .$$

4.5.2 Addition

Seien $ct_1 = (\underline{\boldsymbol{c}}_0, \underline{\boldsymbol{c}}_1, \ldots, \underline{\boldsymbol{c}}_j)$ und $ct_2 = (\underline{\boldsymbol{d}}_0, \underline{\boldsymbol{d}}_1, \ldots, \underline{\boldsymbol{d}}_k)$ zwei Chiffretexte der Größe $j+1$ bzw. $k+1$, wobei $j \leq k$ gilt. Dann ist die generalisierte Addition in SEAL v2.3.0-4 definiert durch:

$$ct_{add} = ([\underline{\boldsymbol{c}}_0 + \underline{\boldsymbol{d}}_0]_q, \ldots, [\underline{\boldsymbol{c}}_j + \underline{\boldsymbol{d}}_j]_q, \underline{\boldsymbol{d}}_{j+1}, \ldots, \underline{\boldsymbol{d}}_k) \ .$$

D.h., die Addition wird analog zu der Addition im ursprünglichen FV durchgeführt und der Rest des längeren Chiffretextes an das Ergebnis angehangen, sodass der entstandene Chiffretext dieselbe Länge des längeren Chiffretext hat (in diesem Fall ct_2 mit einer Länge von $k+1$).

4.5.3 Multiplikation

Da es durch die neue Entschlüsselungsfunktion möglich ist, Chiffretexte mit mehr als zwei Polynomen zu entschlüsseln, ist es nicht zwingend nötig, nach der Multiplikation den Chiffretext zu relinearisieren. Dementsprechend wurde in SEAL v2.3.0-4 die Multiplikation getrennt von der Relinearisierung implementiert.
Seien $ct_1 = (\underline{\boldsymbol{c}}_0, \underline{\boldsymbol{c}}_1, \ldots, \underline{\boldsymbol{c}}_j)$ und $ct_2 = (\underline{\boldsymbol{d}}_0, \underline{\boldsymbol{d}}_1, \ldots, \underline{\boldsymbol{d}}_k)$ zwei Chiffretexte der Größe $j+1$ bzw. $k+1$. Der durch die Multiplikation von ct_1 mit ct_2 entstandene Chiffretext wird notiert als $ct_{mult} = (\underline{\boldsymbol{C}}_0, \underline{\boldsymbol{C}}_1, \ldots, \underline{\boldsymbol{C}}_{j+k})$ und hat eine Länge von $j+k+1$. Die Polynome $\underline{\boldsymbol{C}}_m \in R_q$ werden berechnet über:

$$\underline{\boldsymbol{C}}_m = [\lfloor \frac{t}{q} (\sum_{r+s=m} \underline{\boldsymbol{c}}_r \cdot \underline{\boldsymbol{d}}_s) \rceil]_q \ .$$

$r + s = m$ steht dabei für alle Kombinationen von $r + s$, die m ergeben. Im Falle vom ursprünglichen FV aus [67] entspricht das genau der Multiplikationsfunktion von FV.

4.5.4 Relinearisierung

Im ursprünglichen FV war die Relinearisierung nach jeder Multiplikation nötig und hat so nur Chiffretexte der Länge 3 auf Chiffretexte der Länge 2 verkleinert und damit wieder relinearisiert. In SEAL v2.3.0-4 reduziert die Relinearisierung einen Chiffretext um eine Länge von 1 auf mindestens 2 Elemente. D.h., für einen gegebenen Chiffretext $ct = (\underline{\boldsymbol{c}}_0, \ldots, \underline{\boldsymbol{c}}_k)$ der Länge $k+1$ erzeugt die Relinearisierung einen Chiffretext $(\underline{\boldsymbol{c}}'_0, \ldots, \underline{\boldsymbol{c}}'_{k-1})$ der Länge k. Diese Funktion kann so oft angewendet werden, bis der Chiffretext die Länge $k = 2$ erreicht hat und führt bei einer Entschlüsselung zu gleichen Ergebnissen.

Hierzu wird eine Generalisierung der Version 1 der Relinearisierung aus dem ursprünglichen FV genutzt. Dabei ist es nötig, eine Menge an Relinearisation-Keys zu erzeugen. Wir erzeugen einen für jede Potenz von $\underline{\boldsymbol{s}}$, d.h., rlk_2 beinhaltet $\underline{\boldsymbol{s}}^2$, rlk_3 beinhaltet $\underline{\boldsymbol{s}}^3$ und so weiter bis rlk_k, der $\underline{\boldsymbol{s}}^k$ beinhaltet. Jeden dieser Relinearisation-Keys erzeugen wir gemäß der Schlüsselerzeugung im ursprünglichen FV. Die Relinearisierung führen wir analog zu der ursprünglichen Version von FV durch, außer dass wir das letzte Polynom mit den ersten beiden verrechnen. Sei der zu relinearisierende Chiffretext ct definiert durch $(\underline{\boldsymbol{c}}_0, \underline{\boldsymbol{c}}_1, \ldots, \underline{\boldsymbol{c}}_k)$, dann wird in der Relinearisation der Chiffretext $(\underline{\boldsymbol{c}}'_0, \underline{\boldsymbol{c}}'_1, \underline{\boldsymbol{c}}_2 \ldots, \underline{\boldsymbol{c}}_{k-1})$ berechnet, der nach der Entschlüsselung zu dem gleichen Ergebnis führt.

$$\underline{\boldsymbol{c}}'_0 = [\underline{\boldsymbol{c}}_0 + \sum_{i=0}^{l} rlk_k[i][0] \cdot \underline{\boldsymbol{c}}_k^{(i)}]_q$$

$$\underline{\boldsymbol{c}}'_1 = [\underline{\boldsymbol{c}}_1 + \sum_{i=0}^{l} rlk_k[i][1] \cdot \underline{\boldsymbol{c}}_k^{(i)}]_q \ .$$

Dabei müssen wir beachten, dass uns zur Erzeugung der Relinearisation-Keys der Secret-Key $\underline{\boldsymbol{s}}$ bekannt sein muss. Der Besitzer des Schlüssels muss dementsprechend die nötige

Anzahl an Schlüsseln für die nötigen Berechnungen vorher erzeugen. Wird nach jeder Multiplikation relinearisiert, ist es nur nötig rlk_2 bereitzustellen, womit sich die Verschlüsselung wie das ursprüngliche FV relinearisiert.

4.5.5 Weitere Operationen

In SEAL v2.3.0-4 können wir einen Chiffretext negieren, sodass wir mit einer Addition eines negierten Chiffretextes eine Subtraktion ausführen können.
Des Weiteren stehen in SEAL v2.3.0-4 ein paar zusätzliche Methoden zur Laufzeitverbesserung bereit:

- Multiplikation und Addition von Klartextpolynomen mit Chiffretexten
- Quadrieren eines Chiffretextes
- Potentieren eines Chiffretextes

4.5.6 Rauschverhaltens in SEAL

Das Rauschen und die Rauschentwicklung wurden in einer Tabelle aus [33, S. 15] sehr gut zusammengefasst. Einen Ausschnitt dieser Tabelle wollen wir an dieser Stelle dem Leser mit an die Hand geben. Um das Rausch Budget zu überprüfen stellt SEAL v2.3.0-4 die Funktion "invariant_noise_ budget(ct)" bereit, die das verfügbare Rausch Budget von einem Chiffretext in Bit zurück gibt.

Operation	Input	Schranke für das Rauschen
Verschlüsseln	Klartext $\underline{\boldsymbol{m}}$	$\frac{r_t(q)}{q} \cdot \lVert\underline{\boldsymbol{m}}\rVert \cdot N_{\underline{\boldsymbol{m}}} + \frac{7nt}{q} \cdot \min\{B, 6\sigma\}$
Negieren	Chiffretext ct	$\underline{\boldsymbol{v}}$
Add. / Sub.	Chiffretexte ct_1 und ct_2	$\underline{\boldsymbol{v}}_1 + \underline{\boldsymbol{v}}_2$
Add. / Sub. mit Klartext	Chiffretext ct und Klartext $\underline{\boldsymbol{m}}$	$\underline{\boldsymbol{v}} + \frac{r_t(q)}{q} \cdot \lVert\underline{\boldsymbol{m}}\rVert \cdot N_{\underline{\boldsymbol{m}}}$
Multiplikation mit Klartext	Chiffretext ct und Klartext $\underline{\boldsymbol{m}}$	$\underline{\boldsymbol{v}} \cdot \lVert\underline{\boldsymbol{m}}\rVert \cdot N_{\underline{\boldsymbol{m}}}$
Multiplikation	Chiffretexte ct_1 (Größe j_1+1) und ct_2 (Größe j_2+1)	$t \cdot \sqrt{3n} \cdot [\underline{\boldsymbol{v}}_2 \cdot (12n)^{j_1/2} + \underline{\boldsymbol{v}}_1 \cdot (12n)^{j_2/2} + (12n)^{(j_1+j_2)/2}]$
Quadrieren	Chiffretexte ct (Größe j)	das gleiche wie $ct \cdot ct$
Relinearisierung	Chiffretexte ct (Größe K), Zielgröße L, mit $2 \leq L < K$	$\underline{\boldsymbol{v}} + \frac{2t}{q} \cdot \min\{B, 6\sigma\} \cdot (K-L) \cdot n \cdot (l+1) \cdot w$

Tabelle 4.1: Ausschnitt der Rausch Schätzungen aus SEAL. Quelle: [33, S. 15].

4.6 Parameter

Um einen Überblich über die Parameter zu geben, haben wir diese vorweg in Tabelle 4.2 zusammengefasst.

Parameter	Name	Möglichkeiten
$\Phi_m(x)$	Polynommodulus	• $x^n + 1$
q	Koeffizientenmodulus	• Produkt eindeutiger Primzahlen mit Länge bis 60 Bit • SEAL Wahl: in Abhängigkeit von n und λ nach Tabelle 4.3
t	Klartextmodulus	• $t \in \mathbb{Z}$ • beliebig • $t = n$ (Ergebnis Experimente, vgl. Kapitel 7.1.3)
T	Basis (Relinearisierung)	• $1 \leq T \leq 60$ • $T = \lceil \sqrt{q} \rceil$ (bspw.)
σ	Standardabweichung (χ Verteilung)	• $\sigma = 3.19$ (Standard)

Tabelle 4.2: Parameter der FV Verschlüsselung. λ: Sicherheitsparameter.

Die Parameter der FV Verschlüsselung wirken sich stark auf die Laufzeit und die Größe der Verschlüsselung aus. Die Begrenzung $\Phi_m(x)$ unseres Ringes R wird Polynommodulus genannt. Es handelt sich dabei um ein Kreisteilungspolynom. In SEAL v2.3.0-4 wird er auf die Form $x^n + 1$ begrenzt, wobei n eine beliebige gerade Zahl darstellt. Ein größerer Polynommodulus führt zu einem höheren Sicherheitslevel, aber auch zu größeren Chiffretexten und höherer Laufzeit.

Der Koeffizientenmodulus q bestimmt die Größe der Koeffizienten innerhalb des Chiffretextes, da diese auf R_q definiert sind. In der ursprünglichen Version von FV war q nicht eingeschränkt. In SEAL v2.3.0-4 wurde die Laufzeit mithilfe des "Chinese Remainder Theorems" optimiert. Aus diesem Grund wird in SEAL q als Produkt eindeutiger Primzahlen mit einer beliebigen Bitlänge bis 60 Bits definiert. Die Anzahl der Primfaktoren sollte möglichst gering gehalten werden, da sie die Laufzeit negativ beeinflussen. Ein größerer Koeffizienten Modulus ermöglicht ein größeres Rausch Budget, verringert aber gleichzeitig das Sicherheitslevel. Dies kann zu Lasten der Laufzeit wiederum durch eine Erhöhung des Polynommodulus kompensiert werden. Um ein vorher gewähltes Sicherheitsniveau von λ zu garantieren, sind daher in SEAL die Funktionen "coeff_modulus_128(n)", für $\lambda = 128$ und "coeff_modulus_192(n)", für $\lambda = 192$ implementiert. Sie geben zu einem gewählten

n den entsprechend geeigneten Parameter q zurück, um das jeweilig angestrebte Sicherheitsniveau λ zu garantieren. Die Bit Längen von q, die diese Funktion liefert, befinden sich in Tabelle 4.3. Tendentiell kann q kleiner gewählt werden, wenn dies die Größe der darzustellenden Zahlen erlaubt. Dies erhöht das Sicherheitsniveau. SEAL v2.3.0-4 stellt hierzu die Funktionen "small_mods_60bit", "small_mods_50bit", "small_mods_40bit" und "small_mods _30bit" bereit.

n	Bit Länge von q	
	128 Bit Sicherheit	192 Bit Sicherheit
1024	29	20
2048	56	39
4096	110	77
8192	219	153
16384	441	300
32768	885	600

Tabelle 4.3: Bitlänge q für 128 Bit und 192 Bit Sicherheitslevel in Abhängigkeit von n. Quelle: [33, S. 22].

Der Klartextmodulus t bestimmt, wie groß die verschlüsselbaren Klartexte bzw. dessen Koeffizienten werden können, da sie in R_t definiert sind. Er kann als beliebige ganze Zahl gewählt werden. Eine Erhöhung des Klartextmodulus sorgt neben der Möglichkeit größere Nachrichten zu verschlüsseln für ein kleineres Rausch Budget, einen höheren Verbrauch des Rausch Budgets sowie eine höhere Laufzeit. Er sollte dementsprechend möglichst gering gehalten werden. In unseren folgenden Experimenten hat sich gezeigt, dass $t = n$ für korrekte Ergebnisse gewählt werden sollte (vgl. Kapitel 7.1.3).

Die Basis T zur Aufteilung eines Polynoms innerhalb der Relinearisierung sorgt bei einer Erhöhung für einen kleineren Relinearisation-Key und ein erhöhtes Rauschen während der Relinearisierung. Zur Wahl von T gibt es zwei Strategien: Es bietet sich an, T mindestens so groß zu wählen, dass das Rauschen der Relinearisierung in etwa so hoch ist wie das Rauschen bei einer Multiplikation, sodass wir einen guten Mindestwert für die Größe von T berechnen können. Sollen hingegen die Laufzeit und der Speicherplatz der Relinearisierung optimiert werden, sollte T möglichst groß gewählt werden, z. B. $T = \lceil \sqrt{q} \rceil$, da wir das Polynom so nur in 2 Teile unterteilen müssen. Diese Empfehlung basiert auf der Bitlänge des genutzten q. Für ein so großes T wird typischerweise die erste Relinearisierung zu einem deutlich höherem Rauschen führen als die darauffolgenden.

Der letzte Parameter der FV Verschlüsselung ist die Standardabweichung σ von χ. Da dieser Parameter stark die Sicherheit der Verschlüsselung beeinflusst, muss er sehr

sorgfältig gewählt werden. Es wird in [33] empfohlen: $\sigma = 3,19$. Wurden in SEAL alle Parameter gewählt, muss ein "SEALcontext" Objekt erzeugt werden, dass die gewählten Parameter überprüft und einen Fehler ausgibt, wenn diese in der gewählten Kombination nicht wählbar sind.

4.7 Encoder

Da die Verschlüsselung und ihre Operationen nur auf Polynomen ausführbar sind, meist aber Zahlen verschlüsselt oder verrechnet werden sollen, ist es nötig, diese Zahl in ein Polynom umzuwandeln bzw. zu kodieren (engl.: encode). Diese Aufgabe übernimmt der Encoder, der auch für das spätere Dekodieren (engl.: decode) eines Polynoms in eine Zahl verwendet wird. Das Vorzeichen der Polynome wird bei der Kodierung durch die Signum Funktion gebildet. Es gilt für eine Zahl a:

$$\text{sign}(a) = \begin{cases} +1 & \text{falls } a > 0 \\ 0 & \text{falls } a = 0 \\ -1 & \text{falls } a < 0 \ . \end{cases}$$

4.7.1 Integer Encoder

Der Integer Encoder kodiert eine ganze Zahl $a \in \mathbb{Z}$ in ein Polynom mit vorher zu wählender Basis $B \geq 2$. Genau genommen ist der Integer Encoder eine Sammlung an Encodern. Einer für jede wählbare Basis B. Für $B = 2$ wird die ganze Zahl $a \in [-(2^n - 1), (2^n - 1)]$ in 2-komplement Format bzw. Binärdarstellung $|a| = a_{n-1} \ldots a_1 a_0$ umgewandelt. Die Kodierung von a mit Basis B ist dann:

$$\text{IntegerEncode}(a, B = 2) = \text{sign}(a) \cdot (a_{n-1} x^{n-1} + \ldots + a_1 x + a_0) \ .$$

Die Dekodierung eines Polynoms in eine Zahl erfolgt für jedes beliebige B durch die Auswertung des Polynoms an der Stelle $x = B$, also in diesem Fall an der Stelle $B = 2$. Bei der Verrechnung zweier Polynome sind die Koeffizienten nicht mehr durch 1 begrenzt und vergrößern sich. Es gilt:

$$\text{IntegerDecode}[\text{IntegerEncode}(a, B = 2) + \text{IntegerEncode}(b, B = 2)] = a + b \ ,$$
$$\text{IntegerDecode}[\text{IntegerEncode}(a, B = 2) \cdot \text{IntegerEncode}(b, B = 2)] = a \cdot b \ .$$

Ein korrektes Dekodieren ist nur möglich, wenn während der Verrechnung zweier Polynome keine Reduktion modulo $x^n + 1$ oder modulo t stattfindet.

Wird eine Basis $B > 2$ gewählt, wird die Zahl a in B-Komplementformat umgewandelt und das Polynom analog zum Fall $B = 2$ mit Koeffizienten aus der symmetrischen Menge $\{-\frac{B-1}{2}, \ldots, \frac{B-1}{2}\}$ gebildet. Auf diese Weise kann jede ganze Zahl $a \in \{-\frac{B^n-1}{2}, \frac{B^n-1}{2}\}$ mit n Koeffizienten dargestellt werden. Je größer B gewählt wird, desto kompakter wird das

Polynom mit dem Nachteil, dass die Koeffizienten größer werden. Es wird in [33] empfohlen, $B = 2$ oder 3 zu wählen. Der Unterschied der beiden Basen ist relativ gering. Im Fall $B = 3$ ist die Koeffizienten Größe in etwa gleich groß, die Repräsentation aber kompakter. Wird keine Basis gewählt, ist $B = 2$ Standard.

4.7.2 Fractional Encoder

Um Dezimalzahlen zu kodieren, ist der einfachste und oft effektivste Weg, sie zu einer ganzen Zahl zu skalieren, die Berechnungen im Code anzupassen und über den Integer Encoder zu kodieren. Da aber Dezimalzahlen in den Berechnungen nötig sein können (z. B. um eine Divisionsberechnung zu ermöglichen durch eine Multiplikation mit dem multiplikativen Inversen) und eine Skalierung sehr aufwändig sein kann, stellt SEAL einen Encoder zum Kodieren von Dezimalzahlen bereit. Der Fractional Encoder benötigt zwei zusätzliche Parameter:

- n_i: Anzahl an Bits im Polynom, die für den ganzzahligen Anteil reserviert werden.
- n_f: Anzahl an Bits im Polynom, die für den Anteil an Nachkommastellen reserviert werden.

Dabei muss $n \geq n_f + n_i$ gelten. Im Folgenden werden wir die Kodierung für den Fall $B = 2$ erklären. Die Verallgemeinerung für größere Basen funktioniert analog zu der im Integer Encoder.

Zum Kodieren einer Dezimalzahl wird die Zahl r in ihrem 2-Komplementformat $|r|$ in den ganzzahligen Teil und den Teil der Nachkommastellen unterteilt und getrennt voneinander kodiert. Für die Kodierung des ganzzahligen Anteils wird der Integer Encoder genutzt und der Teil der Nachkommastellen durch einen speziellen Shift, den wir im Folgenden noch erklären werden, rekursiv durch einen weiteren Aufruf an die Kodierung addiert. So kommen wir im Fall $B = 2$ zu der Kodierung:

$$\text{FracEncode}(r, B = 2) = \text{sign}(r)[\text{IntegerEncode}(\lfloor |r| \rfloor, B = 2) + \text{FracEncode}(\{|r|\}, B = 2)] \ .$$

$\{|r|\}$ bezeichnet den Shift der Nachkommastellen der Zahl r. Im Folgenden nehmen wir an, dass die Nachkommastellen durch n_f begrenzt sind. Für diesen Shift liegt der Bereich der Nachkommastellen in seiner 2-Komplementform $|r| = r_{-1}r_{-2}\ldots r_{-n_f}$ vor. Schreiben wir das ganze analog zu dem Vorgehen im Integer Encoder in Exponentialschreibweise und ersetzen die Basis durch die Variable x, erhalten wir $r_{-1} \cdot x^{-1} + r_{-2} \cdot x^{-2} + \ldots + r_{-d} \cdot x^{-n_f}$. Der Shift wird nun durch ein Aufaddieren von n zu jedem Exponenten erzeugt und dabei das Vorzeichen jedes Koeffizienten invertiert, sodass wir folgendes Ergebnis erhalten:

$$\text{FracEncode}(\{|r|\}, B = 2) = -r_{-1} \cdot x^{n-1} - r_{-2} \cdot x^{n-2} - \ldots - r_{-d} \cdot x^{n-n_f} \ .$$

Wir shiften so die Koeffizienten, die die Nachkommastellen repräsentieren, an den Anfang des Polynoms. Um die Funktionsweise richtig zu verstehen, wollen wir analog zu [33] das Ganze an einem Beispiel verdeutlichen. Wollen wir z. B. die Zahl $5,875$ mit der Basis $B = 2$ kodieren gilt $|5,875| = 2^2 + 2^0 + 2^{-1} + 2^{-2} + 2^{-4}$. Für den ganzzahligen Anteil erhalten wir IntegerEncode$(5, B = 2) = x^2+1$. Der Teil der Nachkommastellen $2^{-1}+2^{-2}+2^{-4}$ wird nach Ersetzen der Basis durch x und dem Shift um n zu $x^{n-1}+x^{n-2}+x^{n-4}$, sodass wir durch das Invertieren des Vorzeichens als Ergebnis FracEncode$(\{0,875\}, B = 2) = -x^{n-1} - x^{n-2} - x^{n-4}$ erhalten. Insgesamt erhalten wir also FracEncode$(5,875, B = 2) = -x^{n-1} - x^{n-2} - x^{n-4}+x^2+1$. Die Nachkommastellen werden also als negative Koeffizienten an den Anfang des Polynoms geshiftet. Aus diesem Grund muss die Anzahl der Nachkommastellen limitiert sein, da ansonsten der ganzzahlige Anteil der Zahl im Polynom durch die Koeffizienten der Nachkommastellen überschrieben werden kann.

Die Dekodierung funktioniert genau anders herum. Die ersten n_i Koeffizienten bilden den Teil, der den ganzzahligen Anteil an der Zahl repräsentiert, die restlichen Koeffizienten von hohem Grad repräsentieren den Teil der Nachkommastellen der Zahl. In diesem Teil werden die Vorzeichen invertiert und das Polynom um $-n$ zurückgeshiftet. Schlussendlich muss das so entstandene Polynom sowie der Teil mit niedrigem Grad, der den ganzzahligen Anteil der Zahl repräsentiert, noch an der Stelle $x = 2$ ausgewertet und die beiden Teile aufaddiert werden.

Ein korrektes Dekodieren ist nur möglich, wenn während der Verrechnung zweier Polynome keine Reduktion modulo $x^n + 1$ oder modulo t stattfindet, genauso wie bei dem Integer Encoder und zusätzlich, falls sich der Bereich der Nachkommastellen nicht mit dem ganzzahligen Bereich vermischt. Grundsätzlich sollte aus diesem Grund n_f möglichst gering gewählt werden.

Wird keine Basis gewählt, ist $B = 2$ Standard. Für Interessierte finden sich in [33] weitere Beispiele zur Funktionsweise dieses Encoders.

4.7.3 Weitere Encoder

SEAL unterstützt noch zwei weitere Encoder. Den Scalar Encoder, welcher nach [33] eine schlechtere Version des Integer Encoders darstellt, sowie den CRT Batching Encoder, der es ermöglicht, mehrere Zahlen in einem Polynom zu kodieren, sodass ein Batching mittels Chinese Remainder Theorem (CRT) ermöglicht wird. Da Batching aber kein Teil dieser Bachelor Arbeit werden wird, verweisen wir auf [33] für weitere Informationen.

4.8 Drittes Zwischenfazit

Wir haben in diesem Kapitel gesehen, dass unter dem Label FV Algorithmus mehrere Versionen existieren. Wir benutzen für unsere Experimente die Version aus SEAL v2.3.0-4.

Nachdem ein potentieller Klient schwere Entscheidungen zur Auswahl des zu nutzenden Kryptosystems getroffen hat, wird er zu weiteren Entscheidungen im Rahmen der Parametrisierung gezwungen. Wir haben gezeigt, dass sich die Parameter untereinander beeinflussen, zeitgleich aber sehr offene Beschränkungen haben. Dies führt uns zu der Vermutung, dass für eine optimale Parameterwahl maßgeschneiderte Parametrisierungen nötig werden. Es kommen Zweifel auf, dass "die eine" beste Parametrisierung existiert. Den Prozess der Parameterfindung werden in unseren Experimenten in Kapitel 7 am Beispiel mehrerer k-means Clusteranalysen genauer betrachten und die entsprechenden Laufzeiten dokumentieren. Im folgenden Kapitel werden wir das Verfahren der Clusteranalyse genauer erläutern.

Kapitel 5

Clusteranalyse

Die Clusteranalyse oder kurz das Clustering ist ein Verfahren zur Gruppenbildung, bei dem ähnliche Objekte zu Gruppen zusammengefasst werden. Clusteranalysen basieren auf Beaobachtungen. Im Alltag wenden wir beispielsweise solch ein Verfahren täglich unbewusst an, wenn wir Dackel, Huskies und Doggen zur Familie der Hunde zählen und nicht zur Familie der Katzen. Im wissenschaftlichen Kontext wurden Clusteranalysen bereits von Aristoteles angewandt [145]. Nach der Überblicksarbeit von Jain [93] ist die Aufteilung großer Datenmengen in disjunkte Teilmengen mit ähnlichen Eigenschaften, sogenannte Cluster, die bedeutungsvoll, nützlich oder beides sind, eine intuitive und zentrale Fragestellung in den verschiedensten wissenschaftlichen Disziplinen. In unserem Zeitalter von Big Data gewinnen unüberwachte Clusteranalysen (engl.: unsupervised clustering) zunehmend an Bedeutung. Diese zielen darauf ab, Strukturen in Daten ohne Hilfe von Klassenlabels oder einer Expertenmeinung zu finden. Als skalierbarer, maschineller Lernalgorithmus zur Bewältigung der steigenden Datenmenge setzen wir unsupervised Clustering nach [115, S. 4] hauptsächlich ein:

1. wenn die Einsetzung des Labels im Dataset aufwendig oder unmöglich ist,
2. wenn die verfügbaren Labels von den Daten missverständlich sind,
3. wenn das Verständnis der Dateneigenschaften verbessert werden muss,
4. wenn die Summe der Daten reduziert und die originalen Daten transformiert werden sollen.

Unüberwachte Clusteranalysen sind somit ein grundlegendes Werkzeug für Data Mining, Dokumentenabruf, Image-Segmentation und Pattern-Classifikation [93, 115, 140, 145]. Zur Lösung des Optimierungsproblems wurde bereits in den 1950er Jahren die auch heute noch meist verwendete Partitionierungsstrategie, die k-means Kostenfunktion, entwickelt [110, 153, 94]. Der dazu geeignete Algorithmus nach Lloyd [110] wurde von den Organisatoren der IEEE International Conference on Data Mining, 2008, als einer der zehn

einflussreichsten Algorithmen weltweit bezeichnet [161]. Aufgrund seiner großen Bedeutung und Verbreitung haben wir dieses Verfahren als Beispielsanwendung einer Verschlüsselung gewählt.

5.1 Grundlegende Definitionen

Im Folgenden klären wir grundlegende Definitionen und diskutieren Vor- und Nachteile. Die unüberbewachte Clusteranalyse ist standardmäßig für Vektoren aus $\mathbb{R}^d$ definiert [140]. Diese Vektoren bezeichnen wir als Beobachtungen.

5.1.1 Definition (Beobachtung). Der zugrunde liegende Raum möglicher Beobachtungen ist eine d dimensionale Zufallsvariable X. Eine Beobachtung ist ein Vektor $\vec{x} = \{X_1, \ldots, X_d\} \in X$, bestehend aus den Merkmalen X_j. Sei $\mathcal{T} = \{\vec{x}_1, \ldots, \vec{x}_n\} \subset X$ eine Menge von n Beobachtungen, dann beschreiben wir die Merkmale der Beobachtung $\vec{x}_i$ als X_{ij}. Das Merkmal heißt multivariat für $d > 1$, sonst univariat und wir schreiben $x \equiv \vec{x}$.

Die Daten, die wir bei unseren Experimenten FV verschlüsselt einsetzen werden, können nicht nur an einem Zeitpunkt, sondern auch über Zeitspannen erhoben worden sein. Die so entstehenden Listen von reellwertigen, zeitabhängigen Messwerten werden als Zeitreihen bezeichnet.

5.1.2 Definition (Zeitreihe). Sei $\vec{x}_t$ eine Beobachtung zum Zeitpunkt $t \in \mathbb{N}$. Eine Zeitreihe ist eine Menge $Z \subset X$ von Beobachtungen, wobei auf Z eine strenge Totalordnung $<_t \subseteq X \times X$ mittels der Zeit definiert ist, sodass $\vec{x}_i <_t \vec{x}_j \Leftrightarrow i < j$ gilt. Die Zeitreihe heißt multivariat für $d > 1$, sonst univariat.

Zeitreihen müssen nicht zwingend die gleiche Länge haben. In der Praxis wählen wir daher aus einer gegebenen Zeitreihe Merkmale wie die Länge, den maximalen oder den minimalen Wert, den Durchschnitt aller oder bestimmter Werte oder Standardabweichungen [117, 104]. Beispielsweise ist der maximale Wert ausreichend, wenn wir eine Stichprobe bei Geschwindigkeitsmessungen in Temposündern und Nicht Temposünder klassifizieren wollen. Wir müssen jedoch bei diesem Vorgehen bedenken, dass solche Transformationen immer mit einem Informationsverlust verbunden sind und die spezifische Form der Zeitreihe verloren geht. Weiter gehen wir auf diese Problematik nicht ein, da die uns zur Verfügung stehenden Daten für unsere Experimente bereits in vorverarbeiteter Form vorliegen.

Nach diesen unabdingbaren Vorbereitungen und der üblichen Datenbereinigung wie dem Umgang mit fehlenden Werten, der Ausreißernspektion etc. können wir damit beginnen, die Eingabemenge an Daten so in Teilmengen, den Clustern, zu partitionieren, dass die Elemente eines Clusters sich so ähnlich wie möglich sind (Definition 5.1.4) und sich von den Elementen der anderen Cluster maximal unterscheiden (Definition 5.1.5). Wir benötigen dazu also ein Ähnlichkeits- oder Abstandsmaß, $d : X \times X \to \mathbb{R}^+$, das für zwei

Elemente $\vec{x}_1, \vec{x}_2 \in X$ einen reellwertigen Abstand ermittelt. Die Zuordnung der Punkte zu den jeweiligen Clustern wird durch die Qualitätsfunktion beschrieben.

5.1.3 Definition (Qualitätsfunktion). Sei die Anzahl k der Cluster gegeben und jedes Cluster durch eine ganze Zahl $j \in \{1, 2, \ldots, k\}$ eindeutig ausgezeichnet. Die Abbildung $C(i) = j$ weist der i-ten Beobachtung das k-te Cluster zu.

5.1.4 Definition (Innerer Abstand Within). Minimiert werden soll der Abstand innerhalb eines Clusters C:

$$W(C) = \frac{1}{2} \cdot \sum_{j=1}^{k} \sum_{C(i)=j} \sum_{C(i')=j} d(\vec{x}_i, \vec{x}_{i'}) \ .$$

5.1.5 Definition (Zwischenunähnlichkeit Between). Maximiert werden soll der Abstand zwischen den Clustern:

$$B(C) = \frac{1}{2} \cdot \sum_{j=1}^{k} \sum_{C(i)=j} \sum_{C(i')\neq j} d(\vec{x}_i, \vec{x}_{i'}) \ .$$

Dies führt uns zu der formalen Definition der Clusteranalyse:

5.1.6 Definition (Optimierungsproblem der Clusteranalyse). Wenn wir die Summe aller Abstände $T = \frac{1}{2} \sum_{i=1}^{n} \sum_{i'=1}^{n} d(i, i')$ betrachten, ergänzen sich $W(C) + B(C) = T$, sodass die Minimierung von $W(C)$ automatisch der Maximierung von $B(C)$ entspricht. Daher haben wir nur ein Optimierungsproblem. Sei $\bar{x}_j = (\bar{x}_{1j}, \ldots, \bar{x}_{dj})$ der Vektor der Mittelwerte aller Variablen in Cluster j und $N_j = \sum_{i=1}^{n} I(C(i) = j)$, dann ist das Optimierungsproblem:

$$C^* = \min_C \sum_{j=1}^{k} N_j \sum_{C(i)=j} (d(\vec{x}_i, \bar{x}_i))^2 \ .$$

5.1.7 Definition (Distanz). Zur Ermittlung der Distanz zweier Vektoren $\vec{x}, \vec{y} \in X$ gibt es verschiedene Distanzfunktionen, die beim Datamining Verwendung finden. Die am häufigsten genutzte Distanz, die wir auch in unserer Umsetzung benutzen, heißt euklidische Distanz:

$$d(\vec{x}, \vec{y}) = \| \vec{x} - \vec{y} \| = \sqrt{\sum_{i=1}^{d} (X_i - Y_i)^2} \ .$$

Entscheidungen, die ein Nutzer der Clusteranalyse treffen muss, sind:

- Bestimmung des Abstandsmaßes
- Formulierung des Optimierungsproblems
- Repräsentation der Cluster
- Bestimmung von k

5.2 Der k-means Algorithmus

Der k-means Algorithmus, der synonym als Lloyd's Algorithmus bezeichnet wird [145, 140], löst das Optimierungsproblem beim Clustering. Er ordnet die zu clusternden Punkte der Menge $\mathcal{T} = \{\vec{x}_1, \ldots, \vec{x}_n\} \subset X$ eindeutig zu k verschiedenen Clustern $C(1), \ldots, C(k)$ zu, sodass die Distanz der Punkte zu dem jeweiligen Clustermittelpunkt, den Zentroiden $\vec{m}_1, \ldots, \vec{m}_k$, kleiner sind als zu jedem anderen Clustermittelpunkt. Verarbeiten kann er nur numerische Merkmale. Als Abstandsmaß $d(\vec{x}, \vec{y})$ nutzen wir die quadratische euklidische Distanz. Das Clustering durch k-means erfolgt in drei Schritten wie Abbildung 5.1 visualisiert:

1. wähle k Beobachtungen aus $\mathcal{T}$ zufällig als initiale Zentroiden $\vec{m}_1, \ldots, \vec{m}_k$ von Clustern aus.

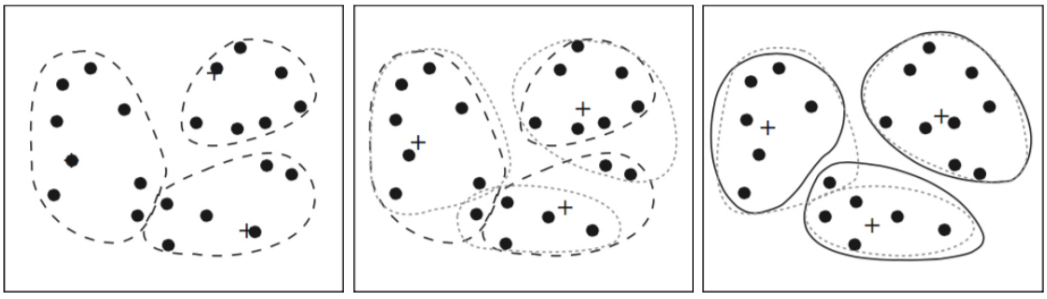
2. Berechne das Clustering anhand der Mittelpunkte:

$$C(i) = \text{argmin}_{1 \leq j \leq k} d(\vec{x}_i, \vec{m}_j)^2 \ .$$

3. Berechne die Mittelpunkte entsprechend $C(i)$:

$$\vec{m}_i = \text{argmin}_m \sum_{i=1}^{n} d(\vec{x}_i, \vec{m})^2 \ .$$

4. Wiederhole Schritt 2 und 3 bis die Zuweisungen sich nicht mehr ändern.
   Gib zurück: $C(1), \ldots, C(k)$



Anfängliches Cluster | Iteration | Letztendliches Cluster

Abbildung 5.1: Funktionsweise des k-means Algorithmus. +: jeweiliger Zentroid. Quelle: [90, S. 453]

Die Zuweisung von Punkten zu ihrem nächstgelegenen Clustermittelpunkt ist optimal für die gegebenen Zentroiden und für jedes Cluster ist der neu berechnete Zentroid der optimale Clustermittelpunkt. Somit ist die neue Zentroidwahl und damit das einhergehende Clustering entweder besser oder gleich im Vergleich zur vorherigen Iteration. Im letzteren Fall hat der Algorithmus konvergiert und terminiert. Auf diese Weise wird ein lokales

Optimum für das Optimierungsproblem berechnet. Den Beweis lassen wir an dieser Stelle aus. Interessierte können ihn in [58] nachlesen.

Die Initialwahl der Zentroiden beeinträchtigt stark das berechnete Clustering. Soll das Ergebnis in mehreren Durchläufen vergleichbar sein, muss eine feste Initialwahl im ersten Schritt des Algorithmus stattfinden. In der Praxis macht es dagegen Sinn, mehrere Durchläufe mit unterschiedlicher Initialwahl durchlaufen zu lassen, um eine Auswahl unterschiedlicher lokaler Optima zu erhalten. Weiterhin zu beachten ist, dass bei dem quadratischen euklidischen Abstand jeweils die Datenpunkte mit dem größten Abstand auch den größten Einfluss haben. Das Verfahren ist daher ausreißerempfindlich. Der Aufwand der Berechnungen ist proportional zu $n \cdot k$, da für jeden Datenpunkt der Abstand zu jedem Zentroiden berechnet wird.

5.3 Viertes Zwischenfazit

In diesem Kapitel haben wir gelernt, dass die Clusteranalyse fachübergreifend angewandt wird und der k-means Algorithmus das Optimierungsproblem löst. Neben dem k-means Algorithmus gibt es natürlich noch eine Vielzahl weiterer Methoden, die sich mit der gleichen Fragestellung befassen und die je nach Fall bessere und schlechtere Ergebnisse liefern können. Jedoch reicht das Verfahren aus, um aus großen Datenmengen tiefgehende (erste) Erkenntnisse zu ziehen. Zu den großen Vorteilen zählt aus unserer Sicht weiter, dass die k-means Clusteranalyse ein verhältnismäßig einfaches Verfahren ist, das statistisch verlässliche Einsichten in große Datenmengen liefert und damit ein fester Bestandteil im Repertoire der Analysten verschiedenster Fachrichtungen geworden ist. Sein hoher Verbreitungsgrad hat zur Folge, dass der k-means Algorithmus in vielen Statistikprogrammen standardmäßig implementiert ist.

Jedoch hat der k-means Algorithmus einige Nachteile. Die Daten müssen zum Beispiel messbar sein, das heißt nicht kategorial. Außerdem muss (wenigstens ungefähr) die Clusterzahl k bekannt sein. Das Ergebnis einer k-means Clusteranalyse kann, abhängig von den Startwerten, eine andere Struktur als die natürliche Struktur der Daten wiedergeben. Eine erhöhte Genauigkeit erfordert sehr viel mehr Laufzeit.

Kapitel 6

Konzept und Methoden

Mit diesem Kapitel beginnt der praktische Teil dieser Untersuchung. Wir werden das Testszenario und die Testumgebung beschreiben sowie Hinweise zu den Implementierungen und Testdurchführungen geben. Schließlich werden die verwendeten Datensätze vorgestellt.

6.1 Testszenario

Ein denkbares Szenario für den Einsatzbereich von k-means auf verschlüsselten Daten ist die ausgelagerte Berechnung des Clusterings. Nur der Dateneigentümer soll Informationen über die Daten erhalten und nicht die berechnende Instanz. Hierzu stellen wir uns vor, dass die Daten von einem oder mehreren Klienten als Chiffretext zu dem Server geschickt werden. Dieser Server soll keine Informationen über die Daten, die Berechnungen und das Ergebnis erhalten. Dort werden sie aggregiert und der k-means Algorithmus ausgeführt. Wir brauchen ein Kommunikationsprotokoll. Dies ist nötig, da während der Berech-

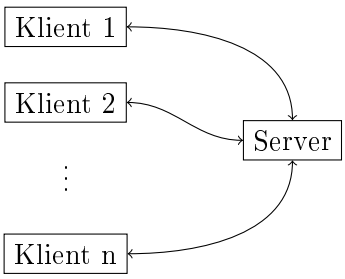


Abbildung 6.1: Kommunikationsprotokoll.

nung der Server Daten zu den Klienten schicken muss, die diese zur weiteren Verarbeitung entschlüsseln, verarbeiten und neu verschlüsseln müssen. Damit niemand währenddessen Rückschlüsse auf die reellen Werte der Daten ziehen kann, fügen wir ein zufälliges Rauschen r hinzu. Soll die Nachticht m neu verschlüsselt werden, sendet der Server den Ciffretext $ct(m+r)$ an einen Klienten, der ihn entschlüsselt. Das Rauschen verhindert einen Informa-

tionsgewinn. Der Server addiert unter Ausnutzen der additiven Homomorphieeigenschaft $-r$ mit dem zurückgegebenen Wert und erhält so die Neuverschlüsselung von $ct(m)$.

6.2 Testumgebung

Unsere Testszenarien werden auf einem Heimcomputer mit Windows 10 Professional (64 Bit), der über einen i7 4770k 3.5 Ghz Quadcore Prozessor mit Hyperthreading und 16 GB Arbeitsspeicher verfügt, innerhalb eines Docker Containers ausgeführt. Docker Container sind Prozesse, die von einem vorher erzeugten Docker Image aus laufen. Dieses stellt alle für die Prozesse benötigten Daten bereit. Die Container laufen so isoliert vom restlichen System. Das Docker image stellt Ubuntu in Version 18.04.01 mit 64 Bit bereit. Der Prozessor wird mit allen acht Kernen genutzt und der Arbeitsspeicher wurde für den Container auf 11.2GB reserviert. Dies stellt das verfügbare Maximum unseres Heimcomputers dar, das wir für Docker reservieren konnten. Zur Umsetzung der Implementierung verwenden wir Python 3. Wir nutzen die offizielle Implementierung SEAL v2.3.0-4 von Microsoft mithilfe eines Pythonwrappers[1]. Die Laufzeitmessungen wurden direkt in der Implementierung durchgeführt. Über die Funktion "clock()" aus der Python Bibliothek "time" wird zu Beginn und zum Ende der Messung die aktuelle Systemzeit ausgelesen. Die Differenz dieser beiden Werte ergibt die Laufzeit in Sekunden.

6.3 K-means Umsetzung

Der Kernpunkt einer k-means Clusteranalyse ist die Distanzfunktion, die einen Größenvergleich impliziert (vgl. Definition 5.1.7). Prinzipiell ist es möglich, auf verschlüsselten Daten zu überprüfen, ob eine Zahl kleiner als die andere ist. Dies setzt vorraus, dass man auf die einzelnen Bits in der Binärdarstellung der Zahl zurückgreifen kann, was bei der FV-Verschlüsselung nicht möglich ist. Aus diesem Grund muss der k-means Algorithmus ohne eine Größenrelation auskommen. Bei der Umsetzung orientieren wir uns an der in [63] vorgestellten k-means Version, die keine explizite Zuweisung der Datenpunkte zu den k verschiedenen Clustern vornimmt, wie in Algorithmus 6.1 zu sehen ist.

Die Initialisierung ist gleich zu der des in Kapitel 5.2 vorgestellten k-means Algorithmus. Werden keine initialen Zentroiden beim Aufruf mit übergeben, werden k Zentroiden zufällig aus der Datenpunktmenge $\mathcal{T}$ gewählt. Damit wir bei der Berechnung der Distanzmatrix eine Division durch 0 vermeiden, führen wir den Shift Parameter s ein. Dieser Parameter s wird abhängig von dem Datensatz als kleinst mögliche Distanz gewählt und ersetzt die Distanz von 0. Beispielsweise ist s bei einem Datensatz mit Ausprägungen die eine Nachkommastelle haben 0.1 oder bei 15 Nachkommastellen $0.1 \cdot 10^{-15}$. Da es auf FV-verschlüsselten Daten nicht möglich ist zu bestimmen, ob zwei verschlüsselte Werte sich

[1] `https://github.com/Lab41/PySEAL`

Eingabe:	$\mathcal{T} = \{\vec{x}_1, \ldots, \vec{x}_n\}$	(Datenpunktmenge)
	k	(Anzahl Cluster)
	i	(Anzahl Iterationen)
	init	(Startzentroiden)
Ausgabe:	$M = \{\vec{m}_1, \ldots, \vec{m}_k\}$	(Menge der jeweiligen Zentroide)
	W	(Gewichtungsmatrix)

1: wähle k initiale Zentroide
2: **while** Anzahl Iterationen $\leq i$ **do**
3: berechne für alle $\vec{x}_i \in \mathcal{T}, \vec{m}_j \in M$:
Distanzmatrix: $d_{ij} = \frac{1}{\|\vec{x}_i - \vec{m}_j\|^2}$
Gewichtungsmatrix: $w_{ij} = \frac{d_{ij}}{\sum_{j=1}^{k} d_{ij}}$
4: berechne für alle $\vec{m}_j \in M$: $\vec{m}_j = \frac{1}{\sum_{i=1}^{n} w_{ij}} \cdot \sum_{i=1}^{n} w_{ij} \cdot \vec{x}_i$
5: **end while**
6: **return** M, W

Algorithmus 6.1: k-means Basisalgorithmus

unterscheiden, muss die Anzahl an Iterationen, die die Schleife in Zeile 2 des Algorithmus durchlaufen soll beim Aufruf vorgegeben werden. Im Aktualisierungsschritt wird zuerst für jeden Datenpunkt $\vec{x}_i$ die Distanz zu jedem der k Zentroiden $\vec{m}_j$ ermittelt. Im Gegensatz zu Lloyds k-means Version wird hier die inverse quadratische euklidische Distanz betrachtet. Das Verhältnis von Distanz zu einem Zentroid zur Summe aller Distanzen aller Zentroiden für einen Punkt $\vec{x}_i$ bildet die Gewichtungsmatrix, aus der später durch den Klienten die Zuweisung zu einem Cluster bestimmt wird. Die Zuweisung erfolgt also nicht explizit, sondern implizit während der Iterationen. Die Aktualisierung der Zentroide in Zeile 4 wurde in [63] fälschlicherweise als $\vec{m}_j = \sum_{i=1}^{n} w_{ij} \cdot \vec{x}_i$ definiert. Dies muss für eine sinnvolle Zentroid Wahl mit $\frac{1}{\sum_{i=1}^{n} w_{ij}}$ skaliert werden. Um die multiplikative Tiefe begrenzt zu halten, werden die neu berechneten Zentroiden nach dem Aktualisierungsschritt mithilfe des Kommunikationsprotokolls neu verschlüsselt. Der Algorithmus gibt nach der vorher festgelegten Anzahl an Iterationen die Menge der Zentroide und die Gewichtungsmatrix zurück. Danach ist es Aufgabe der Klienten, jeden ihrer Datenpunkte dem Cluster zuzuordnen, zu dessen Zentroid er die höchste Gewichtung hat.

Um eine Vergleichbarkeit der Rechenoperationen auf verschlüsselten und unverschlüsselten Daten zu ermöglichen, haben wir den Algorithmus sowohl für verschlüsselte als auch für unverschlüsselte Daten implementiert. Auf unverschlüsselten Daten entfällt das Kommunikations- und Divisionsprotokoll, sodass der Server alle Daten ohne jede Kommunikation selbst berechnet. Die Rechenschritte dabei bleiben dieselben.

6.4 Divisionsprotokoll

Da es nötig ist, in Zeile 3 und 4 aus Algorithmus 6.1 Divisionen durchzuführen und die FV Verschlüsselung eine Division nicht auf direktem Weg ermöglicht, nutzen wir ein Divisionsprotokoll nach [63]. Der Server sendet zu diesem Zweck den Nenner m, mit dem dividiert

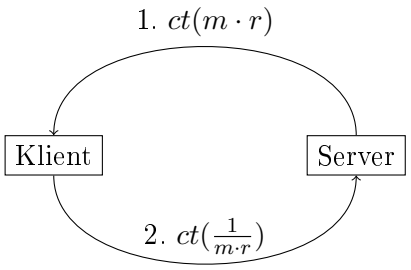


Abbildung 6.2: Divisionsprotokoll.

werden soll, an einen Klienten. Damit der Wert für den Klienten nicht ersichtlich ist, wird dieser mit einem zufälligen Rauschen, analog zu dem Kommunikationsprotokoll, verrechnet. Damit er später von Server herausgerechnet werden kann, wird er multipliziert. Der Klient entschlüsselt dann die übergebene Nachricht, berechnet das multiplikative Inverse $\frac{1}{m \cdot r}$ und gibt den neu berechneten Wert verschlüsselt an den Server zurück. Der Server rechnet anschließend das Rauschen heraus, indem er r, unter Ausnutzen der Homomorphieeigenschaften, mit dem zurückgegebenen Wert multipliziert und so den Chiffretext $ct(\frac{1}{m})$ erhält. Somit ist es möglich, mithilfe einer homomorphen Multiplikation zu dividieren.

In unseren Testläufen werden wir nur einen Klienten implementieren und die übergebenen Nachrichten nicht mit einem Rauschen versehen. Dies ermöglicht uns, die Division durch 0 effektiver durch den Klienten verhindern zu lassen und sorgt für eine Erhöhung der Ergebnisgenauigkeit, die wir im Experimentteil analysieren wollen. Die im Divisionsprotokoll übergebenen Werte werden, falls sie 0 sind, vom Klienten auf einen Shiftparameter s korrigiert.

6.5 Multiplikative Tiefe

Zur Bestimmung der oberen Schranke der multiplikativen Tiefe betrachten wir den Worst-Case des Algorithmus 6.1. Vor der While Schleife findet keine multiplikation statt, womit wir zu Beginn der While Schleife eine multiplikative Tiefe von 0 haben. Im Zuordnungsschritt wird zuerst die Distanzmatrix berechnet. Für jedes Element aus dieser Matrix wird zuerst die quadratische euklidische Distanz $\|\vec{x}_i - \vec{m}_j\|^2 = \sum_{j=1}^{d}(\vec{x}_i - \vec{m}_j)^2$ für $\vec{x}_i, \vec{m}_j \in X$ berechnet. Dann ist die multiplikative Tiefe 1, da jedes Element der Summe genau einmal quadriert wird. Für die Berechnung des multiplikativen Inversen mithilfe des Divisionsprotokolls steigt die multiplikative Tiefe bei der Multiplikation mit dem Rauschen auf 2, sinkt

auf 0 durch den Entschlüsellungsschritt beim Klienten und steigt wieder auf 1 durch das Herausmultiplizieren des Rauschens durch den Server. D.h., die multiplikative Tiefe nach Erzeugen der Distanzmatrix liegt für jedes Element bei 1.

Bei dem Term $\sum_{j=1}^{k} d_{ij}$ der Berechnung der Gewichtungsmatrix bleibt die multiplikative Tiefe bei 1, da keine Multiplikation ausgeführt wird. Die Berechnung des Terms $\frac{1}{\sum_{j=1}^{k} d_{ij}}$ hat, analog zur vorherigen Berechnung, eine multiplikative Tiefe von 1, sodass wir für jedes Element der Gewichtungsmatrix eine multiplikative Tiefe von 2 erhalten, da die beiden Terme multipikativ verrechnet werden.

Beim Zuordnungsschritt entsteht bei der Berechnung des Terms $\frac{1}{\sum_{i=1}^{n} w_{ij}}$ eine multiplikative Tiefe von 1, analog zur vorherigen Abschätzung. In der Summe $\sum_{i=1}^{n} w_{ij} \cdot \vec{x}_i$ werden die jeweiligen Punkte einmal mit dem jeweiligen Gewicht multipliziert. Die Elemente der Gewichtungsmatrix haben dabei eine multiplikative Tiefe von 2, sodass wir durch die Multiplikation zu einer multiplikativen Tiefe von 3 gelangen. Nach der Multiplikation der beiden Terme liegt eine multiplikative Tiefe von 4 vor.

Damit die multiplikative Tiefe nicht unkontrolliert anwächst, lassen wir den neu berechneten Zentroiden durch den Klienten neu verschlüsseln. Das Rauschen wird auf den Zentroid additiv verrechnet und später durch eine Subtraktion wieder herausgerechnet. So erhalten wir für die neu berechneten Zentroiden wieder eine multiplikative Tiefe von 0.

Wir kommen bei unserer k-means Umsetzung also zu einer maximalen multiplikativen Tiefe von 4. Dies ist der Grund warum wir den FV Algorithmus, ein LHE Schema der zweiten Generation gewählt haben. LHE Schemata der dritten Generation, wie z. B. SHIELD, produzieren nach [118] ein höheres Rauschen im ersten Multiplikationsschritt und in den folgenden Multiplikationen weniger Rauschen. Erst ab einer multiplikativen Tiefe von 10 entwickelt sich ein asymptotisch besseres Rauschverhalten.

6.6 Klartextgröße

Der Klartextmodulus t soll so klein wie möglich gewählt werden, da sich ansonsten der Verbrauch des Rausch Budgets und die Laufzeit erhöhen und sich parallel das Rausch Budget verkleinert. Zu diesem Zweck müssen wir die maximale Klartextgröße ermitteln, die während der Berechnungen auftreten kann. Ziel ist es, eine maximale Obergrenze der Klartextgröße zu erhalten. Wir betrachten also den Worst Case Fall.

Bei der Initialisierung werden k verschiedene Punkte als initiale Zentroiden gewählt. Sei x_{max} bzw. x_{min} der Datenpunkt, dessen Elemente alle die maximale bzw. minimale Größe haben, dann ist die Begrenzung der initialen Zentroiden gegeben durch:

$$\begin{aligned} c_{max} &= x_{max} \ , \\ c_{min} &= x_{min} \ . \end{aligned} \tag{6.1}$$

Im Folgenden betrachten wir den allgemeinen Fall der Iterationen der While Schleife und verwenden daher c_{max} bzw. c_{min} als Begrenzung der Zentroiden und nicht x_{max} bzw. x_{min}. Bei der Berechnung der Distanzmatrix wird zuerst die quadratische euklidische Distanz $\|x_i - c_j\|^2$ für alle x_i und c_j einzeln berechnet. Durch Einsetzen der Definition und der oben definieren Grenzwerte erhalten wir folgende Abschätzung:

$$\max_{i,j}(\|x_i - c_j\|^2) = \max_{i,j}(\sum_{k=1}^{d}(x_{i_k} - c_{j_k})^2) \leq \sum_{k=1}^{d}(x_{max_k} - c_{min_k})^2 \ .$$

Da $\sum_{k=1}^{d}(x_{max_k} - c_{min_k})^2$ d-Mal die gleichen Werte summiert, folgt direkt der obere Grenzwert dist_{max}:

$$\text{dist}_{max} = \sum_{k=1}^{d}(x_{max_k} - c_{min_k})^2 = d \cdot (x_{max} - c_{min})^2 \ . \tag{6.2}$$

Der untere Grenzwert dist_{min} unterscheidet sich etwas in der Herleitung. Dieser wird minimal, wenn der Zentroid und der Datenpunkt aufeinander liegen. Dies tritt im Algorithmus nur bei der Initialwahl auf und wird durch Aufaddieren eines kleinen Wertes s (Shift Parameter, vgl. Kapitel 6.3) ausgeglichen. Bei aufeinanderliegenden Punkten beträgt der quadratische euklidische Abstand unserer gesuchten Untergrenze:

$$\text{dist}_{min} = d \cdot s^2 \ . \tag{6.3}$$

Die einzelnen Elemente der Distanzmatrix sind die jeweilige Inverse zum quadratischen euklidischen Abstand. Diese werden maximal, wenn die Distanz minimal wird und umgekehrt. Daraus folgen direkt die beiden Grenzwerte:

$$\begin{aligned} d_{max} &= \frac{1}{\text{dist}_{min}} \ , \\ d_{min} &= \frac{1}{\text{dist}_{max}} \ . \end{aligned} \tag{6.4}$$

Zur Berechnung der Grenzwerte für die Elemente der Gewichtungsmatrix müssen wir zuerst die Summe über die Distanzen von einem Punkt zu jedem Zentroid $\sum_{j=1}^{k} d_{ij}$ abschätzen. Dies geschieht durch Einsetzen der Gleichung 6.4 und Umformen:

$$k \cdot d_{min} = \sum_{j=1}^{k} d_{min} \leq \sum_{j=1}^{k} d_{ij} \leq \sum_{j=1}^{k} d_{max} = k \cdot d_{max} \ . \tag{6.5}$$

Die Grenzwerte für die Elemente der Gewichtungsmatrix ergeben sich direkt durch Einsetzen von Gleichung 6.4 und 6.5 in $\frac{d_{ij}}{\sum_{j=1}^{k} d_{ij}}$ unter dem Aspekt, dass die Elemente maximal bzw. minimal werden genau dann, wenn die Distanz zum jeweiligen Zentroid maximal bzw. minimal ist und Summe über alle k Distanzen maximal bzw. minimal ist.

$$\begin{aligned} w_{max} &= \frac{d_{max}}{k \cdot d_{min}} \ , \\ w_{min} &= \frac{d_{min}}{k \cdot d_{max}} \ . \end{aligned} \tag{6.6}$$

Schließlich betrachten wir die beiden Teilterme zur Aktualisierung der Zentroiden. Die Abschätzung des Nenners des Skalierungsfaktors folgt direkt durch Einsetzen von Gleichung 6.6 in $\sum_{i=1}^{d} w_{ij}$ und Umstellen der erhaltenen Gleichungen:

$$d \cdot w_{min} = \sum_{i=1}^{d} w_{min} \leq \sum_{i=1}^{d} w_{ij} \leq \sum_{i=1}^{d} w_{max} = d \cdot w_{max} \ . \tag{6.7}$$

Da $\sum_{i=1}^{d} w_{ij} \cdot \vec{x}_i$ maximal bzw. minimal wird, wenn w_{ij} und $\vec{x}_i$ maximal bzw. minimal werden, folgt durch Einsetzen von Gleichung 6.6 in die Gleichung und Umstellung folgende Abschätzung:

$$\begin{aligned} d \cdot w_{min} \cdot x_{min} &= \sum_{i=1}^{d} w_{min} \cdot x_{min} \leq \sum_{i=1}^{d} w_{ij} \cdot x_i \ , \\ d \cdot w_{max} \cdot x_{max} &= \sum_{i=1}^{d} w_{max} \cdot x_{max} \geq \sum_{i=1}^{d} w_{ij} \cdot x_i \ . \end{aligned} \tag{6.8}$$

Durch die unterschiedliche Definition von dist_{max} und dist_{min} und der starken Verallgemeinerung auf die Extremwerte der Datenpunkte lassen sich diese Gleichungen nicht auf die Berechnung des nächsten Zentroiden anwenden. Da er Algorithmus korrekte Ergebnisse liefert, können wir davon ausgehen, dass die neu berechneten Zentroiden wieder durch x_{max} und x_{min} begrenzt sind.

Die Parameter der FV Verteilung müssen mindestens so groß gewählt werden, dass die größte berechnete Obergrenze der oben vorgestellten Grenzwerte verschlüsselt dargestellt werden kann, da sonst unter Umständen keine sinnvolle Entschlüsselung möglich ist. In Tabelle 6.1 fassen wir die Abschätzungen in einem Überblick zusammen.

Gleichung	Formel
$x_{max}, c_{max}/x_{min}, c_{min}$	
dist_{max}	$d \cdot (x_{max} - c_{min})^2$
dist_{min}	$d \cdot s^2$
d_{max}	$\frac{1}{\text{dist}_{min}}$
d_{min}	$\frac{1}{\text{dist}_{max}}$
Term 1 Max	$k \cdot d_{max}$
Term 1 Min	$k \cdot d_{min}$
w_{max}	$\frac{d_{max}}{k \cdot d_{min}}$
w_{min}	$\frac{d_{min}}{k \cdot d_{max}}$
Term 2 Max	$d \cdot w_{max}$
Term 2 Min	$d \cdot w_{min}$
Term 3 Max	$d \cdot w_{max} \cdot x_{max}$
Term 3 Min	$d \cdot w_{min} \cdot x_{min}$

Tabelle 6.1: Übersicht: Formeln zur Bestimmung der Klartextgröße. d: Dimension des Datensatzes, k: Anzahl Cluster, x_{max}, c_{max}: Datenpunkte mit maximaler Merkmalgröße, x_{min}, c_{min}: Datenpunkte mit minimaler Merkmalgröße.

6.7 Beispieldaten

Wir werden unsere Experimente an zwei verschiedenen horizontalen Datensätzen durchführen. Bei horizontalen Daten liegen für jeden Datenpunkt alle Ausprägungen vor. Einer davon ist öffentlich zugänglich, der andere stammt aus einem Forschungsprojekt. Somit ist teilweise eine barrierefreie Replikation unserer Experimente möglich.

6.7.1 Datensatz 1: Schwertlilien (Iris) Blätter

Bei dem Datensatz handelt es sich um den oft zum Testen von Klassifikationsmethoden verwendeten multivariaten Datensatz der Schwertlilien (Iris) aus dem UCI - Machine Learning Repository[2]. Er wurde 1936 von Sir Ronald Aylmer Fisher in [69] als Beispiel für eine Diskriminanzanalyse vorgestellt. Der Datensatz besteht aus 150 Stichproben von 3 verschiedenen Schwertlilien Arten (Iris setosa, Iris versicolour und Iris virginica). Folgende Merkmale werden zur Verfügung gestellt:

1. Länge der Kelchblätter (lat.: Sepalen)
2. Breite der Kelchblätter
3. Länge der Kronblätter (lat.: Petalen)

[2] `https://archive.ics.uci.edu/ml/datasets/iris`

4. Breite der Kronblätter

5. Schwertlilien Art

Die Längen- und Breitenangaben sind in cm angegeben mit jeweils maximal einer Nachkommastelle. Der größte Wert liegt bei 7.1cm. Abbildung 6.3 zeigt die Lage der Kelch- und Kronblätter einer Schwertlilie. Durch die Angabe der Schwertlilien Art wird eine Validierung eines (testweise durchgeführten) Clusterergebnisses möglich. Die Clusteranalyse in den Experimenten wird über die jeweiligen Längen- und Breitenangaben erfolgen.

Abbildung 6.3: Iris Daten: Kronblatt (gelb) und Kelchblatt (rot) einer Schwertlilie. Quelle: https://pixabay.com/de/schwertlilie-iris-blume-flora-blau-2339883/

6.7.2 Datensatz 2: Verkehrsdaten aus Dublin

Für diese Bachelorarbeit wurden uns ein Teil der bereits vorverarbeiteten Daten aus [106] freundlicherweise zur Verfügung gestellt. Hierbei handelt es sich um Verkehrsdaten, die vom 01.01.2013 bis zum 14.05.2013 durch das "Sydney Coordinated Adaptive Traffic System" (SCATS) über Sensoren aufgezeichnet wurden. Zur Verfügung stehen uns davon die Daten aus Januar 2013.

SCATS stellt über 750 Sensoren Beobachtungen in Form von Zeitreihen über den Verkehr in Dublin bereit (vgl. Kapitel 5.1). In der Vorverarbeitung in [106] wurden diese zu 296 Sensorknoten zusammengefasst und diskretisiert. Das aufgezeichnete Verkehrsaufkommen wurde in zwei Datensätzen mit je 15 bzw. 30 Minuten Abständen aggregiert. Die einzelnen Messwerte liegen zwischen 0 und 1 mit bis zu 15 Nachkommastellen. In dem Datensatz wurden die Sensorknoten als Zeilen gespeichert. Die anfänglichen Spalten beinhalten die jeweiligen aggregierten Messungen zum Verkehrsaufkommen, gefolgt von einer eindeutigen ID, der Anzahl an Sensoren, die zu dem Sensorknoten zusammengefasst wurden, den Koordinaten (Latitude und Longitude) sowie einem Kürzel der Zugehörigkeit zu einem Stadtteil von Dublin (ccity, ncity, scity und wcity).

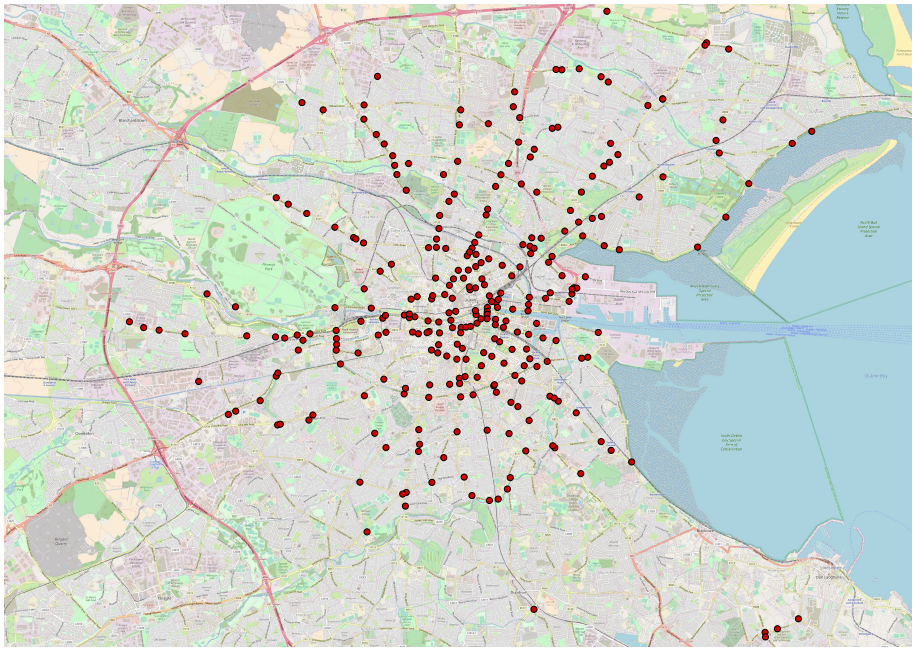


Abbildung 6.4: Dublin Daten: Sensorenstandorte der 296 zusammengefassten Sensoren.

Bei dem Datensatz mit 15 minütigen Abständen kommen wir so auf 2976 Messdaten. Bei dem Datensatz mit 30 minütigen Abständen auf 1488 Messdaten. Die Anzahl an Messdaten gilt dabei für jeden Datenpunkt. Die Clusteranalyse in den Experimenten wird über die Messwerte erfolgen. Abbildung 6.4 zeigt den Standort der 296 Sensorknoten in Dublin.

Kapitel 7

Experimente

Die Präsentation der Ergebnisse richtet sich nach der Reihenfolge der ersten drei Forschungsfragen. Laufzeiten werden durchgängig zugunsten einer kompakten und leichter vergleichbaren Darstellung im Format "Stunden : Minuten : Sekunden" (z. B. 00:01:54.01) angegeben. Die folgenden Experimente beziehen sich zum einen auf den Schwertliliendatensatz, der Iris Daten, (Kapitel 6.7.1) und zum anderen auf den Dublindatensatz (Kapitel 6.7.2), der Dublin Daten. Der Sinn der Experimente ist es, die Auswirkungen der verschiedenen Parameter der FV Verschlüsselung auf die Laufzeit und die Ergebnisgenauigkeit zu betrachten. Alle Experimente werden in Abhängigkeit des Polynommodulus und dem daraus resultierenden Koeffizienten- und Klartextmodulus untersucht. Zusätzlich wird anhand der Iris Daten die Auswirkung des Parameters T, der Basis der Relinearisierung und anhand der Dublin Daten die Auswirkung der Merkmalsanzahl bzw. Größe der Dimensionen der einzelnen Datenpunkte dokumentiert.

7.1 Aufwand der Parameterbestimmung

Die erste Forschungsfrage lautete: "Wie aufwendig ist die Parameterbestimmung?". Wir müssen die Parameter für unsere implementierte Version von k-means für unverschlüsselte Daten bestimmen. Diesen Algorithmus nennen wir im Folgenden den Basisalgorithmus. Die Iris Daten haben immer eine Dimension von 4. Die Dublin Daten dagegen zwischen 48 und 336. Dies erklärt sich dadurch, dass der uns zur Verfügung stehende Arbeitsspeicher nicht ausreicht, um die großen Datenmengen zu verschlüsseln. In dem Fall bricht der Algorithmus ohne Fehlermeldung ab. Um den Datensatz in kleinere Mengen zu unterteilen, reduzieren wir die Dimensionen auf einen bis sieben Tage. Das entspricht 48 Messungen für einen Tag bei dem Dublin Datensatz mit 30 minütigen Abständen. Alle verschlüsselten Testreihen haben das Ziel, eine hohe Sicherheit von 128 Bit zu garantieren (vgl. Kapitel 2.3.3). Unser Polynommodulus muss bei mindestens $x^{1024} + 1$ liegen, um diese Sicherheit zu garantieren

[33]. Den passenden Koeffizientenmodulus q lassen wir gemäß Tabelle 4.3 durch SEAL bestimmen.

7.1.1 Anzahl Iterationen: k-means auf unverschlüsselten Daten

Um die nötige Anzahl an Iterationen herauszufinden, wenden wir den von uns ebenfalls implementierten k-means Algorithmus nach Lloyd auf die unverschlüsselten Iris Daten und Dublin Daten an. Dabei lassen wir uns die benötigte Anzahl an Durchläufen ausgeben. Dies ist nötig da es nicht möglich ist, auf den verschlüsselten Daten zu bestimmen, ob die Zuordnung konvergiert (vgl. Kapitel 6.3).

Für den Iris Datensatz wählen wir $k = 3$, da sich der Datensatz in drei Klassen unterteilt. Bei dem Dublin Datensatz halten wir uns an [106], die mit $k = 15$ die Clusteranalyse berechnet haben. Aufgrund der Vergleichbarkeit führen wir die Clusteranalyse zusätzlich für $k = 3$ durch. In Tabelle 7.1 befinden sich die Ergebnisse dieser Testdurchläufe.

d	Datensatz	$k = 3$	$k = 15$
4	Iris	12	-
48	Dublin	10	20
96		13	09
144		06	06
192		08	07
240		06	06
288		06	08
336		09	08

Tabelle 7.1: Parameterfindung: Benötigte Anzahl an Iterationen des k-means Algorithmus nach Lloyd. d: Dimension des Datensatzes, k: Klassen, -: nicht ausgeführt.

Um eine Vergleichbarkeit sicherzustellen, initialisieren wir alle Testläufe mit denselben initialen Zentroiden. Die Wahl für den Iris Datensatz besteht aus dem 1-ten, 60-ten und 149-ten Datenpunkt, sodass jeweils ein Datenpunkt jeder Schwertlilien Art enthalten ist. Für $k = 3$ bei den Dublin Daten wählen wir den 1-ten, 90-ten und 170-ten Datenpunkt und für $k = 15$ den 19-ten, 38-ten, 57-ten, 76-ten, 95-ten, 114-ten, 133-ten, 152-ten, 171-ten, 190-ten, 209-ten, 228-ten, 247-ten, 266-ten und 285-ten Datenpunkt.

7.1.2 Parameterfindung Iris Daten

Um die weiteren Parameter zu finden, berechnen wir die Klartextgröße gemäß der in Kapitel 6.6 hergeleiteten Schranken für $k = 3$ und $d = 4$. Da die Ausprägungen der Iris Daten maximal eine Nachkommastelle haben, wählen wir $s = 0.1$. Die Ergebnisse werden in Tabelle 7.2 präsentiert. Somit liegt die obere Schranke für die Zahlengröße bei 2 403 180.

Diese wird selbst bei der kleinsten Basis von 2 nicht beschränkt. Da wir Gleitkommazahlen für die Division benötigen, verwenden wir den Fractional Encoder. Den ganzzahligen Anteil begrenzen wir auf 14 Bit, wobei mit einer Basis von $B = 3$ alle Zahlen bis 4 782 969 darstellbar sind, was die obere Schranke mit einschließt.

Berechnung	Schranke
x_{max}, c_{max}	7.9
x_{min}, c_{min}	0.1
$\text{dist}_{max} = d \cdot (x_{max} - c_{min})^2$	9126
$\text{dist}_{min} = d \cdot s^2$	1.5
$d_{max} = \frac{1}{\text{dist}_{min}}$	0.6667
$d_{min} = \frac{1}{\text{dist}_{max}}$	0.0001
$k \cdot d_{max}$ (Max)	2
$k \cdot d_{min}$ (Min)	0.0003
$w_{max} = \frac{d_{max}}{k \cdot d_{min}}$	2028
$w_{min} = \frac{d_{min}}{k \cdot d_{max}}$	5.4789
$d \cdot w_{max}$ (Max)	304200
$d \cdot w_{min}$ (Min)	0.0082
$d \cdot w_{max} \cdot x_{max}$ (Max)	2403180
$d \cdot w_{min} \cdot x_{min}$ (Min)	0.0008

Tabelle 7.2: Iris Daten: Klartextgröße für $d = 4$, $s = 0.1$ und $k = 3$. d: Dimension des Datensatzes, s: Shift Parameter, k: Anzahl Cluster, x_{max}, c_{max}: Datenpunkte mit maximaler Merkmalgröße, x_{min}, c_{min}: Datenpunkte mit minimaler Merkmalgröße.

Die Begrenzung für den Teil der Nachkommastellen zu wählen ist aufwändiger. Sie muss, wie in Kapitel 4.7.2 beschrieben, so klein wie möglich gehalten werden, um ein Überschreiben der Bereiche der ganzen Zahlen des Polynoms durch die Bereiche der Nachkommastellen zu vermeiden. Gleichzeitig wird so aber die Genauigkeit der Berechnungen vermindert. Wir setzen den Bereich der Nachkommastellen auf eine Schätzung von 256 Bit. Nach dem Herleiten der restlichen Parameter werden wir Testläufe durchführen, um zu überprüfen wie weit sich dieser reduzieren lässt.

Um bei diesen Testläufen die Vergleichbarkeit zu garantieren, wählen wir das T als $\lceil\sqrt{q}\rceil$, wie in Kapitel 4.6 beschrieben. Nach Tabelle 4.3 ist die Bitlänge q in unserem Testreihen am größten für $n = 8192$. Dies entspricht dem Wert 219 und wir kommen so auf $T = 15$. Bei anfänglichen Verschlüsselungstests hat sich für T gezeigt, dass bei $n = 8192$ die Basis mindestens $T = 4$ sein muss.

Problematisch wurde, anders als erwartet, an dieser Stelle die Bestimmung des Klartextmodulus t. Wie sich in verschiedenen Testläufen herausgestellt hat, muss dieser für

sinnvolle Ergebnisse bei unseren Untersuchungen dem Parameter n des Polynommodulus entsprechen (vgl. Kapitel 7.1.3).

Um die Basis B des Fractional Encoders zu bestimmen, haben wir mehrere Testläufe mit den Parametern aus Tabelle 7.3 laufen lassen. Als Werte halten wir uns an die Empfehlungen aus [33] und vergleichen die zwei Fälle $B = 2$ und $B = 3$. Für $B = 2$ muss der Anteil der ganzen Zahlen auf $B = 22$ Bit angehoben werden.

Parameter	Konfiguration
Polynommodulus	$x^{4096} + 1$
Koeffizientenmodulus	coeff_modulus_128(4096)= 110
Klartextmodulus	$t = 4096$
Basis (Relinearisierung)	$T = 15$
Anteil Ganzzahlig	$n_i = 14$ $(B = 3)$,
	$n_i = 22$ $(B = 2)$
Anteil Nachkommastellen	$n_f = 256$

Tabelle 7.3: Iris Daten: Parameter für Testläufe zu Parameter B. B: Basis des Fractional Encoders.

Das durchschnittliche Ergebnis der Testläufe findet sich in Tabelle 7.4. Da die Laufzeit für $B = 3$ im Schnitt schneller ist, entscheiden wir uns für diese Basis.

B	durchschnittliche Laufzeit
2	00:01:14.08
3	00:01:13.71

Tabelle 7.4: Iris Daten: Laufzeitvergleich mit $B = 2$ und $B = 3$.

Zur Bewertung der Genauigkeit haben wir die Iris Daten bei jeder Rechenoperation durch den Klienten entschlüsseln und überprüfen lassen. Das Ergebnis der Testdurchläufe ändert sich ab $n_f = 64$ hinsichtlich der Genauigkeit nicht mehr. Für weiter begrenzte Nachkommastellen verschlechtert sich die Abweichung vom korrekten Ergebnis der einzelnen Rechenoperationen. Die Wahl der Begrenzungen im Fractional Encoder hat zu keinen messbaren Laufzeitunterschieden geführt. Tabelle 7.5 zeigt die ermittelten Parameter für unsere weiteren Experimente auf dem Iris Datensatz.

Parameter	Konfiguration
Polynommodulus	$x^n + 1$, $n \in \{1024, 2048, 4096, 8192\}$
Koeffizientenmodulus	coeff_modulus_128(n)
Klartextmodulus	$t = n$
Basis (Relinearisierung)	$4 \leq T \leq 60$
Anteil Ganzzahlig	$n_i = 14$
Anteil Nachkommastellen	$n_f = 64$
Basis (Encoder)	$B = 3$ Bit

Tabelle 7.5: Iris Daten: Parameter für die weiteren Experimente.

7.1.3 Bemerkung zur Bestimmung der Parameter

Im Laufe der Parameterbestimmung haben wir wesentlich mehr Testläufe machen müssen, durch die weitgehend freie Wahlmöglichkeit der Parameter. Die meisten dieser Testläufe führten zu keinen verwertbaren Ergebnissen.

Es hat sich bei den Testläufen zur Parameterbestimmung des Klartextmodulus gezeigt, dass bei einer Basis $B = 2$ des Fractional Encoders $t > 2$ gewählt werden muss, da ansonsten alle Zahlen negativ verschlüsselt werden. Ab $t > 9$ wurden Zentroiden und Gewichtungen so berechnet, dass sie nicht nur aus Nullen bestanden. Zu dieser Zeit der Testverfahren war nicht klar, warum die Ergebnisse dennoch nicht vergleichbar bzw. wiederholbar waren. Es hat sich später herausgestellt, dass wir nur sinnvolle Ergebnisse erhalten können, wenn der Klartextmodulus mit dem n des Polynommodulus übereinstimmt, obwohl er nach unserer Abschätzung der Klartextgröße und nach [33] frei wählbar sein müsste.

Da es bei den Tests zum Anteil der Nachkommastellen zu keinem messbaren durchschnittlichen Laufzeitunterschied gekommen ist, kann dieser auch größer geschätzt werden, solange kein Überlauf produziert wird.

7.1.4 Parameterfindung Dublin Daten

Aufgrund der am Anfang des Kapitels erwähnten Abbruchsproblematik können wir für den Dublin Datensatz den Polynommodulus $x^{8192}+1$ nicht verwenden (vgl. Kapitel 7.1.5). Wir berechnen die Klartextgröße gemäß der in Kapitel 6.6 hergeleiteten Schranken für $k = 3$ und $d = 336$. Da die Ausprägungen der Dublin Daten maximal 15 Nachkommastellen haben, wählen wir $s = 0.1 \cdot 10^{-15}$. Für diese Konfiguration treten die größten Zahlen auf. Die Klartextgröße nimmt für kleinere d und größere k ab. Die Ergebnisse werden in Tabelle 7.6 präsentiert. Analog zu Kapitel 7.1.2 wählen wir für die Begrenzung des ganzzahligen Anteils im Fractional Encoder 68 Bit. Damit lassen sich alle Zahlen bis $2.78 \cdot 10^{32}$ darstellen, was die obere Schranke $1.12 \cdot 10^{32}$ bei unseren Berechnungen mit einschließt.

Berechnung	Schranke
x_{max}, c_{max}	1.0
x_{min}, c_{min}	0.0
$\text{dist}_{max} = d \cdot (x_{max} - c_{min})^2$	336
$\text{dist}_{min} = d \cdot s^2$	$3.3600000000000003 \cdot 10^{-28}$
$d_{max} = \frac{1}{\text{dist}_{min}}$	$2.976190476190476 \cdot 10^{27}$
$d_{min} = \frac{1}{\text{dist}_{max}}$	0.002976190476190476
Term 1 Max $= k \cdot d_{max}$	$8.928571428571427 \cdot 10^{27}$
Term 1 Min $= k \cdot d_{min}$	0.008928571428571428
$w_{max} = \frac{d_{max}}{k \cdot d_{min}}$	$3.333333333333333 \cdot 10^{29}$
$w_{min} = \frac{d_{min}}{k \cdot d_{max}}$	$3.3333333333333338 \cdot 10^{-31}$
Term 2 Max $= d \cdot w_{max}$	$1.1199999999999999 \cdot 10^{32}$
Term 2 Min $= d \cdot w_{min}$	$1.12 \cdot 10^{-28}$
Term 3 Max $= d \cdot w_{max} \cdot x_{max}$	$1.1199999999999999 \cdot 10^{32}$
Term 3 Min $= d \cdot w_{min} \cdot x_{min}$	0.0

Tabelle 7.6: Dublin Daten: Klartextgröße für $d = 336$, $s = 0.1 \cdot 10^{-15}$ und $k = 3$. d: Dimension des Datensatzes, s: Shift Parameter , k: Anzahl Cluster, x_{max}, c_{max}: Datenpunkte mit maximaler Merkmalgröße, x_{min}, c_{min}: Datenpunkte mit minimaler Merkmalgröße.

Analog zu Kapitel 7.1.2 trat für die Begrenzung der Nachkommastellen auf 64 Bit keine Abweichung auf. Die restlichen Parameter für diese Testläufe haben wir aus den Parametern für den Iris Datensatz übernommen (vgl. Tabelle 7.5). Aufgrund der Übersichtlichkeit haben wir uns entschieden, auf den Dublin Daten keinen Vergleich des Parameters T durchzuführen. Um eine Vergleichbarkeit zu gewähren, wählen wir ein festes $T = 15$ wie in den Iris Daten. Vorherige Testläufe für diesen Parameter haben ein ausreichendes Rausch Budget bei moderaten Laufzeiten ergeben. Tabelle 7.7 zeigt die ermittelten Parameter für die weiteren Experimente auf dem Dublin Datensatz.

Parameter	Konfiguration
Polynommodulus	$x^n + 1$, $n \in \{1024, 2048, 4096\}$
Koeffizientenmodulus	coeff_modulus_128(n)
Klartextmodulus	$t = n$
Basis (Relinearisierung)	$T = 15$
Anteil Ganzzahlig	$n_i = 68$
Anteil Nachkommastellen	$n_f = 64$
Basis (Encoder)	$B = 3$ Bit

Tabelle 7.7: Dublin Daten: Parameter für die weiteren Experimente.

7.1.5 Auswirkungen der Arbeitsspeicherbegrenzung

Um die Auswirkung des Arbeitsspeichers auf die Verschlüsselung zu untersuchen, begrenzten wir den Speicher für den Docker Container auf 2 GB, denn mit dem Maximum von 11.2 GB Arbeitsspeicher war es nicht möglich, die Dublin Daten für $n = 8192$ zu verschlüsseln. Der Algorithmus bricht für jedes der gewählten d bei den Dublin Daten ab. Tabelle 7.8 zeigt, welche der Verschlüsselungsversuche in diesem Test mit 2 GB Arbeitsspeicher zum Abbruch führten. Abgebrochene Durchläufe wurden bei der Tabelle mit × gekennzeichnet und gelungene mit ✓. Einträge, die mit (✓) markiert wurden, verschlüsselten erst, wenn keine weiteren Verschlüsselungen vorweg ausgeführt wurden.

d	$n = 1024$	$n = 2048$	$n = 4096$	$n = 8192$
48	✓	✓	✓	×
96	✓	✓	×	×
144	✓	✓	×	×
192	✓	(✓)	×	×
240	✓	(✓)	×	×
284	✓	×	×	×
336	✓	×	×	×

Tabelle 7.8: Abbruchproblematik: Verschlüsselungsversuche mit 2 GB Arbeitsspeicher, ×: abgebrochen, ✓: gelungen, (✓): einzeln gelungen.

7.2 Laufzeitkosten der FV Verschlüsselung

In den folgenden Testreihen lassen wir unseren Basisalgorithmus auf den unverschlüsselten und verschlüsselten Iris Daten und Dublin Daten laufen. So erhalten wir einen Vergleichswert, um beurteilen zu können, wie sich die homomorphen Rechenoperationen auf die jeweiligen Laufzeiten auswirken sowie eine Cluster Referenz, mit der wir die Genauigkeit der Clusteranalyse auf den verschlüsselten Daten vergleichen können. Zusätzlich dokumentieren wir die Laufzeitergebnisse über den k-means Algorithmus nach Lloyd aus Kapitel 7.1.1.

7.2.1 Unverschlüsselte Iris Daten und Dublin Daten

Analog zu den Erläuterungen aus Kapitel 7.1.1 lassen wir die Algorithmen auf dem Dublin Daten für $k = 3$ und $k = 15$ und auf den Iris Daten für $k = 3$ laufen. In Tabelle 7.9 und 7.10 befinden sich die Ergebnisse dieser Testdurchläufe.

d		$k = 3$	$k = 15$
48	Laufzeit	00:00:00.57	00:00:04.50
	Laufzeit Lloyd	00:00:00.13	00:00:00.87
96	Laufzeit	00:00:01.30	00:00:03.81
	Laufzeit Lloyd	00:00:00.33	00:00:00.77
144	Laufzeit	00:00:00.87	00:00:03.75
	Laufzeit Lloyd	00:00:00.22	00:00:00.76
192	Laufzeit	00:00:01.49	00:00:05.79
	Laufzeit Lloyd	00:00:00.40	00:00:01.18
240	Laufzeit	00:00:01.35	00:00:06.15
	Laufzeit Lloyd	00:00:00.37	00:00:01.27
288	Laufzeit	00:00:01.58	00:00:09.93
	Laufzeit Lloyd	00:00:00.46	00:00:02.06
336	Laufzeit	00:00:02.74	00:00:11.64
	Laufzeit Lloyd	00:00:00.80	00:00:02.41

Tabelle 7.9: Dublin Daten (unverschlüsselt): Laufzeiten. d: Dimension des Datensatzes, k: Klassen, Laufzeit: Dauer des Basisalgorithmus, Laufzeit Lloyd: Dauer des Lloyds k-means Algorithmus, Durchläufe: benötigte Anzahl Durchläufe von Lloyds k-means Algorithmus und Anzahl Iterationen des Basisalgorithmus.

Messung	Laufzeit für $k = 3$
Laufzeit	00:00:00.11
Laufzeit Lloyd	00:00:00.01

Tabelle 7.10: Iris Daten (unverschlüsselt): Laufzeiten. k: Klassen, Laufzeit: Dauer des Basisalgorithmus, Laufzeit Lloyd: Dauer des Lloyds k-means Algorithmus, Durchläufe: benötigte Anzahl Durchläufe von Lloyds k-means Algorithmus und Anzahl Iterationen des Basisalgorithmus.

Abbildung 7.1 visualisiert den Laufzeitunterschied der verschiedenen Testläufe. Die Laufzeitunterschiede zwischen Lloyds Algorithmus (blau) und unserem Basisalgorithmus (grün) auf den Dublin Daten lassen sich gut erkennen. Die Laufzeit für die Clusteranalysen beider Algorithmen auf den Iris Daten haben wir symbolisch als rote Kurve über alle gewählten Dimensionen hinzugefügt, obwohl diese nur für $d = 4$ durchgeführt wurde.

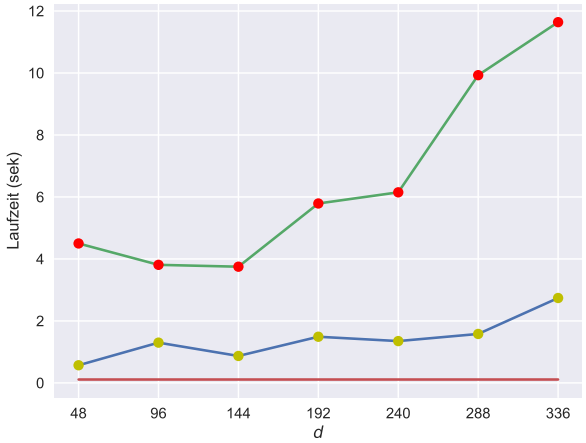


Abbildung 7.1: Laufzeitenvergleich Clusteranalyse auf den unverschlüsselten Datensätzen. blau: Dublin Daten (Llody), grün: Dublin Daten (Basisalgorithmus), rot: Iris Daten (bei $d = 4$), Laufzeit: Sekunden.

7.2.2 Iris Datensatz

Da der Parameter T der Relinearisierung die Laufzeit und Nutzung des Rausch Budgets beeinflussen kann, haben wir vier verschiedene Testreihen durchgeführt. In der ersten Testreihe, deren Ergebnisse in Tabelle 7.11 dokumentiert sind, setzen wir $T = 15$ analog zu den Erläuterungen in Kapitel 7.1.2. Für die zweite Testreihe, Tabelle 7.12, setzen wir unterschiedliche $T = \lceil \sqrt{q} \rceil$ in Abhängigkeit von q, nach der Empfehlung aus [67] ein. Die dritte und vierte Testreihe, Tabellen 7.13 und 7.14, basieren auf dem minimalen und maximalen T. In jeder Testreihe wurde die Laufzeit der Clusteranalyse jeweils für die vier verschiedenen Polynommoduli dokumentiert. Zudem betrachten wir jeweils den Fall einer Iteration sowie der kompletten Clusteranalyse mit 12 Iterationen (vgl. Kapitel 7.1.1).

n	Messung	Laufzeit für: • 1 Iter. • $k = 3$	Laufzeit für: • 12 Iter. • $k = 3$
1024	Verschlüsseln	00:00:00.51	00:00:00.50
	k-means	00:00:09.39	00:01:54.01
	Entschlüsseln	00:00:00.07	00:00:00.08
	Gesamt	00:00:09.97	00:01:54.59
2048	Verschlüsseln	00:00:01.01	00:00:01.08
	k-means	00:00:19.65	00:03:57.93
	Entschlüsseln	00:00:00.14	00:00:00.15
	Gesamt	00:00:20.80	00:03:59.16
4096	Verschlüsseln	00:00:02.33	00:00:02.32
	k-means	00:01:13.29	00:14:43.87
	Entschlüsseln	00:00:00.46	00:00:00.46
	Gesamt	00:01:16.08	00:14:46.65
8192	Verschlüsseln	00:00:05.97	00:00:05.95
	k-means	00:05:16.86	01:03:33.50
	Entschlüsseln	00:00:01.70	00:00:01.70
	Gesamt	00:05:24.53	01:03:41.15

Tabelle 7.11: Iris Daten (verschlüsselt): Laufzeiten mit $T = 15$. Iter.: Iteration, k: Anzahl Cluster, T: Basis der Relinearisierung, n: Parameter des Polynommodulus $(x^n + 1)$.

n	Messung	Laufzeit für: • 1 Iter. • $k = 3$	Laufzeit für: • 12 Iter. • $k = 3$
1024	Verschlüsseln	00:00:00.55	00:00:00.51
	k-means	00:00:10.01	00:01:57.92
	Entschlüsseln	00:00:00.07	00:00:00.07
	Gesamt	00:00:10.63	00:01:58.50
2048	Verschlüsseln	00:00:01.07	00:00:01.01
	k-means	00:00:20.76	00:04:05.12
	Entschlüsseln	00:00:00.18	00:00:00.14
	Gesamt	00:00:22.01	00:04:06.27
4096	Verschlüsseln	00:00:02.54	00:00:02.32
	k-means	00:01:15.90	00:15:02.43
	Entschlüsseln	00:00:00.46	00:00:00.49
	Gesamt	00:01:18.90	00:15:05.24
8192	Verschlüsseln	00:00:06.46	00:00:05.89
	k-means	00:05:15.80	01:02:30.15
	Entschlüsseln	00:00:01.69	00:00:01.66
	Gesamt	00:05:23.95	01:02:37.70

Tabelle 7.12: Iris Daten (verschlüsselt): Laufzeiten mit $T = \lceil\sqrt{q}\rceil$. Iter.: Iteration, k: Anzahl Cluster, T: Basis der Relinearisierung, n: Parameter des Polynommodulus $(x^n + 1)$.

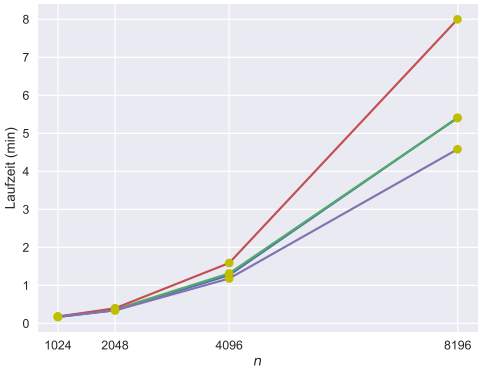
n	Messung	Laufzeit für: • 1 Iter. • $k = 3$	Laufzeit für: • 12 Iter. • $k = 3$
1024	Verschlüsseln	00:00:00.55	00:00:00.51
	k-means	00:00:10.19	00:02:00.51
	Entschlüsseln	00:00:00.07	00:00:00.07
	Gesamt	00:00:10.81	00:02:01.09
2048	Verschlüsseln	00:00:01.06	00:00:01.01
	k-means	00:00:22.07	00:04:21.63
	Entschlüsseln	00:00:00.15	00:00:00.14
	Gesamt	00:00:23.91	00:04:22.78
4096	Verschlüsseln	00:00:02.52	00:00:02.31
	k-means	00:01:32.33	00:18:17.39
	Entschlüsseln	00:00:00.46	00:00:00.46
	Gesamt	00:01:35.31	00:18:20.16
8192	Verschlüsseln	00:00:06.39	00:00:06.01
	k-means	00:07:51.78	01:34:14.42
	Entschlüsseln	00:00:01.67	00:00:01.67
	Gesamt	00:07:59.84	01:34:22.01

Tabelle 7.13: Iris Daten (verschlüsselt): Laufzeiten mit $T = 4$. Iter.: Iteration, k: Anzahl Cluster, T: Basis der Relinearisierung, n: Parameter des Polynommodulus $(x^n + 1)$.

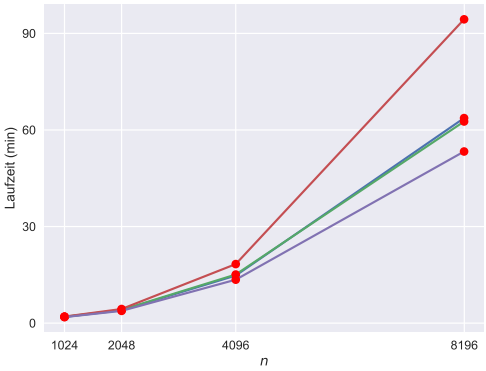
n	Messung	Laufzeit für: • 1 Iter. • $k = 3$	Laufzeit für: • 12 Iter. • $k = 3$
1024	Verschlüsseln	00:00:00.50	00:00:00.50
	k-means	00:00:09.50	00:01:51.48
	Entschlüsseln	00:00:00.07	00:00:00.07
	Gesamt	00:00:10.70	00:01:52.05
2048	Verschlüsseln	00:00:01.01	00:00:01.00
	k-means	00:00:19.28	00:03:47.01
	Entschlüsseln	00:00:00.18	00:00:00.15
	Gesamt	00:00:20.47	00:03:48.16
4096	Verschlüsseln	00:00:02.51	00:00:02.29
	k-means	00:01:07.87	00:13:26.45
	Entschlüsseln	00:00:00.46	00:00:00.46
	Gesamt	00:01:10.84	00:13:29.20
8192	Verschlüsseln	00:00:05.93	00:00:05.90
	k-means	00:04:27.28	00:53:10.87
	Entschlüsseln	00:00:01.66	00:00:01.70
	Gesamt	00:04:34.87	00:53:18.47

Tabelle 7.14: Iris Daten (verschlüsselt): Laufzeiten mit $T = 60$. Iter.: Iteration, k: Anzahl Cluster, T: Basis der Relinearisierung, n: Parameter des Polynommodulus $(x^n + 1)$.

Eine Gegenüberstellung der Gesamtlaufzeiten findet sich in Abbildung 7.2. Es lässt sich gut erkennen, dass für kleinere T die Laufzeit schneller ansteigt und höher ist. Erwartungsgemäß hat die Anzahl an Iterationen keinen Einfluss auf die Form der Kurven. Für mehr Iterationen steigt die Laufzeit im gleichen Maße an. Sie erhöht sich in etwa um den Faktor 12, was der Anzahl an Iterationen entspricht. Betrachten wir den maximalen



1 Iteration



12 Iterationen

Abbildung 7.2: Iris Daten (verschlüsselt): Gegenüberstellung der Gesamtlaufzeiten. rot: $T = 4$, grün: $T = \lceil\sqrt{q}\rceil$, dunkelblau (liegt unter grün): $T = 15$, lila: $T = 60$, Laufzeit: Minuten.

Laufzeitunterschied zwischen den schnellsten Testläufen mit dem Parameter $T = 60$ im Vergleich zu den langsamsten Testläufen mit dem Parameter $T = 4$ steigt der Anteil, mit dem der Parameter die Laufzeit beeinflusst, mit steigendem n. Eine Gegenüberstellung und die Ergebnisse dieses Vergleichs befinden sich in Tabelle 7.15. Verglichen werden darin die Laufzeiten für eine Iteration des Basisalgorithmus.

	$n = 1024$	$n = 2048$	$n = 4096$	$n = 8192$
$T = 60$	00:00:09.50	00:00:19.28	00:01:08.87	00:04:27.28
$T = 4$	00:00:10.19	00:00:22.07	00:01:32.33	00:07:51.78
Differenz:	00:00:00.69	00:00:02.79	00:00:23.46	00:03:24.50
	(6.77%)	(12.64%)	(25.41%)	(43.35%)

Tabelle 7.15: Iris Daten: Laufzeitunterschied zwischen $T = 4$ und $T = 60$ bei einer Iteration. T: Basis der Relinearisierung, n: Parameter des Polynommodulus $(x^n + 1)$, Differenz: ist in absoluten und realen Werten angegeben.

Die Laufzeit einer Iteration auf den Iris Daten kann sich durch den Parameter um bis zu 43.35 Prozent erhöhen. Wir weisen darauf hin, dass eine Wahl von sehr hohen T schnell für nicht ausreichendes Rausch Budget sorgen kann. Da die Laufzeiten für $T = \lceil\sqrt{q}\rceil$ und ein festes $T = 15$ relativ identisch sind, vergleichen wir nur letzteren Fall. In Tabelle 7.16 findet sich eine Gegenüberstellung für $T = 60$ und $T = 15$. Bei $n = 1024$ bis $n = 4096$ bleibt

der Laufzeitzuwachs auf geringem Niveau im Vergleich zu dem maximalen Laufzeitanstieg aus Tabelle 7.15. Bei $n = 8192$ steigt die Laufzeit durch den Parameter wieder stärker an, bleibt aber auf niedrigerem Niveau.

	$n = 1024$	$n = 2048$	$n = 4096$	$n = 8192$
$T = 60$	00:00:09.50	00:00:19.28	00:01:08.87	00:04:27.28
$T = 15$	00:00:09.97	00:00:20.80	00:01:16.08	00:05:24.53
Differenz:	00:00:00.47 (4.71%)	00:00:01.25 (6.01%)	00:00:07.21 (9.48%)	00:00:57.25 (17,64%)

Tabelle 7.16: Iris Daten: Laufzeitunterschied zwischen $T = 15$ und $T = 60$ bei einer Iteration. T: Basis der Relinearisierung, n: Parameter des Polynommodulus $(x^n + 1)$, Differenz: ist in absoluten und realen Werten angegeben.

In Abbildung 7.3 wird durch unterschiedliche Farben der Kurven die durchschnittliche Laufzeit der Verschlüsselung (blau) und der Entschlüsselung (grün) über alle Testreihen visualisiert. Da während der Entschlüsselung die berechneten Zentroide entschlüsselt und während der Verschlüsselung alle Datenpunkte verschlüsselt werden, sind die beiden Kurven nicht vergleichbar. Das Diagramm dient der Veranschaulichung, welchen Anteil die Ver- und Entschlüsselung im Vergleich zueinander an der Gesamtlaufzeit einnehmen, welcher durchschnittlich mit maximal 6.06 Sekunden sehr gering ausfällt.

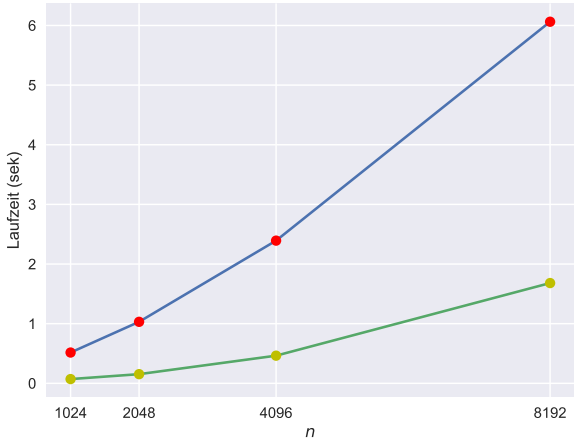


Abbildung 7.3: Iris Daten (verschlüsselt): Laufzeiten aller Verschlüsselungen (blau) und Entschlüsselungen (rot). n: Parameter des Polynommodulus $x^n + 1$, Laufzeit: Sekunden.

Die unterschiedlichen Gesamtlaufzeiten werden in Abbildung 7.4 in das Verhältnis gesetzt. In dem Diagramm visualisieren die unteren Laufzeiten die Verläufe einer Iteration und die oberen die Laufzeiten für jeweils 12 Iterationen, jeweils für die unterschiedlichen T. Wie wir in Abbildung 7.2 gesehen haben, verlaufen die beiden Kurven gleich, nur mit

anderen Skalierungen. Während der Unterschied bis $n = 4096$ noch mit ca. 15 Minuten vergleichsweise gering ist, steigt er bei $n = 8192$ auf ein deutlich höheres Niveau mit bis zu 1.5 Stunden an.

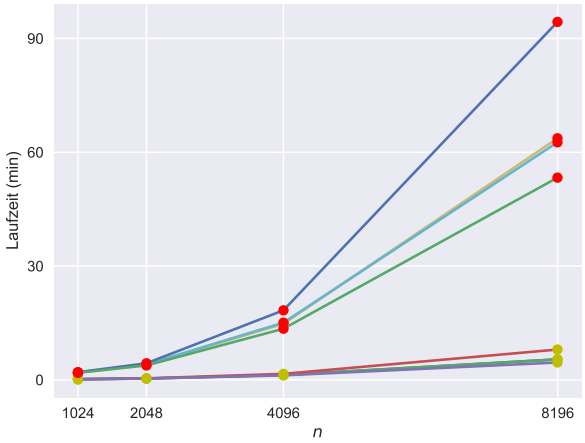


Abbildung 7.4: Iris Daten (verschlüsselt): Gesamtlaufzeiten Übersicht mit unterschiedlichen Parametern T. n: Parameter des Polynommodulus $x^n + 1$, T: Basis der Relinearisierung, unten: Laufzeiten mit einer Iteration, oben: Laufzeiten mit 12 Iterationen, Laufzeit: Minuten.

7.2.3 Dublin Datensatz

Ähnlich wie bei den Iris Daten, haben wir mit den Dublin Daten Laufzeittests durchgeführt. Der Parameter T der Relinearisierung wird bei diesem Test mit festem $T = 15$ gewählt, wie wir bereits in Kapitel 7.1.4 erwähnt haben. Wir haben für jeweils $k = 3$ und $k = 15$ Cluster die Clusteranalyse auf den verschlüsselten Daten für eine Iteration mit jeweils $d \in \{48, 96, 144, 240, 288, 336\}$ Dimensionen laufen lassen. Wie wir in den Laufzeittests der Iris Daten (Kapitel 7.2.2) gesehen haben, ändert sich der Verlauf der Laufzeit nicht bei steigenden Iterationen. Die Laufzeit ändert sich um einen konstanten Faktor, der von der Anzahl der Iterationen abhängig ist. Da unterschiedliche Iterationen, basierend auf Tabelle 7.1, für die gesamte Clusteranalyse gewählt wurden, ist eine Vergleichbarkeit der Laufzeiten so nur für die einzelnen Iterationen gegeben. Wie sich in ersten Testläufen herausgestellt hat, reicht das Rausch Budget, wie bei den Iris Daten für einen Polynommodulus von $x^{1024} + 1$ und $x^{2048} + 1$ nicht aus. Daher werden wir die gesamte Clusteranalyse für $k = 3$ Cluster nur auf ausreichendem Rausch Budget, zur Veranschaulichung und späterem Vergleich der Ergebnisgenauigkeit berechnen. Eine typische Aufgabenstellung für diese Clusteranalyse auf den Dublin Daten könnte die Klassifikation der Datenpunkte in voll, mittel und leer sein, basierend auf den Messungen eines bzw. mehrerer Tage.

Tabellen 7.17 bzw. 7.18 zeigen die Laufzeiten für eine Iteration mit einem Polynommodulus von $x^{1024}+1$ bzw. $x^{2048}+1$. In Tabelle 7.19 finden sich die Laufzeiten für eine Iteration und die gesamte Clusteranalyse für einen Polynommodulus von $x^{4096}+1$.

d	Messung	Laufzeit für: • 1 Iter. • $k=3$	Laufzeit für: • 1 Iter. • $k=15$
48	Verschlüsseln	00:00:12.13	00:00:12.02
	k-means	00:02:56.96	00:14:44.13
	Entschlüsseln	00:00:00.17	00:00:00.82
	Gesamt	00:03:09.26	00:14:56.97
96	Verschlüsseln	00:00:25.00	00:00:24.28
	k-means	00:05:48.48	00:29:12.85
	Entschlüsseln	00:00:00.18	00:00:00.92
	Gesamt	00:06:13.66	00:29:38.05
144	Verschlüsseln	00:00:36.12	00:00:35.83
	k-means	00:08:41.79	00:43:20.44
	Entschlüsseln	00:00:00.21	00:00:01.04
	Gesamt	00:09:18.12	00:43:57.31
192	Verschlüsseln	00:00:48.35	00:00:47.81
	k-means	00:11:38.37	00:57:35.25
	Entschlüsseln	00:00:00.23	00:00:01.14
	Gesamt	00:12:26.95	00:58:24.20
240	Verschlüsseln	00:01:00.55	00:00:59.98
	k-means	00:14:36.09	01:11:51.08
	Entschlüsseln	00:00:00.26	00:00:01.25
	Gesamt	00:15:36.90	01:12:52.31
288	Verschlüsseln	00:01:11.64	00:01:12.42
	k-means	00:17:17.44	01:26:39.78
	Entschlüsseln	00:00:00.28	00:00:01.51
	Gesamt	00:18:29.36	01:27:53.71
336	Verschlüsseln	00:01:24.69	00:01:24.36
	k-means	00:21:01.63	01:43:34.96
	Entschlüsseln	00:00:00.50	00:00:02.21
	Gesamt	00:22:26.82	01:45:01.53

Tabelle 7.17: Dublin Daten (verschlüsselt): Laufzeiten für $n=1024$ mit $T=15$. Iter.: Iteration, k: Anzahl Cluster, T: Basis der Relinearisierung, n: Parameter des Polynommodulus (x^n+1), d: Dimension des Datensatzes.

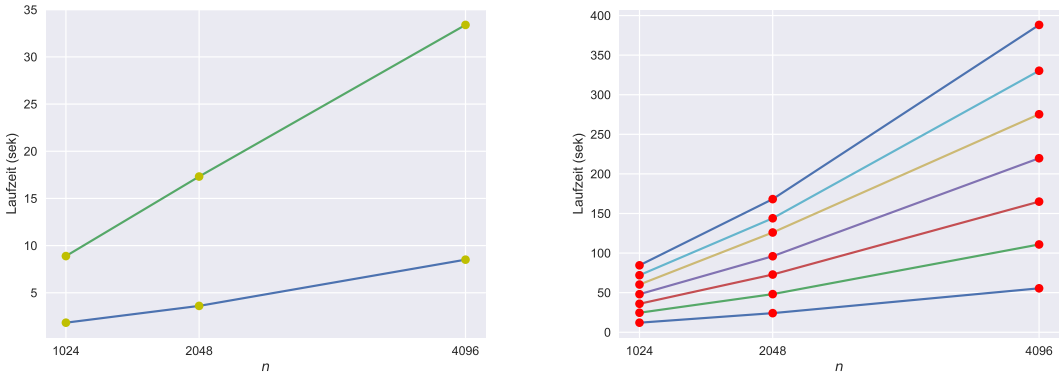
d	Messung	Laufzeit für: • 1 Iter. • $k = 3$	Laufzeit für: • 1 Iter. • $k = 15$
48	Verschlüsseln	00:00:24.16	00:00:24.09
	k-means	00:06:10.48	00:30:53.15
	Entschlüsseln	00:00:00.34	00:00:01.71
	Gesamt	00:06:34.98	00:31:18.95
96	Verschlüsseln	00:00:47.99	00:00:48.31
	k-means	00:12:09.76	01:00:52.78
	Entschlüsseln	00:00:00.37	00:00:01.84
	Gesamt	00:12:58.12	01:01:42.93
144	Verschlüsseln	00:01:13.86	00:01:11.92
	k-means	00:18:12.49	01:30:39.67
	Entschlüsseln	00:00:00.41	00:00:02.07
	Gesamt	00:19:26.76	01:31:53.66
192	Verschlüsseln	00:01:35.90	00:01:36.13
	k-means	00:24:58.50	02:04:16.05
	Entschlüsseln	00:00:00.66	00:00:03.28
	Gesamt	00:26:35.06	02:05:55.46
240	Verschlüsseln	00:02:09.70	00:02:02.28
	k-means	00:31:12.33	02:31:32.56
	Entschlüsseln	00:00:00.66	00:00:02.55
	Gesamt	00:33:22.69	02:33:37.39
288	Verschlüsseln	00:02:24.20	00:02:23.70
	k-means	00:36:16.61	03:01:05.19
	Entschlüsseln	00:00:00.57	00:00:02.80
	Gesamt	00:38:41.38	03:03:31.69
336	Verschlüsseln	00:02:47.94	00:02:48.38
	k-means	00:42:33.00	03:32:14.09
	Entschlüsseln	00:00:00.60	00:00:03.07
	Gesamt	00:45:21.54	03:35:05.54

Tabelle 7.18: Dublin Daten (verschlüsselt): Laufzeiten für $n = 2048$ mit $T = 15$. Iter.: Iteration, k: Anzahl Cluster, T: Basis der Relinearisierung, n: Parameter des Polynommodulus $(x^n + 1)$, d: Dimension des Datensatzes.

d	Messung	Laufzeit für: • 1 Iter. • $k = 3$	Laufzeit für: • 1 Iter. • $k = 15$	Laufzeit für: • Iter. nach Tabelle 7.1 • $k = 3$
48	Verschlüsseln	00:00:55.43	00:00:55.55	00:00:55.69
	k-means	00:23:40.59	01:58:03.77	03:57:10.80
	Entschlüsseln	00:00:01.03	00:00:05.17	00:00:01.07
	Gesamt	00:24:37.05	01:59:04.49	03:58:07.56
96	Verschlüsseln	00:01:50.40	00:01:51.34	00:01:50.61
	k-means	00:46:52.75	03:53:45.29	10:09:17.74
	Entschlüsseln	00:00:01.18	00:00:05.90	00:00:01.19
	Gesamt	00:48:44.33	03:55:42.53	10:11:09.54
144	Verschlüsseln	00:02:45.36	00:02:44.57	00:02:46.50
	k-means	01:09:45.47	05:49:20.22	07:02:40.72
	Entschlüsseln	00:00:01.33	00:00:06.60	00:00:01.32
	Gesamt	01:12:32.16	05:52:11.39	07:05:28.54
192	Verschlüsseln	00:03:40.17	00:03:39.37	00:03:41.80
	k-means	01:33:09.87	07:45:36.75	12:29:43.59
	Entschlüsseln	00:00:01.47	00:00:07.48	00:00:01.51
	Gesamt	01:36:51.51	07:49:23.60	12:33:29.90
240	Verschlüsseln	00:04:35.31	00:04:35.30	00:04:36.18
	k-means	01:56:01.06	09:45:07.65	11:40:57.14
	Entschlüsseln	00:00:01.61	00:00:08.24	00:00:01.64
	Gesamt	02:00:37.98	09:49:51.19	11:45:34.96
288	Verschlüsseln	00:05:30.28	-	00:05:36.44
	k-means	02:19:04.44	-	14:03:39.43
	Entschlüsseln	00:00:01.89	-	00:00:01.76
	Gesamt	02:24:36.61	-	14:09:17.63
336	Verschlüsseln	00:06:28.16	-	-
	k-means	-	-	-
	Entschlüsseln	-	-	-
	Gesamt	-	-	-

Tabelle 7.19: Dublin Daten (verschlüsselt): Laufzeiten für $n = 4096$ mit $T = 15$. -: abgebrochene Durchläufe, Iter.: Iteration, k: Anzahl Cluster, T: Basis der Relinearisierung, n: Parameter des Polynommodulus $(x^n + 1)$, d: Dimension des Datensatzes.

Abbildung 7.5 stellt die Laufzeiten der Verschlüsselungen (rechts) mit durchschnittlichen Laufzeiten der Entschlüsselungsvorgänge (links) gegenüber. Wie zu erwarten, ist die Laufzeit analog zu den Iris Daten für die Verschlüsselungen länger als für die Entschlüsselungen. Bei der Verschlüsselungen werden die sieben verschiedenen Dimensionen des Datensatzes durch die sieben Kurven repräsentiert. Die Laufzeit korelliert mit der Anzahl der Merkmale und steigt linear für wachsendes n. Die Anzahl an Clustern beeinflusst die Laufzeit der Entschlüsselung. Die untere Kurve repräsentiert die durchschnittlichen Laufzeiten der Entschlüsselung für $k = 3$ und die obere für $k = 15$.



Entschlüsselung Verschlüsselungen

Abbildung 7.5: Dublin Daten (verschlüsselt): Laufzeiten aller Verschlüsselungen (gelbe Punkte) und Entschlüsselungen (rote Punkte). n: Parameter des Polynommodulus $x^n + 1$, Laufzeit: Sekunden.

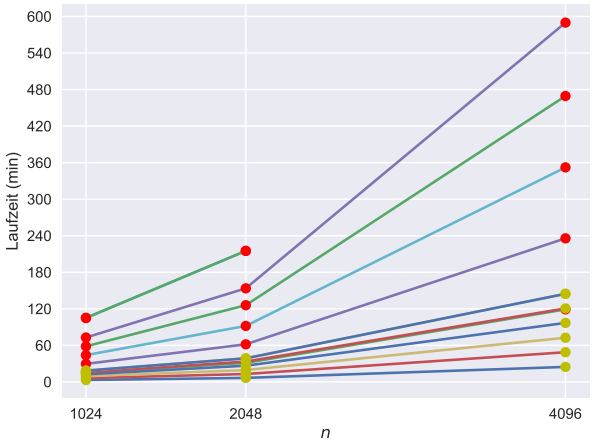


Abbildung 7.6: Dublin Daten (verschlüsselt): Gesamtlaufzeiten einer Iteration. n: Parameter des Polynommodulus $x^n + 1$, Laufzeit: Minuten.

Für jeweils eine Iteration visualisiert Abbildung 7.6 die unterschiedlichen Gesamtlaufzeiten. Zu erkennen ist ein unterschiedlicher Verlauf der Kurven mit roten Punkten zu den Kurven mit gelben Punkten. Die Kurven mit den roten Punkten visualisieren die Laufzeiten für $k = 15$ und die Kurven mit gelben Punkten die Laufzeiten für $k = 3$.

7.2.4 Laufzeitvergleich

Zugunsten eines einfachen Überblicks haben wir in Abbildung 7.7 die bereits präsentierten Abbildungen 7.4 und 7.6 gegenübergestellt. Die Skalierungen der beiden Diagramme unserer Laufzeittests sind unterschiedlich. Das ist zum einen darin begründet, dass bei den Dublin Daten (rechts) die Verschlüsselung bei $n = 8192$ zum Abbruch führte. Zum anderen dauern die Laufzeiten der Dublin Daten erwartungsgemäß länger.

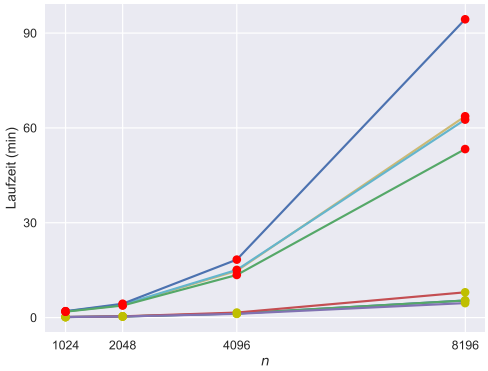


Abbildung 7.4

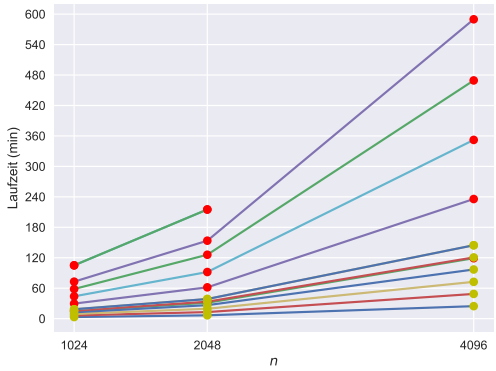


Abbildung 7.6

Abbildung 7.7: Gegenüberstellung der Laufzeiten der Iris Daten (links) und der Dublin Daten (rechts). n: Parameter des Polynommodulus $x^n + 1$, Laufzeit: Minuten

7.3 Ergebnisgenauigkeit der Clusternalysen

Zur Beantwortung der dritten Forschungsfrage "Wie lässt sich die Ergebnisgenauigkeit einer k-means Clusternalyse auf FV verschlüsselten Daten beschreiben?" haben wir bei den erfolgten Experimenten während der Berechnung vom Klienten das Rausch Budget bei jeder Rechenoperation überprüfen lassen. Des Weiteren wurden von uns die Ergebnisse der Testdurchläufe dokumentiert. Leichte Abweichungen des Rausch Budgets untereinander lassen sich damit begründen, dass das anfängliche Rauschen zufällig gewählt wird.

7.3.1 Iris Datensatz

Das minimale Rausch Budget während der Testläufe auf den Iris Daten listen wir in Tabelle 7.20 auf. Wir wählen hierzu die Schreibweise "a / b", wobei a das minimale Rausch Budget

bei einer Iteration und b bei 12 Iterationen darstellt. Es lässt sich erkennen, dass für einen Polynommodulus von $x^{1024}+1$ und $x^{2048}+1$ das Rausch Budget in allen Testreihen nicht ausreichte und dementsprechend das Rauschen bei der Entschlüsselung zu groß geworden ist, um vollständig herausgekürzt werden. Die beiden anderen gewählten Polynommoduli waren dagegen ausreichend für alle Testreihen mit Ausnahme von dem maximalen $T = 60$ bei einem Polynommodulus von $x^{4096}+1$.

n	$T = 15$	$T = \lceil\sqrt{q}\rceil$	$T = 4$	$T = 60$
1024	0 / 0			
2048	0 / 0			
4096	4 / 7	2 / 5	3 / 5	0 / 0
8192	107 / 109	104 / 106	105 / 109	71 / 71

Tabelle 7.20: Iris Daten (verschlüsselt): minimales Rausch Budget. Angaben für: 1 Iteration / 12 Iterationen, q: Koeffizientenmodulus, T: Basis der Relinearisierung, n: Parameter des Polynommodulus (x^n+1).

Es ist in Tabelle 7.20 gut zu erkennen, dass die Parameter $T = 15$, $T = \lceil\sqrt{q}\rceil$ und $T = 4$ in unserem Fall das Rausch Budget vergleichbar beeinflussen. Zudem hat sich in unseren Experimenten in Kaptiel 7.2.2 herausgestellt, dass größere T zu einer besseren Laufzeit führen. Bezogen auf unsere getesteten Parameter T ist $T = 15$ die beste Wahl.

Tabelle 7.21 zeigt den Vergleich der Ergebnisse. Verglichen werden die berechneten Cluster unseres Basisalgorithmus der Testreihen mit jeweils 12 Durchläufen auf verschlüsselten und unverschlüsselten Daten. Wir wählen zur Präsentation die Schreibweise "richtig erkannt / falsch erkannt". Es ist gut zu erkennen, dass alle Ergebnisse mit ausreichendem Rausch Budget zu einer Übereinstimmung mit dem Ergebnis des Basisalgorithmus führen. Reicht das Rausch Budget nicht aus, werden unvorhersehbare und nicht wiederholbare Ergebnisse produziert. Das ist darin begründet, dass das Rauschen zufällig gewählt wird und nicht erkennbar ist wie weit das Rausch Budget durch das Rauschen überschritten wird. Dementsprechend werden, je nachdem wie weit das Rausch Budget überschritten wird, dennoch manche Datenpunkte korrekt zugeordnet.

n	$T = 15$	$T = \lceil\sqrt{q}\rceil$	$T = 4$	$T = 60$
1024	051 / 099	049 / 101	053 / 097	053 / 097
2048	047 / 103	032 / 118	050 / 100	046 / 104
4096	150 / 0	150 / 0	150 / 0	063 / 087
8192	150 / 0	150 / 0	150 / 0	150 / 0

Tabelle 7.21: Iris Daten (verschlüsselt): Ergebnisgenauigkeit. Schreibweise: richtig / falsch erkannt, q: Koeffizientenmodulus, n: Parameter des Polynommodulus (x^n+1), T: Basis der Relinearisierung.

7.3.2 Dublin Datensatz

Wie bei den Iris Daten reicht bei den Dublin Daten das Rausch Budget bei einer Wahl des Polynommodulus von $x^{1024}+1$ und $x^{2048}+1$ nicht aus. Diese werden wir nicht in einer Tabelle präsentieren. In Tabelle 7.22 listen wir das minimale Rausch Budget für die Testreihen für einen Polynommodulus von $x^{4096}+1$ auf.

d	k	Rausch Budget
48	3	7
	15	6
96	3	11
	15	8
144	3	4
	15	3
192	3	3
	15	4
240	3	6
	15	7
288	3	6
	15	-

Tabelle 7.22: [Dublin Daten (verschlüsselt): minimales Rausch Budget für $n = 4192$. k: Anzahl der Cluster, n: Parameter des Polynommodulus $(x^n + 1)$, d: Dimension des Datensatzes, -: nicht mögliche Durchläufe.

Das nicht ausreichende Rausch Budget der Polynommoduli $x^{1024}+1$ und $x^{2048}+1$ waren der Grund, weshalb wir uns entschieden haben, die Gesamtdurchläufe ausschließlich auf $x^{4096}+1$ durchzuführen. Ebenso wie bei den Iris Daten waren bei den Dublin Daten alle Clusteranalysen auf den verschlüsselten Daten korrekt.

Abbildungen 7.8 und 7.9 zeigen das Ergebnis der Clusteranalyse auf den verschlüsselten und unverschlüsselten Dublin Daten. Es lässt sich gut erkennen, dass die Einteilung in die verschiedenen Cluster erwartungsgemäß für die verschiedene Anzahl an Ausprägungen unterschiedlich ausfällt. Für die gleiche Anzahl Dimensionen sind die Ergebnisse bei mehreren Durchläufen die gleichen, wenn die Initialzentroiden gleich gewählt wurden. Anhand der unterschiedlichen Punktfärbungen sind drei verschiedene Cluster zu erkennen. Die Cluster, die mit roten Punkten symbolisiert werden, zeigen beispielsweise die Zubringerstraßen zu den Autobahnen bei Dublin. Interessierte finden die restlichen Bilder zu den Clusterergebnissen im Anhang.

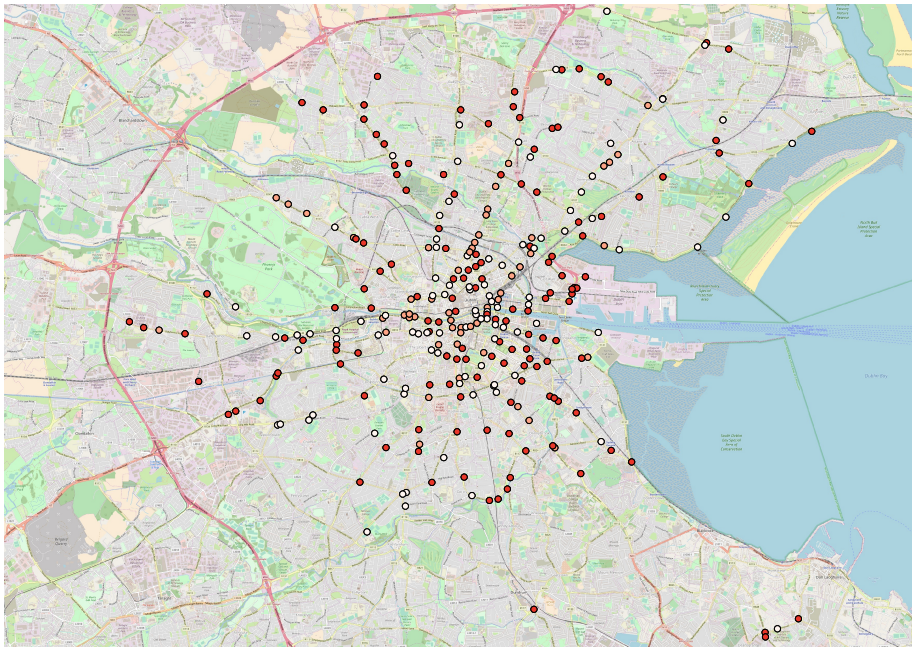


Abbildung 7.8: Ergebnis Clusteranalyse auf Dublin Daten mit 48 Messungen. $k = 3$ Cluster, $d = 48$ Dimensionen.

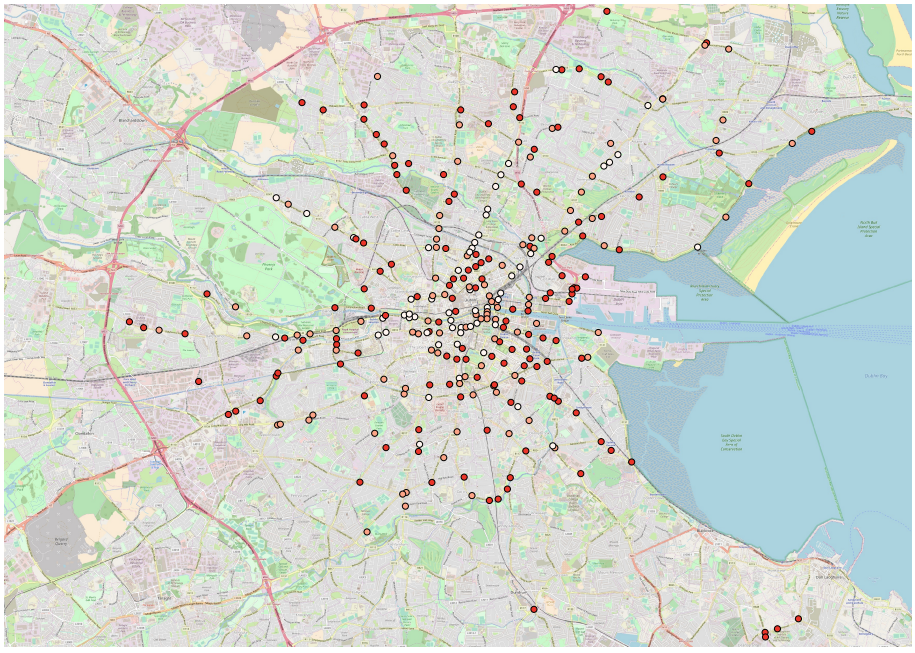


Abbildung 7.9: Ergebnis Clusteranalyse auf Dublin Daten mit 288 Messungen. $k = 3$ Cluster, $d = 288$ Dimensionen.

Kapitel 8

Diskussion

Das Hauptziel der vorliegenden Bachelorarbeit war es, die Einsatzfähigkeit des Fan Vercauteren (FV) Verschlüsselungssystems [67] in der Version v2.3.0-4 der SEAL Bibliothek [33] am Beispiel einer k-means Clusteranalyse zu dokumentieren. Gleichzeitig wollten wir potentiellen Anwendern den Einstieg in diese moderne Algorithmenklasse erleichtern und die von vorheriger Forschung geforderten Praxisdaten bereitstellen. Dabei legten wir besonderen Wert auf eine hohe Nachvollziehbarkeit. Unsere Experimente führten wir zum einen mit einem öffentlich zugänglichen Datensatz, den Iris Daten[1], durch. Damit sind unsere Ergebnisse zum Teil barrierefrei replizierbar. Zum anderen wurden k-means Clusteranalysen auf zur Verfügung gestellten spatio-temoralen Forschungsdaten durchgeführt. Diese wurden als Dublin Daten bezeichnet. Von diesen vorverarbeiteten, horizontalen Daten haben wir eine Teilmenge von 48 bis 336 Messungen untersucht. Wir formulierten fünf Forschungsfragen, zu deren Beantwortung nun Ergebnisse des theoretischen und des praktischen Teils der vorliegenden Untersuchung herangezogen werden.

Bei der Gestaltung des theoretischen Teils der Untersuchung gingen wir von der Beobachtung aus, dass in Zeiten von Big Data fachübergreifend zwar ein großes Interesse an kryptografischen Algorithmen besteht, allerdings Lehrbücher die modernen homomorphen Algorithmen (LHE) meist noch stiefmütterlich behandeln. Darum erläuterten wir die Grundlagen der homomorphen Kryptologie besonders ausführlich. Wir klärten das typische Begriffswirrwar in der Literatur dieses jungen Forschungsfeldes, indem wir die Algorithmen nach zwei in der Forschungscommunity häufig zitierten Arbeiten ordneten. Wir arbeiteten heraus, dass die eher grobe Klassifikation von Albrecht [7] für die Praxis ausreicht, während die feinere und eher technische Klassifikation von Armknecht [8] Kryptografen bei der Weiterentwicklung der Algorithmen mehr entgegenkommt. Zudem gaben wir einen umfassenden Überblick über den Forschungsstand aller homomorphen Algorithmenklassen. Wir versprechen uns für Praktiker und Forscher einen gleichermaßen hohen Nutzen durch

[1] `https://archive.ics.uci.edu/ml/datasets/iris`

die von uns zusätzlich bereitgestellten Überblicke in tabellarischer Form. Diese sollten nicht nur geeignet sein, die Nachvollziehbarkeit unserer Experimente zu steigern. Potentiellen Anwendern wollten wir ein dauerhaft nützliches Tool an die Hand geben. In diesen Überblickstabellen finden sich nämlich Praxishinweise, Literaturangaben sowie Angaben zu historischen Laufzeiten und der damals verwendeten Technik. Diese Überblickstabellen können somit die Auswahl eines passenden Algorithmus für ein spezifisches Szenarium erleichtern und bei der eigenständigen Einarbeitung in die Literatur helfen. Interessierte können die Tabellen zukünftig leicht ergänzen.

Der Praxisteil dieser Untersuchung ist die Grundlage zur Beantwortung unserer ersten drei technischen Forschungsfragen. Die wichtigsten Ergebnisse unserer Experimente werden in den folgenden drei Absätzen noch einmal herausgestellt und mit Bezug zu vorheriger Forschung diskutiert. In einem Unterkapitel zeigen wir die Limitationen unserer Untersuchung auf. Auf dieser Basis können wir schließlich alle Forschungsfragen beantworten und Hinweise für zukünftige Anwender und Forscher bereitstellen.

Zur Beantwortung der Frage "Wie aufwendig ist die Parameterbestimmung?" müssen wir zunächst betonen, dass diese für gute Ergebnisse maßgeschneidert auf die jeweiligen Daten und Experimente angepasst werden mussten. Vor der eigentlichen Parameterwahl muss sich der potentielle Anwender mit den Rechenoperationen des geplanten Verfahrens auseinandersetzen, um die multiplikative Tiefe zu bestimmen. Darauf basierend kann das passende Verschlüsselungsschema ausgewählt werden. Für den von uns genutzten k-means Algorithmus ermittelten wir eine multiplikative Tiefe von 4. Aus diesem Grund entschieden wir uns für das FV Verschlüsselungsschema, einem LHE Algorithmus der zweiten Generation. FV gilt als das sicherste Verschlüsselungsschema [159]. Wir konnten uns nicht bedingungslos auf vorherige Forschung [63] stützen, sondern mussten den Basisalgorithmus anpassen, damit dieser korrekte Ergebnisse liefert.

Zuerst wurde die Sicherheit λ, die wir garantieren wollten, gewählt. Darauf basierend konnten wir die ersten Parameter der FV Verschlüsselung wählen, nämlich den Polynommodulus $\Phi_m(x) = x^n + 1$, den darauf basierenden Koeffizientenmodulus q und den Klartextmodulus t, die sich gegenseitig im Hinblick auf die Sicherheit, die Ergebnisgenauigkeit bzw. das Rauschen und die Laufzeit beeinflussen. Erleichtert wurde dies durch die Funktion aus SEAL v2.3.0-4, die erlaubt, automatisch den Koeffizientenmodulus, basierend auf dem n des Polynommodulus und der gewünschten Sicherheit zu wählen. Es waren verschiedene Testläufe nötig, um herauszufinden, dass für unsere Experimente der Klartextmodulus mit dem Polynommodulus übereinstimmen sollte.

Die Wahl der Basis der Relinearisierung T betrachteten wir anhand der Iris Daten. Es stellte sich heraus, dass die Empfehlung für $T = \lceil \sqrt{q} \rceil$ aus [67] und unsere darauf basierende Wahl $T = 15$ ein guter Kompromiss zwischen Ergebnisgenauigkeit und Lauf-

zeit war. Schlussendlich musste der Encoder gewählt werden, in unserem Fall war dies der Fractional Encoder, der weitere Parameterwahlen erforderlich machte. Dieser benötigte eine Begrenzung des ganzzahligen Anteils n_i der zu verschlüsselnden Zahlen, der jeweiligen Nachkommastellen n_f und die Basis B, auf die die Zahlen in die jeweiligen Polynome aufgeteilt werden. Es stellte sich durch Testläufe heraus, dass $B = 3$ zu geringerer Laufzeit führt. Für die Begrenzung des ganzzahligen Anteils war es nötig, die maximale Zahlengröße innerhalb aller Berechnungen des Algorithmus abzuschätzen. Die Begrenzung der Nachkommastellen wurden durch verschiedene Testläufe ermittelt. Sie kann einen Einfluss auf die Ergebnisgenauigkeit haben, da durch die Nachkommastellen der ganzzahlige Anteil im Polynom überschrieben werden kann. Mit unserer Parametrisierung war dies nicht der Fall. Des Weiteren zeigte sich in unseren Experimenten, dass die Begrenzungen keinen Einfluss auf die Laufzeit haben.

Die verwendete Hardware ist ein weiterer Faktor, der bei der Parameterwahl betrachtet werden muss. In unserer Untersuchung war der verfügbare Arbeitsspeicher von 11.2 GB ein limitierender Faktor. Bei dem Dublin Datensatz wurden die Testdurchläufe mit einem Polynommodulus von $x^{8192} + 1$ während der Verschlüsselung aus diesem Grund abgebrochen. Zudem war ein gelungener Verschlüsselungsvorgang noch kein Garant dafür, dass die Clusteranalyse mit diesen Parametern durchgeführt werden konnte. So brachen die Testdurchläufe des Basisalgorithmus für eine Iteration auf den Dublin Daten bei den eigentlichen Experimenten mit einem Polynommodulus von $x^{4096}+1$ für $d = 336$ bei $k = 3$ Clustern ab, ebenso wie für $d = 288$ und $d = 336$ bei $k = 15$ Clustern.

Zusammenfassend muss die Parameterwahl als ausgesprochen aufwändig, anspruchsvoll und essenziell für eine Verschlüsselung bezeichnet werden, die zu korrekten Ergebnissen führt. Ohne zeitaufwändige Testläufe müssen sehr grobe Parameterwahlen erfolgen. Die Folge sind zu lange Laufzeiten und unter Umständen verfälschte Ergebnisse durch zu hohes Rauschen. Es ist uns aufgefallen, dass in vorhergehender Forschung [63] eine nicht so aufwendige Parameterbestimmung durchgeführt worden ist. Dies müssen wir für den späteren Vergleich mit vorheriger Forschung im Hinterkopf behalten.

Zur Beantwortung unserer Forschungsfrage nach der Höhe der Laufzeitkosten bei einer k-means Clusteranalyse auf FV verschlüsselten Daten haben wir die Laufzeiten unserer Testläufe gestoppt und mit den Testläufen der unverschlüsselten Daten verglichen. Erwartungsgemäß waren die benötigten Zeiten sowohl bei dem Iris Datensatz wie auch bei den zur Verfügung gestellten Sensordaten aus Dublin deutlich höher als bei den Versuchen auf den Klartextdaten. Es zeigte sich, dass bei der Erhöhung des Polynommodulus und der damit einhergehenden Erhöhung des Koeffizienten und Klartextmodulus die Laufzeit überproportional ansteigt. Anhand der Experimente auf den Iris Daten zeigte sich, dass der Laufzeitzuwachs bis zu einem Polynommodulus von $x^{4096} + 1$ moderat ansteigt. Für $x^{8192} + 1$ war die Laufzeit deutlich höher. Die Anzahl an Iterationen hatte erwartungs-

gemäß keinen Einfluss auf den Verlauf des Laufzeitanstiegs, sondern sorgte nur für eine Erhöhung der Laufzeit bei gleichbleibendem Verlauf der Kurven. Des Weiteren zeigte sich, dass der Parameter T die Laufzeit um so mehr negativ beeinflusst, je größer er gewählt wird. Dieses Phänomen verstärkt sich, je größer der Polynommodulus ist. Für das kleinst mögliche T, in unserem Fall mit einem Wert von 4, stieg die Laufzeit im Vergleich zu dem größten $T = 60$ beispielsweise um bis zu 43.35 Prozent an. Für den Fall $T = 15$ zeigte sich bis zu einem Polynommodulus von $x^{4096} + 1$ ein moderater und deutlich geringerer Anstieg der Laufzeit im Vergleich zu $T = 4$. Er stieg um einen Anteil von bis zu 9.48 Prozent im Vergleich zu dem größten $T = 60$ an. Im Vergleich dazu stieg bei $T = 4$ die Laufzeit um bis zu 25.41 Prozent an. Mit einem Polynommodulus von $x^{8192} + 1$ ergab sich mit $T = 15$ ein Laufzeitanstieg um $17,64$ Prozent und mit $T = 4$ ein Laufzeitanstieg um 43.35 Prozent. Die Laufzeiten für die Durchläufe mit ausreichendem Rausch Budget lagen zwischen ca. 15 Minuten und 1.5 Stunden.

Die Experimente auf den Dublin Daten unterschieden sich hinsichtlich der Dimensionen und der Anzahl an Clustern für eine Iteration. Erwartungsgemäß waren die Laufzeiten, wie bei den Iris Daten, für die Verschlüsselung länger als für die Entschlüsselung. Bei den Verschlüsselungen der sieben verschiedenen Dimensionen zeigten die Kurven einen identischen Verlauf. Die Anzahl der Cluster beeinflusste die Laufzeit, aber sorgte nicht für eine Veränderung des Verlaufs der Kurven. Die Berechnungsvorgänge für $d = 336$ bei $k = 3$ Clustern sowie $d = 288$ und $d = 336$ bei $k = 15$ Clustern brachen aufgrund des nicht ausreichenden Arbeitsspeichers ab.

Die geringste Laufzeit für die gesamte Clusteranalyse auf den Dublin Daten betrug gerundet 3 Stunden und 58 Minuten bei 48 Messungen je Datenpunkt. Die längste Laufzeit lag bei gerundet 14 Stunden und 9 Minuten mit 288 Messungen. Es liegt auf der Hand, dass solche Zeiten in der Praxis nicht akzeptabel sind. Dazu kommt, dass Testläufe von in etwa gleicher Dauer zur Bestimmung der Parameter im Vorfeld erforderlich sind. Die jeweiligen Ver- und Entschlüsselungen bei den Experimenten auf den Dublin Daten und Iris Daten nahmen einen sehr geringen Anteil an der Gesamtlaufzeit aller Experimente ein. Zusammenfassend zeigte sich in unseren Experimenten, dass Berechnungen nur auf kleinen Datensätzen mit wenigen Dimensionen heutzutage auf vergleichbaren Computersystemen praktikabel sind.

Die dritte Forschungsfrage zielte auf eine Beurteilung der Ergebnisgenauigkeit einer k-means Clusteranalyse auf FV verschlüsselten Daten im Vergleich zu Klartextdaten ab. Hier konnten wir keinen Unterschied feststellen, solange das Rausch Budget durch die Parametrisierung groß genug war. Wir beobachteten, dass bei nicht ausreichendem Rausch Budget keine Warnung ausgegeben wird, sondern nur das verfälschte Ergebnis. Zwar können unsinnige Werte wie z. B. negative Vorzeichen bei den Zentroiden zu Verwunderung führen, doch müssen solche Werte nicht zwangsläufig auftreten. Im schlimmsten Fall ist das

Ergebnis nicht als falsches Ergebnis erkennbar. Daher raten wir dazu, das Rausch Budget laufend zu überprüfen. Unsere Experimente zeigten, dass für unseren Basisalgorithmus ein Polynommodulus von mindestens $x^{4096} + 1$ nötig war. Clusteranalysen auf den Iris Daten zeigten, dass der Parameter T sorgfältig gewählt werden muss, da er den Verbrauch des Rausch Budgets soweit beeinflussen kann, dass dies nicht mehr ausreicht, wie im Falle $T = 60$ bei einem Polynommodulus von $x^{4096} + 1$. So wurde gezeigt, dass bei steigendem T zwar die Laufzeit sinkt, dafür aber auch das Rausch Budget.

An den Experimenten auf den Dublin Daten konnten wir sehen, dass für große Datenmengen der verfügbare Arbeitsspeicher die Mengen an möglichen Parametrisierungen für ausreichendes Rausch Budget weiter begrenzt. So konnte ein sinnvolles Ergebnis nur für einen Polynommodulus von $x^{4096} + 1$ berechnet werden. An dieser Stelle müssen wir auf die Parameterbestimmung zurückkommen. Denn auch die Wahl der Begrenzungen für den Encoder ist für die Dublin Daten weitläufiger geworden, sodass ein Überlauf der Nachkommastellen wahrscheinlicher wurde, dieser aber in unserem Fall nicht aufgetreten ist. Mit steigender Anzahl an Dimensionen d der verwendeten Datensätze oder kleiner werdender Anzahl Cluster k würden sich die darzustellenden Zahlen während der Berechnungen vergrößern. Dies könnte durch eine höhere Anzahl Cluster ausgeglichen werden, was wiederum aber die Laufzeit erhöhen würde. Wir kommen so zu dem Schluss, dass nur mit maßgeschneiderten Parametern für kleinere Datensätze korrekte Ergebnisse in verhältnismäßig akzeptabler Zeit berechnet werden können.

Verglichen mit der Parameterbestimmung bei den praxiserprobten (P)HE Algorithmen können wir in Übereinstimmung mit vorheriger Forschung festhalten, dass die erhöhte Flexibilität der LHE Algorithmen mit einem erheblich höheren Aufwand einhergeht. Anwender müssen zahlreichere und komplexere Entscheidungen treffen. In jüngster Vergangenheit sind dazu zwar hilfreiche Whitepapers erschienen, allerdings ersetzen sie nicht ein fundiertes kryptologisches Wissen hinsichtlich der unterschiedlichen Verfahren und des Forschungsstands.

Unsere vierte Forschungsfrage lautete: "Unterscheiden sich die Ergebnisse von vorheriger Forschung und falls ja, wie?". Betrachten wir die auf Grundlage einer Literaturrecherche zusammengestellte Laufzeittabelle (vgl. Tabelle 3.10) und bedenken wir die verwandte Technik, dann sind unsere Laufzeitergebnisse konform mit vorangegangener Forschung. Die optimistische Sicht der in der Einleitung zitierten Arbeiten von Aslett und Kollegen [9, 10], die postulierten, LHEs seien heute schon konkurrenzfähig einsetzbar, können wir dahingehend bestätigen, dass wir einschränkend ergänzen: Bei relativ kleinem $\Phi_m(x)$, leistungsfähiger Technik, nicht zu hoher Anzahl an Datenpunkten und Ausprägungen sowie kleinem k, im Falle von k-means Clusteranalysen. Ein konkretes Beispiel für private Daten, die durch LHEs heute geschützt werden können, sind ärztliche Diagnosen in schriftlicher Form. Bei Bilddateien, wie beispielsweise Röntgenbildern, sind wir aufgrund der großen

Datenmenge und unserer Ergebnisse skeptisch. Hier schließen wir uns eher der vorsichtigen Sicht von Wiese und Kollegen [159] an, die vor dem Einsatz der modernen Algorithmen bestimmte Anforderungen erfüllt sehen wollen. Unsere Beobachtung, dass in vorhergehender Literatur [63] die Parameterbestimmung nicht dokumentiert wurde und nicht alle Parameter angegeben wurden führt zu unserem Wunsch, dass zukünftige Forschung dabei anders verfährt. Nur durch Transparenz kann die Forschung zu homomorphen Verschlüsselungen nachvollzogen und voran getrieben werden. In der folgenden Betrachtung der Limitationen werden wir weitere Anforderungen mit betrachten.

8.1 Limitationen

Erste Limitationen ergeben sich natürlich durch den zeitlich beschränkten Rahmen einer Bachelorarbeit. Die langen Laufzeiten von zum Teil mehr als 12 Stunden in Abhängigkeit der Größe des $\Phi_m(x)$, q, t, T, k, der Datenmenge und der Anzahl an untersuchten Merkmalen, die zudem aufgrund technischer Probleme nicht immer im ersten Anlauf gelangen, verhinderten weitere Kontrolldurchläufe. Aufgrund seiner fachübergreifenden hohen Verbreitung haben wir die k-means Clusteranalyse als Beispielverfahren gewählt. In der Praxis wird dieses Verfahren auf Klartextdaten als intelligenter Lernalgorithmus meist mit Modifikationen angewandt, die zu kürzen Laufzeiten führen. Daher lag es auf der Hand, dass auch wir uns in der Hoffnung, den Aufwand für die Kommunikation und der homomorphen Berechnungen zu reduzieren, in Coresets eingearbeitet haben und uns mit einer streaming Approximation von k-means intensiv auseinandergesetzt haben. Wie bei Klartextdaten käme es bei der Verwendung von Coresets durch die reduzierte Anzahl an Datenpunkten und ggf. einer Reduktion der Merkmale vermutlich zu geringeren Laufzeiten. Allerdings ist die Bestimmung eines Coresets auf verschlüsselten Daten aufgrund der vielen Divisionen und Abstandsvergleiche und damit einhergehenden hohen Kommunikationkosten sehr zeitaufwändig. Es muss aufgrund der zeitlichen Beschränkung dieser Untersuchung weiterer Forschung überlassen bleiben, ob sich der zu erwartende Laufzeitvorteil tatsächlich lohnt.

Eine weitere Einschränkung der Ergebnisse ergibt sich durch die Simulation der Klient-Server Umgebung. Es wurde nur der Fall mit einem Klienten und keine Latenzen durch die Versandwege der Daten betrachtet. Wie Wiese und Kollegen in ihrer Anforderungsanalyse bemängeln [159], ist bislang noch kein LHE oder FHE in einer realen Cloud getestet worden. Denkbar ist, dass sich in der Praxis noch unentdeckte Herausforderungen zeigen, die sowohl die Laufzeiten als auch die Ergebnisgenauigkeit beeinflussen könnten.

Hinsichtlich der Gültigkeit der vorliegenden Arbeit wird der theoretische Teil, auch wenn er wie üblich nicht den Anspruch auf Vollständigkeit erheben darf, valide bleiben. Die Ergebnisse der Experimente dagegen werden aufgrund der sich rasant weiterentwickelnden Algorithmen und Technik nur kurzfristig replizierbar sein. Verbesserungen des FV Algorithmus sind bereits publiziert [34], jedoch noch nicht in SEAL implementiert. Da ein

Autor, Kim Laine, gleichzeitig Administrator von SEAL ist, wird es vermutlich nicht mehr lange dauern. Während wir diese Diskussion schreiben, stellen wir fest, dass bereits die Version SEAL v2.3.1 publiziert wurde.

Unsere Ergebnisse basieren auf dem Beispielalgorithmus FV unter Verwendung von nur einer Bibliothek SEAL v2.3.0-4. Wir möchten weitere Forschung dazu animieren, unsere Experimente unter Verwendung anderer Bibliotheken zu replizieren. Wir können nicht ausschließen, dass dabei kleine aber entscheidende Unterschiede im Hinblick auf die Parameterbestimmung, Laufzeiten und Ergebnisgenauigkeit gefunden werden.

8.2 Ausblick

Zusammengenommen zeigte diese Untersuchung, dass auf mit FV verschlüsselten Daten [67] Clusteranalysen zu korrekten Ergebnissen führen. Allerdings sind sowohl die Laufzeitkosten als auch die korrekte Parameterwahl mit den zugehörigen Testläufen für die Praxis wohl meist noch zu hoch. Einzig bei kleinen Datensätzen mit wenig Dimensionen kann der Schutz privater Daten mit dem FV Algorithmus heute schon empfohlen werden. Ein denkbares Beispiel ist das nachträgliche Clustern einzelner Datenpunkte in ein bestehendes Clustering, sodass nur noch wenige Iterationen für das neue Ergebnis notwendig sind. Praktikern müssen wir allerdings aufgrund der von uns dokumentierten Implementierungsprobleme und der grundsätzlichen Komplexität der Parameterwahl heute (noch) für die meisten Szenarien empfehlen, möglichst so viel PHE Verschlüsselung wie möglich durchzuführen. Dabei ist eine Vorverarbeitung der Klartexte anzuraten, die die Berechnung über die Chiffretexte erheblich erleichtern kann. Auch sind Verfahren denkbar, die homomorphe Verschlüsselungsverfahren mit anderen Methoden zum Schutz privater Daten kombinieren. Interessierten Praktikern empfehlen wir die Zusammenstellung praxiserprobter Verfahren von Thomas Liebig [105]. Zukünftige Forschung dagegen möchten wir animieren, den Einsatz verschiedener LHE voranzutreiben, indem sie besonders die Parameterwahl nachvollziehbar dokumentiert und idealer Weise ihre Daten für Replikationen zur Verfügung stellt. Dann könnten nicht nur Standards sondern auch verlässlich Benschmarks bereitgestellt werden. Die Schaffung einer Datenbank mit den Ergebnissen solcher Replikationsstudien könnte die Forschung zum Praxiseinsatz von LHE Verfahren entscheidend vorantreiben. Dann erst werden konkrete Vergleiche der Laufzeiten ermöglicht und die Auswahl der Verfahren erleichtert. Wir denken, dass unsere Arbeit für Praktiker und Forscher gleichermaßen interessant ist und hoffen etwas dazu beigetragen zu haben, dass die Mauer zwischen Theorie und Praxis hinsichtlich der Anwendung moderner kryptografischer Verfahren etwas verkleinert werden konnte.

Literaturverzeichnis

[1] Acar, A., H. Aksu, S. A. Uluagac und M. Conti: *A Survey on Homomorphic Encryption Schemes: Theory and Implementation*. CoRR, abs/1704.03578, 2017.

[2] Aggarwal, C. C. und P. S. Yu: *Privacy-Preserving Data Mining: Models and Algorithms*. Springer US, 2008.

[3] Agrawal, R. und R. Srikant: *Privacy-preserving Data Mining*. SIGMOD Rec., 29(2):439–450, Mai 2000.

[4] Aguilar-Melchor, C., S. Fau, C. Fontaine, G. Gogniat und R. Sirdey: *Recent advances in homomorphic encryption: A possible future for signal processing in the encrypted domain*. IEEE Signal Processing Magazine, 30(2):108–117, 2013.

[5] Ahila, S. S. und K. L. Shunmuganathan: *State Of Art in Homomorphic Encryption Schemes*. International Journal of Engineering Research and Applications, 4(2):37–43, 2014.

[6] Albrecht, M., S. Bai und L. Ducas: *A Subfield Lattice Attack on Overstretched NTRU Assumptions*. In: Robshaw, M. und J. Katz (Hrsg.): *Advances in Cryptology – CRYPTO 2016*, S. 153–178, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[7] Albrecht, M., M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, J. Hoffstein, K. Lauter, S. Lokam, D. Micciancio et al.: *Homomorphic Encryption Standard*. The Second Homomorphic Encryption Standardization Workshop, MIT, Cambridge, 2018.

[8] Armknecht, F., C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter und M. Strand: *A Guide to Fully Homomorphic Encryption*. IACR Cryptology ePrint Archive, 2015:1192, 2015.

[9] Aslett, L. J. M., P. M. Esperança und C. C. Holmes: *Encrypted statistical machine learning: new privacy preserving methods*. arXiv preprint arXiv:1508.06845, 2015.

[10] Aslett, L. J. M., P. M. Esperança und C. C. Holmes: *A review of homomorphic encryption and software tools for encrypted statistical machine learning*. arXiv preprint arXiv:1508.06574, 2015.

[11] Banerjee, A., C. Peikert und A. Rosen: *Pseudorandom Functions and Lattices*. In: Pointcheval, D. und T. Johansson (Hrsg.): *Advances in Cryptology – EUROCRYPT 2012*, S. 719–737, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[12] Barthelemy, L.: *A brief survey of Fully Homomorphic Encryption, computing on encrypted data)*. `https://blog.quarkslab.com/a-brief-survey-of-fully-homomorphic-encryption-computing-on-encrypted-data.html`, 2016. 25.06.2018.

[13] Bayer, H., M. Aksogan, E. Celik und A. Kondiloglu: *Big data mining and business intelligence trends*. Journal of Asian Business Strategy, 7(1):23, 2017.

[14] Benaloh, J.: *Dense probabilistic encryption*. In: *Proceedings of the workshop on selected areas of cryptography*, S. 120–128, 1994.

[15] Bilogrevic, I., M. Jadliwala, V. Joneja, K. Kalkan, J.-P. Hubaux und I. Aad: *Privacy-Preserving Optimal Meeting Location Determination on Mobile Devices*. IEEE Transactions on Information Forensics and Security, 9(7):1141–1156, July 2014.

[16] Boneh, D., E.-J. Goh und K. Nissim: *Evaluating 2-DNF Formulas on Ciphertexts*. In: Kilian, J. (Hrsg.): *Theory of Cryptography*, S. 325–341, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[17] Bonnoron, G., C. Fontaine, G. Gogniat, V. Herbert, V. Lapôtre, V. Migliore und A. Roux-Langlois: *Somewhat/Fully Homomorphic Encryption: Implementation Progresses and Challenges*. In: El Hajji, S., A. Nitaj und E. M. Souidi (Hrsg.): *Codes, Cryptology and Information Security*, S. 68–82, Cham, 2017. Springer International Publishing.

[18] Bos, J., L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe und D. Stehlé: *CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM*. IACR Cryptology ePrint Archive, 2017:634, 2017.

[19] Bos, J. W., K. Lauter, J. Loftus und M. Naehrig: *Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme*. In: Stam, M. (Hrsg.): *Cryptography and Coding*, S. 45–64, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[20] Boxen, P., D. Fischer und R. Griessl: *Untersuchung des Forschungs-und Implementierungsstands von Homomorphic-Encryption und Anwendbarkeit für Cyber-Physical-Systems*. Masterprojekt, Universität Duisburg Essen, Paluno The Ruhr Institute for Software Technology, Oktober 2016.

[21] Brakerski, Z.: *Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP*. In: Safavi-Naini, R. und R. Canetti (Hrsg.): *Advances in Cryptology – CRYPTO 2012*, S. 868–886, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[22] Brakerski, Z., C. Gentry und S. Halevi: *Packed Ciphertexts in LWE-Based Homomorphic Encryption*. In: Kurosawa, K. und G. Hanaoka (Hrsg.): *Public-Key Cryptography – PKC 2013*, S. 1–13, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[23] Brakerski, Z., C. Gentry und V. Vaikuntanathan: *(Leveled) fully homomorphic encryption without bootstrapping*. ACM Transactions on Computation Theory (TOCT), 6(3):13, 2014.

[24] Brakerski, Z. und V. Vaikuntanathan: *Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages*. In: Rogaway, P. (Hrsg.): *Advances in Cryptology – CRYPTO 2011*, S. 505–524, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[25] Brakerski, Z. und V. Vaikuntanathan: *Efficient Fully Homomorphic Encryption from (Standard) LWE*. SIAM Journal on Computing, 43(2):831–871, 2014.

[26] Brenner, M., W. Dai, S. Halevi, K. Han, A. Jalali, M. Kim, K. Laine, A. Malozemoff, P. Paillier, Y. Polyakov et al.: *A standard api for rlwe-based homomorphic encryption*. Techn. Ber., Technical report, HomomorphicEncryption. org, Redmond WA, 2017.

[27] Brickell, E. F. und Y. Yacobi: *On Privacy Homomorphisms (Extended Abstract)*. In: Chaum, D. und W. L. Price (Hrsg.): *Advances in Cryptology — EUROCRYPT' 87*, S. 117–125, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[28] Buchmann, J.: *Einführung in die Kryptographie*. Springer Spektrum Berlin Heidelberg, 2001.

[29] Bundesamt für Sicherheit in der Informationstechnik (BSI): *Die Lage der IT-Sicherheit in Deutschland 2017*. `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2017.pdf?__blob=publicationFile&v=4`, 2017. 25.08.2018.

[30] Burkschat, M., E. Cramer und U. Kamps: *Beschreibende Statistik: Grundlegende Methoden der Datenanalyse*. Springer-Verlag, 2012.

[31] Chase, M., H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, J. Hoffstein, K. Lauter, S. Lokam, D. Moody, T. Morrison et al.: *Security of homomorphic encryption*. HomomorphicEncryption. org, Redmond WA, Tech. Rep, 2017.

[32] Chen, H., K. Han, Z. Huang, A. Jalali und K. Laine: *Implementation of SEAL*. `http://sealcrypto.org`, 2017. 25.06.2018.

[33] Chen, H., K. Han, Z. Huang, A. Jalali und K. Laine: *Simple Encrypted Arithmetic Library v2.3.0-4*. Techn. Ber., Microsoft Research, Dezember 2017.

[34] Chen, H., K. Laine, R. Player und Y. Xia: *High-Precision Arithmetic in Homomorphic Encryption*. In: Smart, N. P. (Hrsg.): *Topics in Cryptology – CT-RSA 2018*, S. 116–136, Cham, 2018. Springer International Publishing.

[35] Cheon, J. H., J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi und A. Yun: *Batch Fully Homomorphic Encryption over the Integers*. In: Johansson, T. und P. Q. Nguyen (Hrsg.): *Advances in Cryptology – EUROCRYPT 2013*, S. 315–335, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[36] Cheon, J. H., J. Jeong und C. Lee: *An algorithm for NTRU problems and cryptanalysis of the GGH multilinear map without a low-level encoding of zero*. LMS Journal of Computation and Mathematics, 19(A):255–266, 2016.

[37] Cheon, J. H., A. Kim, M. Kim und Y. Song: *Implementation of HEA-AN*. `https://github.com/kimandrik/HEAAN`, 2016. 25.06.2018.

[38] Cheon, J. H., A. Kim, M. Kim und Y. Song: *Homomorphic Encryption for Arithmeti of Approximate Numbers*. In: Takagi, T. und T. Peyrin (Hrsg.): *Advances in Cryptology – ASIACRYPT 2017*, S. 409–437, Cham, 2017. Springer International Publishing.

[39] Cheon, J. H., J. Kim, M. S. Lee und A. Yun: *CRT-based fully homomorphic encryption over the integers*. Information Sciences, 310:149 – 162, 2015.

[40] Cheon, J. H., M. Kim und K. Lauter: *Homomorphic Computation of Edit Distance*. In: Brenner, M., N. Christin, B. Johnson und K. Rohloff (Hrsg.): *Financial Cryptography and Data Security*, S. 194–212, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[41] Cheon, J. H. und D. Stehlé: *Fully Homomophic Encryption over the Integers Revisited*. In: Oswald, E. und M. Fischlin (Hrsg.): *Advances in Cryptology – EUROCRYPT 2015*, S. 513–536, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[42] Chillotti, I., N. Gama, M. Georgieva und M. Izabachène: *Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds*. In: Cheon, J. H. und T. Takagi (Hrsg.): *Advances in Cryptology – ASIACRYPT 2016*, S. 3–33, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[43] Cohen, J. D. und M. J. Fischer: *A robust and verifiable cryptographically secure election scheme*. Yale University. Department of Computer Science, 1985.

[44] Coron, J.-S., T. Lepoint und M. Tibouchi: *Scale-Invariant Fully Homomorphic Encryption over the Integers*. In: Krawczyk, H. (Hrsg.): *Public-Key Cryptography – PKC 2014*, S. 311–328, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[45] Coron, J. S., A. Mandal, D. Naccache und M. Tibouchi: *Fully Homomorphic Encryption over the Integers with Shorter Public Keys*. In: Rogaway, P. (Hrsg.): *Advances in Cryptology – CRYPTO 2011*, S. 487–504, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[46] Coron, J. S., D. Naccache und P. Paillier: *Accelerating Okamoto-Uchiyama public-key cryptosystem*. Electronics Letters, 35(4):291–292, Feb 1999.

[47] Coron, J.-S., D. Naccache und M. Tibouchi: *Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers*. In: Pointcheval, D. und T. Johansson (Hrsg.): *Advances in Cryptology – EUROCRYPT 2012*, S. 446–464, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[48] Costache, A. und N. P. Smart: *Which Ring Based Somewhat Homomorphic Encryption Scheme is Best?*. In: Sako, K. (Hrsg.): *Topics in Cryptology - CT-RSA 2016*, S. 325–340, Cham, 2016. Springer International Publishing.

[49] Costache, A., N. P. Smart und S. Vivek: *Faster Homomorphic Evaluation of Discrete Fourier Transforms*. In: Kiayias, A. (Hrsg.): *Financial Cryptography and Data Security*, S. 517–529, Cham, 2017. Springer International Publishing.

[50] Cramer, R., L. Ducas, C. Peikert und O. Regev: *Recovering Short Generators of Principal Ideals in Cyclotomic Rings*. In: Fischlin, M. und J.-S. Coron (Hrsg.): *Advances in Cryptology – EUROCRYPT 2016*, S. 559–585, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[51] Cramer, R., R. Gennaro und B. Schoenmakers: *A secure and optimally efficient multi-authority election scheme*. European Transactions on Telecommunications, 8(5):481–490, 1997.

[52] Daemen, J. und V. Rijmen: *The design of Rijndael: AES-the advanced encryption standard*. Springer-Verlag Berlin Heidelberg, 2002.

[53] Dai, W.: *Implementation of cuHE (CUDA Homomorphic Encryption Library)*. https://github.com/vernamlab/cuHE, 2017. 25.06.2018.

[54] Dai, W., Y. Doröz und B. Sunar: *Accelerating NTRU based homomorphic encryption using GPUs*. In: *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, S. 1–6, Sept 2014.

[55] Damgård, I. und M. Jurik: *A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System*. In: Kim, K. (Hrsg.): *Public Key Cryptography*, S. 119–136, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[56] Damgård, I. und M. Jurik: *A Length-Flexible Threshold Cryptosystem with Applications*. In: Safavi-Naini, R. und J. Seberry (Hrsg.): *Information Security and Privacy*, S. 350–364, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[57] Damgård, I., V. Pastro, N. Smart und S. Zakarias: *Multiparty Computation from Somewhat Homomorphic Encryption*. In: Safavi-Naini, R. und R. Canetti (Hrsg.): *Advances in Cryptology – CRYPTO 2012*, S. 643–662, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[58] Dasgupta, S.: *CSE 291: Unsupervised learning Lecture 2 - The k-means clustering problem*. https://cseweb.ucsd.edu/~dasgupta/291-unsup/lec2.pdf, 2008. 15.07.2018.

[59] Diffie, W. und M. Hellman: *New directions in cryptography*. IEEE Transactions on Information Theory, 22(6):644–654, Nov 1976.

[60] Doganay, M. C., T. B. Pedersen, Y. Saygin, E. Savaş und A. Levi: *Distributed Privacy Preserving K-means Clustering with Additive Secret Sharing*. In: *Proceedings of the 2008 International Workshop on Privacy and Anonymity in Information Society*, PAIS '08, S. 3–11, New York, NY, USA, 2008. ACM.

[61] Doröz, Y., Y. Hu und B. Sunar: *Homomorphic AES Evaluation using NTRU*. IACR Cryptology ePrint Archive, 2014:39, 2014.

[62] Doröz, Y. und B. Sunar: *Flattening NTRU for Evaluation Key Free Homomorphic Encryption*. IACR Cryptology ePrint Archive, 2016:315, 2016.

[63] Du, Y., L. Gustafson, D. Huang und K. Peterson: *Implementing ML Algorithms with HE*. 2017.

[64] Ducas, L. und D. Micciancio: *FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second*. In: Oswald, E. und M. Fischlin (Hrsg.): *Advances in Cryptology – EUROCRYPT 2015*, S. 617–640, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[65] Dug-Hwan, C., S. Choi und D. Won: *Improvement of Probabilistic Public Key Cryptosystems Using Discrete Logarithm*. In: Kim, K. (Hrsg.): *Information Security and Cryptology — ICISC 2001*, S. 72–80, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[66] El Gamal, T.: *A public key cryptosystem and a signature scheme based on discrete logarithms*. IEEE Transactions on Information Theory, 31(4):469–472, July 1985.

[67] Fan, J. und F. Vercauteren: *Somewhat Practical Fully Homomorphic Encryption*. IACR Cryptology ePrint Archive, 2012:144, 2012.

[68] Fayyad, U., G. Piatetsky-Shapiro und P. Smyth: *From data mining to knowledge discovery in databases*. AI magazine, 17(3):37, 1996.

[69] Fisher, R. A.: *The use of multiple measurements in taxonomic problems*. Annals of Eugenics, 7(2):179–188, 1936.

[70] Fujisaki, E. und T. Okamoto: *How to Enhance the Security of Public-Key Encryption at Minimum Cost*. In: *Public Key Cryptography*, S. 53–68, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[71] Galbraith, S. D.: *Elliptic Curve Paillier Schemes*. Journal of Cryptology, 15(2):129–138, Jan 2002.

[72] Gentry, C.: *A fully homomorphic encryption scheme*. Doktorarbeit, Stanford University, 2009. `crypto.stanford.edu/craig`.

[73] Gentry, C.: *Fully Homomorphic Encryption Using Ideal Lattices*. In: *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, S. 169–178, New York, NY, USA, 2009. ACM.

[74] Gentry, C.: *Computing Arbitrary Functions of Encrypted Data*. Commun. ACM, 53(3):97–105, März 2010.

[75] Gentry, C.: *Computing on the edge of chaos: Structure and randomness in encrypted computation.*. In: *Electronic Colloquium on Computational Complexity (ECCC)*, Bd. 21, S. 106. Citeseer, 2014.

[76] Gentry, C. und S. Halevi: *Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits*. In: *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, S. 107–109, Oct 2011.

[77] Gentry, C. und S. Halevi: *Implementing Gentry's Fully-Homomorphic Encryption Scheme*. In: Paterson, K. G. (Hrsg.): *Advances in Cryptology – EUROCRYPT 2011*, S. 129–148, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[78] Gentry, C., S. Halevi und N. P. Smart: *Fully Homomorphic Encryption with Polylog Overhead*. In: Pointcheval, D. und T. Johansson (Hrsg.): *Advances in Cryptology – EUROCRYPT 2012*, S. 465–482, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[79] Gentry, C., S. Halevi und N. P. Smart: *Homomorphic Evaluation of the AES Circuit*. In: Safavi-Naini, R. und R. Canetti (Hrsg.): *Advances in Cryptology – CRYPTO 2012*, S. 850–867, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[80] Gentry, C., S. Halevi und N. P. Smart: *Homomorphic Evaluation of the AES Circuit (Updated Implementation)*. In: Safavi-Naini, R. und R. Canetti (Hrsg.): *Advances in Cryptology – CRYPTO 2012*, S. 850–867, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[81] Gentry, C., A. Sahai und B. Waters: *Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based*. In: Canetti, R. und J. A. Garay (Hrsg.): *Advances in Cryptology – CRYPTO 2013*, S. 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[82] Goldreich, O., S. Goldwasser und S. Halevi: *Public-key cryptosystems from lattice reduction problems*. In: Kaliski, B. S. (Hrsg.): *Advances in Cryptology — CRYPTO '97*, S. 112–131, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[83] Goldrich, O.: *Foundations of Cryptography, Basic Tools*. Cambridge University Press, 2001.

[84] Goldwasser, S. und S. Micali: *Probabilistic encryption*. Journal of Computer and System Sciences, 28(2):270 – 299, 1984.

[85] Graepel, T., K. Lauter und M. Naehrig: *ML Confidential: Machine Learning on Encrypted Data*. In: Kwon, T., M.-K. Lee und D. Kwon (Hrsg.): *Information Security and Cryptology – ICISC 2012*, S. 1–21, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[86] Gurevich, A. und E. Gudes: *Privacy preserving Data Mining Algorithms without the use of Secure Computation or Perturbation*. In: *2006 10th International Database Engineering and Applications Symposium (IDEAS'06)*, S. 121–128, Dec 2006.

[87] Halevi, S. und V. Shoup: *Faster homomorphic linear transformations in helib*. Techn. Ber., Cryptology ePrint Archive, Report 2018/244, 2018.

[88] Halevi, S. und V. Shoup: *Implementation of HElib*. `https://github.com/shaih/HElib`, 2018. 25.06.2018.

[89] Hallgren, P., M. Ochoa und A. Sabelfeld: *InnerCircle: A parallelizable decentralized privacy-preserving location proximity protocol*. In: *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, S. 1–6, July 2015.

[90] Han, J., J. Pei und M. Kamber: *Data mining: concepts and techniques*. Elsevier, 3 Aufl., 20112.

[91] Hrestak, D. und S. Picek: *Homomorphic encryption in the cloud*. In: *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, S. 1400–1404, May 2014.

[92] Inan, A., S. V. Kaya, Y. Saygin, E. Savaş, A. A. Hintoğlu und A. Levi: *Privacy preserving clustering on horizontally partitioned data*. Data & Knowledge Engineering, 63(3):646–666, 2007.

[93] Jain, A. K.: *Data clustering: 50 years beyond K-means*. Pattern Recognition Letters, 31(8):651 – 666, 2010. Award winning papers from the 19th International Conference on Pattern Recognition (ICPR).

[94] Jain, A. K. und R. C. Dubes: *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[95] Joye, M. und B. Libert: *Efficient Cryptosystems from 2^k-th Power Residue Symbols*. In: Johansson, T. und P. Q. Nguyen (Hrsg.): *Advances in Cryptology – EUROCRYPT 2013*, S. 76–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[96] Kargupta, H., S. Datta, Q. Wang und K. Sivakumar: *On the privacy preserving properties of random data perturbation techniques*. In: *Third IEEE International Conference on Data Mining*, S. 99–106, Nov 2003.

[97] Karpfinger, C. und H. Kiechle: *Kryptologie Algebraische Methoden und Algorithmen*. Vieweg+Teubner, 1 Aufl., 2010.

[98] Keogh, E. und S. Kasetty: *On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration*. Data Mining and Knowledge Discovery, 7(4):349–371, Oct 2003.

[99] Kerckhoffs, A.: *La cryptographie militaire, ou, Des chiffres usités en temps de guerre: avec un nouveau procédé de déchiffrement applicable aux systèmes à double clef*. Librairie militaire de L. Baudoin, 1883.

[100] Khedr, A., G. Gulak und V. Vaikuntanathan: *SHIELD: scalable homomorphic implementation of encrypted data-classifiers*. IEEE Transactions on Computers, 65(9):2848–2858, 2016.

[101] Kim, M. und K. Lauter: *Private genome analysis through homomorphic encryption*. BMC Medical Informatics and Decision Making, 15(5):S3, Dec 2015.

[102] Kirchner, P. und P.-A. Fouque: *Comparison between Subfield and Straightforward Attacks on NTRU*. IACR Cryptology ePrint Archive, 2016:717, 2016.

[103] Lepoint, T. und M. Naehrig: *A Comparison of the Homomorphic Encryption Schemes FV and YASHE*. In: Pointcheval, D. und D. Vergnaud (Hrsg.): *Progress in Cryptology – AFRICACRYPT 2014*, S. 318–335, Cham, 2014. Springer International Publishing.

[104] Lieber, D., M. Stolpe, B. Konrad, J. Deuse und K. Morik: *Quality Prediction in Interlinked Manufacturing Processes based on Supervised & Unsupervised Machine Learning*. Procedia CIRP, 7:193 – 198, 2013. Forty Sixth CIRP Conference on Manufacturing Systems 2013.

[105] Liebig, T.: *Report on Data Privacy*. Techn. Ber. H2020-688380 D4.1, VAVEL Consortium, Dortmund, Germany, May 2017.

[106] Liebig, T., M. Stolpe und K. Morik: *Distributed Traffic Flow Prediction with Label Proportions: From in-Network towards High Performance Computation with MPI*. In: Andrienko, G., D. Gunopulos, I. Katakis, T. Liebig, S. Mannor, K. Morik und F. Schnitzler (Hrsg.): *Proceedings of the 2nd International Workshop on Mining Urban Data (MUD2)*, Bd. 1392 d. Reihe *CEUR Workshop Proceedings*, S. 36–43. CEUR-WS, 2015.

[107] Lindell, Y. und B. Pinkas: *Privacy Preserving Data Mining*. In: Bellare, M. (Hrsg.): *Advances in Cryptology — CRYPTO 2000*, S. 36–54, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[108] Liu, D., E. Bertino und X. Yi: *Privacy of Outsourced K-means Clustering*. In: *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, S. 123–134, New York, NY, USA, 2014. ACM.

[109] Liu, Q., P. Li, W. Zhao, W. Cai, S. Yu und V. C. M. Leung: *A Survey on Security Threats and Defensive Techniques of Machine Learning: A Data Driven View*. IEEE Access, 6:12103–12117, 2018.

[110] Lloyd, S.: *Least squares quantization in PCM*. IEEE transactions on information theory, 28(2):129–137, 1982.

[111] López-Alt, A., E. Tromer und V. Vaikuntanathan: *On-the-fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption*. In: *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, S. 1219–1234, New York, NY, USA, 2012. ACM.

[112] Lu, W., S. Kawasaki und J. Sakuma: *Using Fully Homomorphic Encryption for Statistical Analysis of Categorical, Ordinal and Numerical Data*. IACR Cryptology ePrint Archive, 2016:1163, 2016.

[113] Lyubashevsky, V., C. Peikert und O. Regev: *On Ideal Lattices and Learning with Errors over Rings*. In: Gilbert, H. (Hrsg.): *Advances in Cryptology – EUROCRYPT 2010*, S. 1–23, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[114] Martins, P., L. Sousa und A. Mariano: *A Survey on Fully Homomorphic Encryption: An Engineering Perspective*. ACM Comput. Surv., 50(6):83:1–83:33, Dez. 2017.

[115] Masulli, F. und S. Rovetta: *Clustering High-Dimensional Data*. In: Masulli, F., A. Petrosino und S. Rovetta (Hrsg.): *Clustering High–Dimensional Data*, S. 1–13, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[116] Melchor, C. A., P. Gaborit und J. Herranz: *Additively Homomorphic Encryption with d-Operand Multiplications*. In: Rabin, T. (Hrsg.): *Advances in Cryptology – CRYPTO 2010*, S. 138–154, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[117] Mierswa, I. und K. Morik: *Merkmalsextraktion aus Audiodaten Evolutionäre Aufzucht von Methodenbäumen*. Informatik-Spektrum, 28(5):381–388, Oct 2005.

[118] Migliore, V., G. Bonnoron und C. Fontaine: *Determination and exploration of practical parameters for the latest Somewhat Homomorphic Encryption (SHE) Schemes*. working paper or preprint, Okt. 2016.

[119] Miller, V. S.: *The Weil Pairing, and Its Efficient Calculation*. Journal of Cryptology, 17(4):235–261, Sep 2004.

[120] Naccache, D. und J. Stern: *A New Public Key Cryptosystem Based on Higher Residues*. In: *Proceedings of the 5th ACM Conference on Computer and Communications Security*, CCS '98, S. 59–66, New York, NY, USA, 1998. ACM.

[121] Naehrig, M., K. Lauter und V. Vaikuntanathan: *Can Homomorphic Encryption Be Practical?*. In: *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, S. 113–124, New York, NY, USA, 2011. ACM.

[122] Okamoto, T. und S. Uchiyama: *Security of an identity-based cryptosystem and the related reductions*. In: Nyberg, K. (Hrsg.): *Advances in Cryptology — EUROCRYPT'98*, S. 546–560, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[123] Oliveira, S. R. M. und O. R. Zaiane: *Privacy preserving clustering by data transformation*. Journal of Information and Data Management, 1(1):37, 2010.

[124] Öztürk, E., Y. Doröz, B. Sunar und E. Savas: *Accelerating Somewhat Homomorphic Evaluation using FPGAs*. IACR Cryptology ePrint Archive, 2015:294, 2015.

[125] Paar, C. und J. Pelzl: *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer Berlin Heidelberg, 2009.

[126] Paillier, P.: *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. In: Stern, J. (Hrsg.): *Advances in Cryptology — EUROCRYPT '99*, S. 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[127] Paillier, P.: *Trapdooring Discrete Logarithms on Elliptic Curves over Rings*. In: Okamoto, T. (Hrsg.): *Advances in Cryptology — ASIACRYPT 2000*, S. 573–584, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[128] Paillier, P. et al.: *Implementation of NFLlib*. `https://github.com/quarkslab/NFLlib`, 2016. 25.06.2018.

[129] Paillier, P. et al.: *Implementation of NFLlib*. `https://github.com/CryptoExperts/FV-NFLlib`, 2016. 25.06.2018.

[130] Paillier, P. und D. Pointcheval: *Efficient Public-Key Cryptosystems Provably Secure Against Active Adversaries*. In: am, K.-Y., E. Okamoto und C. Xing (Hrsg.): *Advances in Cryptology - ASIACRYPT'99*, S. 165–179, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[131] Panackal, J. J. und A. S. Pillai: *Privacy preserving data mining: an extensive survey*. In: *ACEEE International Conference on Multimedia Processing, Communication and Information Technology*, 2013.

[132] Parmar, P. V., S. B. Padhar, S. N. Patel, N. I. Bhatt und R. H. Jhaveri: *Survey of various homomorphic encryption algorithms and schemes*. International Journal of Computer Applications, 91(8), 2014.

[133] Pedersen, T. B., Y. Saygin und E. Savaş: *Secret charing vs. encryption-based techniques for privacy preserving data mining*. Eurostat, 2007.

[134] Rackoff, C. und D. R. Simon: *Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack*. In: Feigenbaum, J. (Hrsg.): *Advances in Cryptology — CRYPTO '91*, S. 433–444, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[135] Regev, O.: *On Lattices, Learning with Errors, Random Linear Codes, and Cryptography*. J. ACM, 56(6):34:1–34:40, Sep. 2009.

[136] Reinsel, D., J. Gantz und J. Rydning: *Data Age 2025: The Evolution of Data to Life-Critical Don't Focus on Big Data; Focus on Data That's Big*. IDC, Seagate, April, 2017.

[137] Rivest, R. L., L. Adleman und M. L. Dertouzos: *On data banks and privacy homomorphisms*. Foundations of secure computation, 4(11):169–180, 1978.

[138] Rivest, R. L. und B. Kaliski: *RSA Problem*. In: van Tilborg, H. C. A. und S. Jajodia (Hrsg.): *Encyclopedia of Cryptography and Security*, S. 1065–1069. Springer US, Boston, MA, 2011.

[139] Rivest, R. L., A. Shamir und L. M. Adleman: *A Method for Obtaining Digital Signatures and Public-key Cryptosystems*. Commun. ACM, 21(2):120–126, Feb. 1978.

[140] Roiger, R. J.: *Data mining: a tutorial-based primer*. CRC Press, 2017.

[141] Ryan, G., Y. Polyakov und K. Rohloff: *Implementation of Palisade*. `https://git.njit.edu/groups/palisade`, 2018. 25.06.2018.

[142] Salomon, D.: *Data privacy and security: encryption and information hiding*. Springer Science & Business Media, 2003.

[143] Sander, T., A. Young und M. Yung: *Non-interactive cryptocomputing for NC1*. In: *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, S. 554–566, 1999.

[144] Sander, T., A. Young und M. Yung: *Non-interactive cryptocomputing for NC/sup 1/*. In: *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, S. 554–566, Oct 1999.

[145] Schmidt, M.: *Coresets and streaming algorithms for the k-means problem and related clustering objectives*. Doktorarbeit, Universitätsbibliothek Dortmund, 2014.

[146] Schmidt-Samoa, K. und T. Takagi: *Paillier's Cryptosystem Modulo p^2q and Its Applications to Trapdoor Commitment Schemes*. In: Dawson, E. und S. Vaudenay (Hrsg.): *Progress in Cryptology – Mycrypt 2005*, S. 296–313, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[147] Schneier, B.: *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[148] Schoenmakers, B.: *Verifiable Secret Sharing*. In: van Tilborg, H. C. A. und S. Jajodia (Hrsg.): *Encyclopedia of Cryptography and Security*, S. 1357–1358. Springer US, Boston, MA, 2011.

[149] Shannon, C. E.: *Communication Theory of Secrecy Systems**. Bell System Technical Journal, 28(4):656–715, 1949.

[150] Shanthi, A. S. und M. Karthikeyan: *A review on privacy preserving data mining*. In: *2012 IEEE International Conference on Computational Intelligence and Computing Research*, S. 1–4, Dec 2012.

[151] Smart, N. P. und F. Vercauteren: *Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes*. In: Nguyen, P. Q. und D. Pointcheval (Hrsg.): *Public Key Cryptography – PKC 2010*, S. 420–443, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[152] Stehlé, D. und R. Steinfeld: *Making NTRU as Secure as Worst-Case Problems over Ideal Lattices*. In: Paterson, K. G. (Hrsg.): *Advances in Cryptology – EUROCRYPT 2011*, S. 27–47, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[153] Steinhaus, H.: *Sur la division des corp materiels en parties*. Bull. Acad. Polon. Sci, 1:801–804, 1956.

[154] Stinson, D. R.: *Cryptography: theory and practice*. Chapman & Hall / CRC press, 2005.

[155] Vaidya, J. und C. Clifton: *Privacy-preserving K-means Clustering over Vertically Partitioned Data*. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, S. 206–215, New York, NY, USA, 2003. ACM.

[156] Vaikuntanathan, V.: *Computing Blindfolded: New Developments in Fully Homomorphic Encryption*. In: *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, S. 5–16, Oct 2011.

[157] van Dijk, M., C. Gentry, S. Halevi und V. Vaikuntanathan: *Fully Homomorphic Encryption over the Integers*. In: Gilbert, H. (Hrsg.): *Advances in Cryptology – EUROCRYPT 2010*, S. 24–43, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[158] von Rauchhaupt, U. und R. J. Roiger: *Die Macht des Nein*. `http://www.faz.net/aktuell/wissen/computer-mathematik/mathematik-was-es-mit-p-np-auf-sich-hat-15168441/mathematik-was-es-mit-p-np-15168507.html`, Aug 2017. 24.06.2018.

[159] Wiese, L., D. Homann, T. Waage und M. Brenner: *Homomorphe Verschlüsselung für Cloud-Datenbanken: Übersicht und Anforderungsanalyse*. In: Langweg, H., M. Meier, B. C. Witt und D. Reinhardt (Hrsg.): *Sicherheit 2018*, S. 221–234, Bonn, 2018. Gesellschaft für Informatik e.V.

[160] Wu, D. und J. Haven: *Using homomorphic encryption for large scale statistical analysis*. Technical Report: `cs.stanford.edu/people/dwu4/papers/FHESIReport.pdf`, Leland Stanford Junior University, 2012.

[161] Wu, X., V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand und D. Steinberg: *Top 10 algorithms in data mining*. Knowledge and Information Systems, 14(1):1–37, Jan 2008.

[162] Yang, Z., S. Zhong und R. N. Wright: *Privacy-Preserving Classification of Customer Data without Loss of Accuracy*, S. 92–102. Society for Industrial and Applied Mathematics, 2005.

[163] Yao, A. C.: *Protocols for secure computations*. In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, S. 160–164, Nov 1982.

[164] Yucheng, Y., S. Ling und E. Chi: *Investigation on distributed k-means clustering algorithm of homomorphic encryption*. Computer Technology and Development, 27(2):81–85, 2017.

[165] Zhong, G., I. Goldberg und U. Hengartner: *Louis, Lester and Pierre: Three Protocols for Location Privacy*. In: Borisov, N. und P. Golle (Hrsg.): *Privacy Enhancing Technologies*, S. 62–76, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

Anhang A

Ergänzende Abbildungen

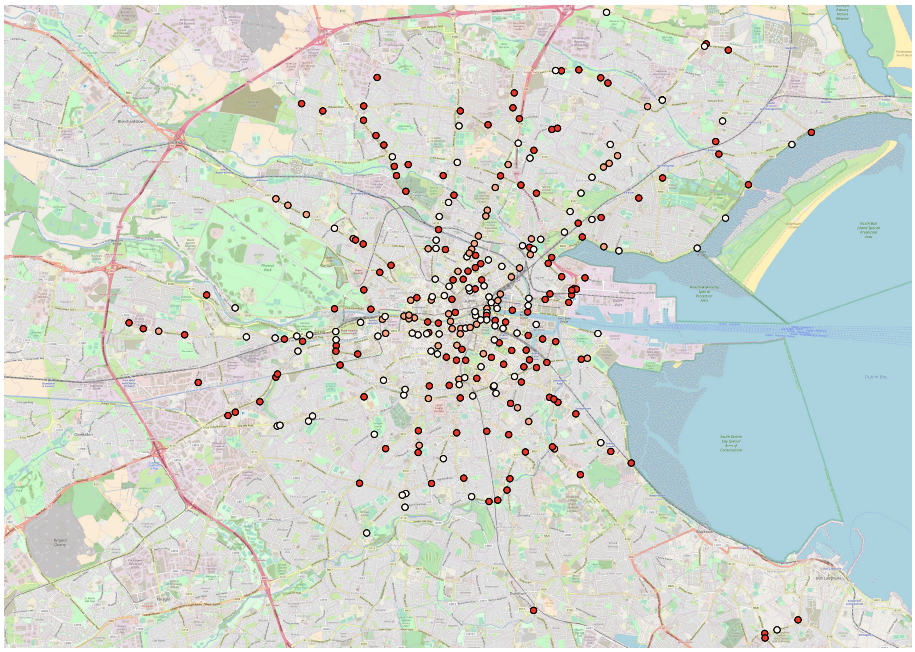


Abbildung A.1: Ergebnis Clusteranalyse auf Dublin Daten mit 48 Messungen. $k = 3$ Cluster, $d = 48$ Dimensionen.

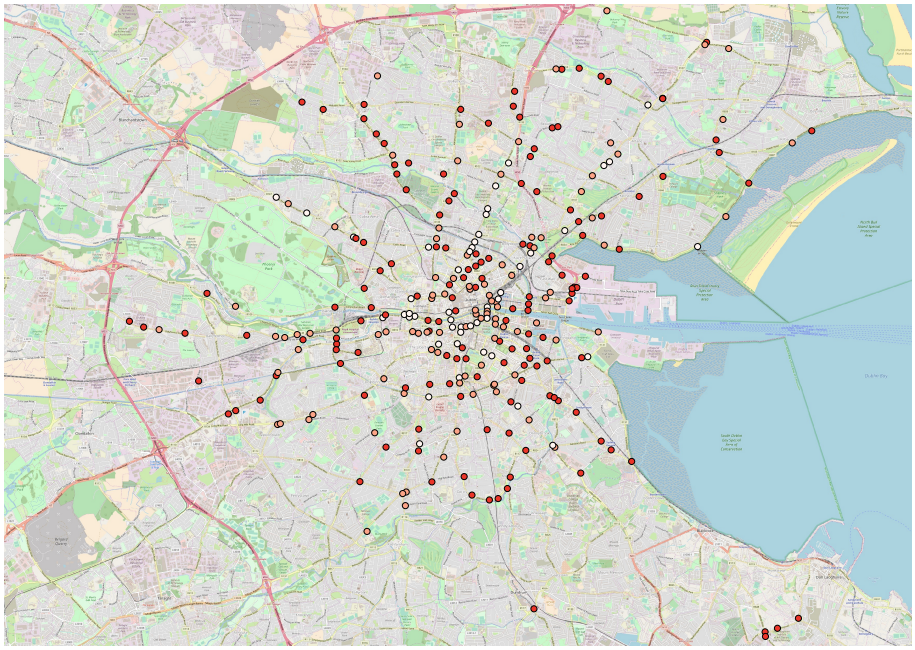


Abbildung A.2: Ergebnis Clusteranalyse auf Dublin Daten mit 96 Messungen. $k = 3$ Cluster, $d = 96$ Dimensionen.

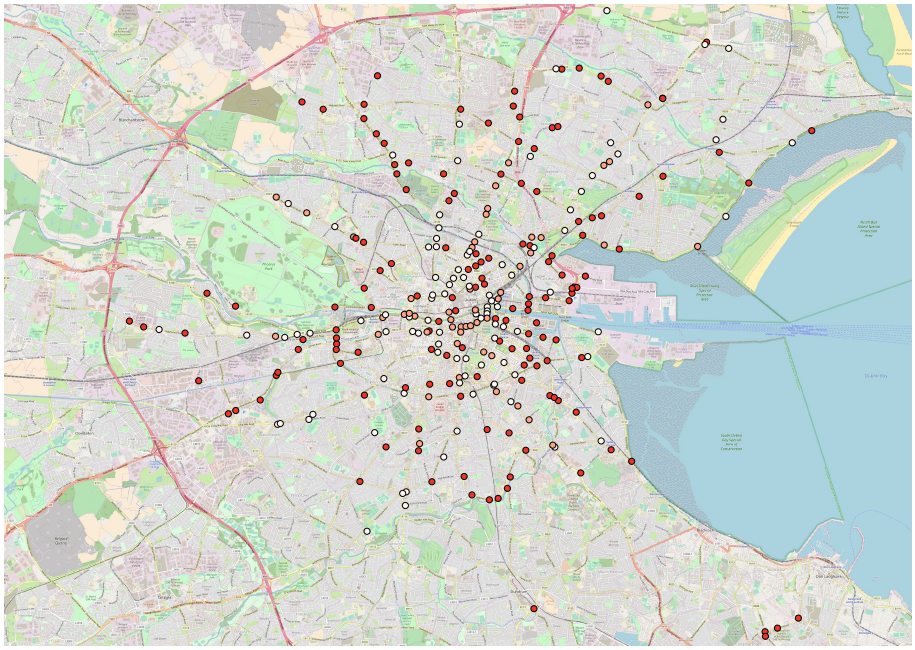


Abbildung A.3: Ergebnis Clusteranalyse auf Dublin Daten mit 144 Messungen. $k = 3$ Cluster, $d = 144$ Dimensionen.

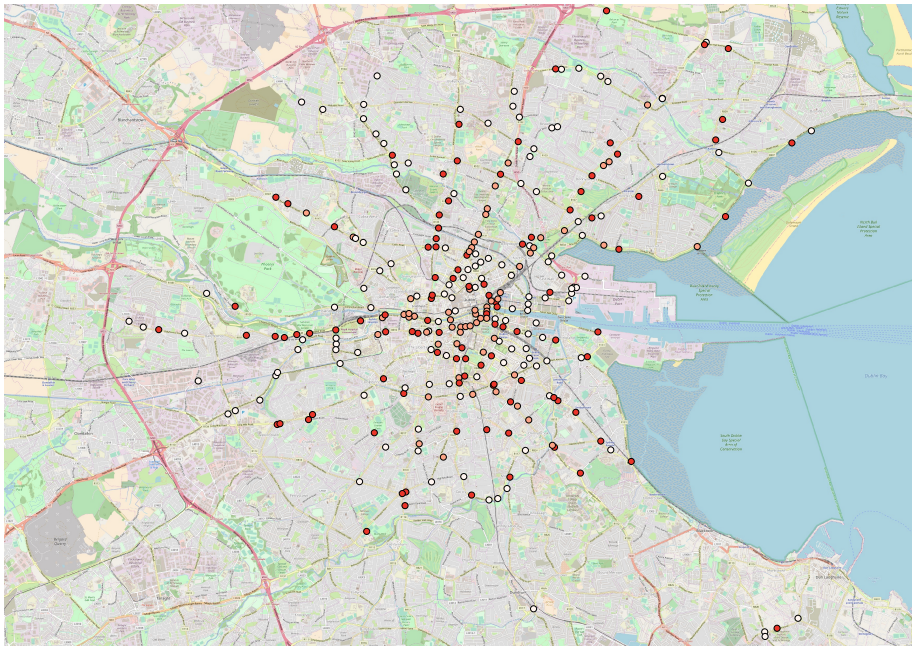


Abbildung A.4: Ergebnis Clusteranalyse auf Dublin Daten mit 192 Messungen. $k = 3$ Cluster, $d = 192$ Dimensionen.

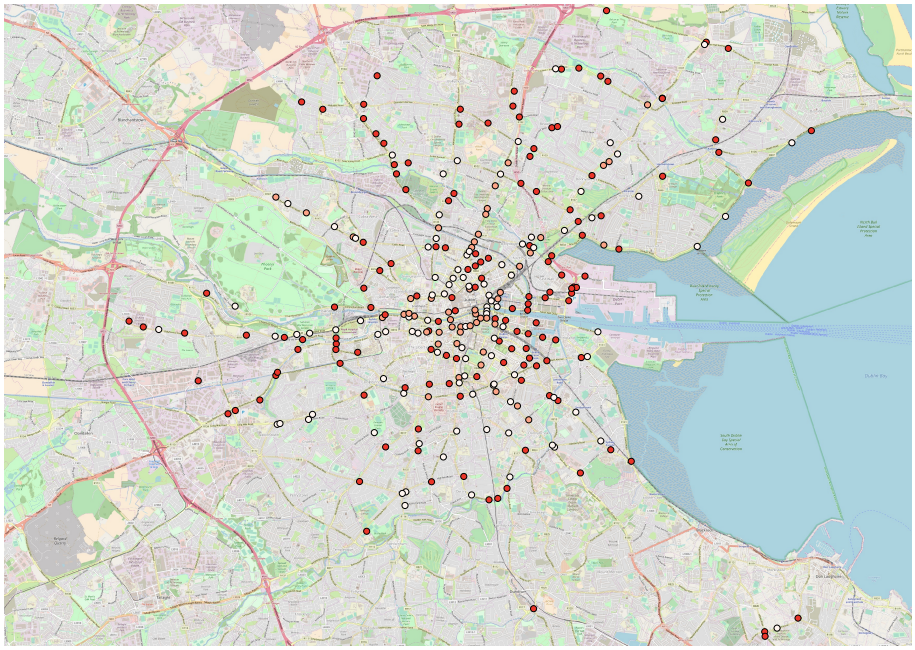


Abbildung A.5: Ergebnis Clusteranalyse auf Dublin Daten mit 240 Messungen. $k = 3$ Cluster, $d = 240$ Dimensionen.

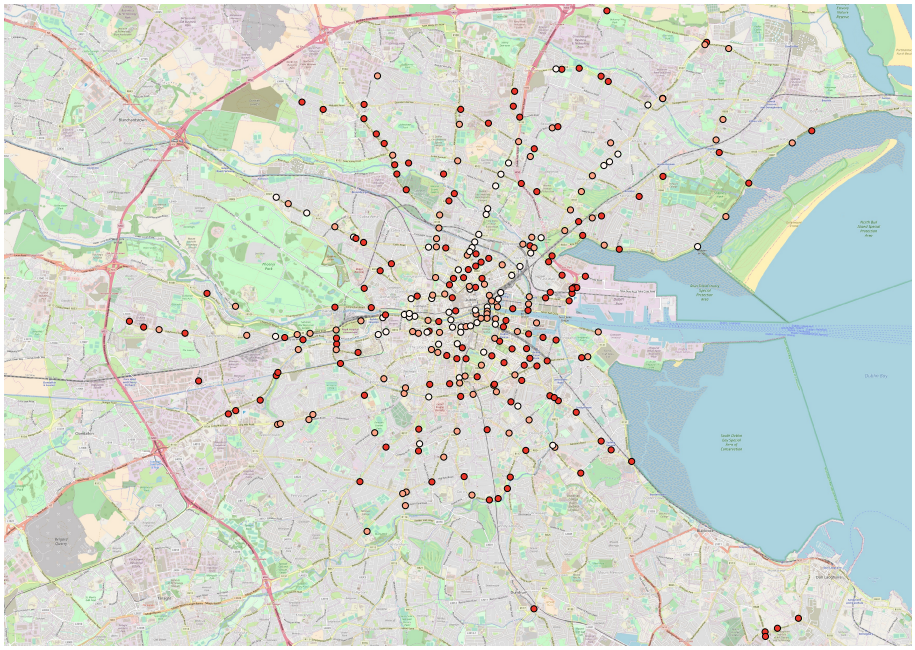


Abbildung A.6: Ergebnis Clusteranalyse auf Dublin Daten mit 288 Messungen. $k = 3$ Cluster, $d = 288$ Dimensionen.

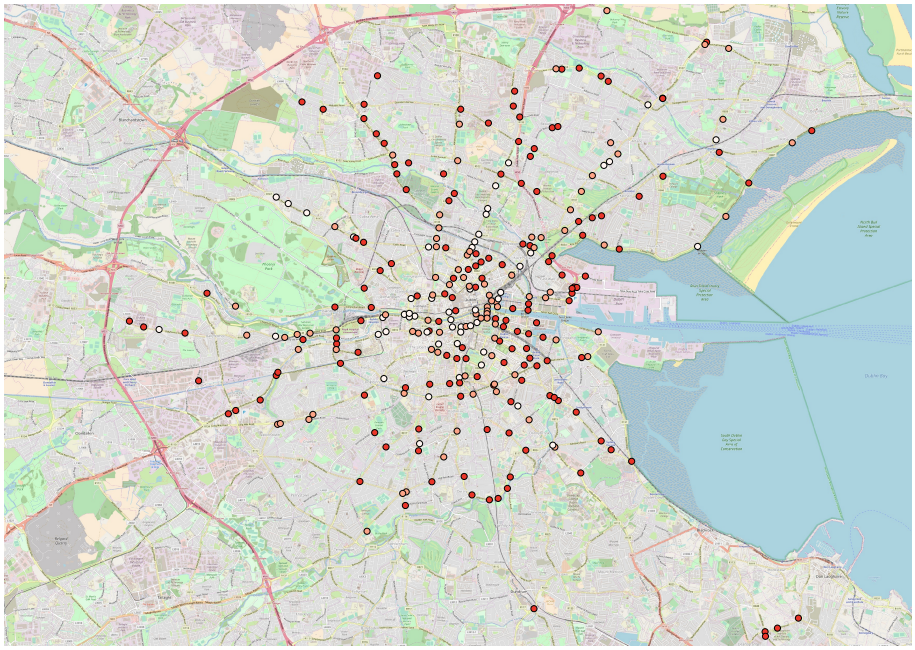


Abbildung A.7: Ergebnis Clusteranalyse auf Dublin Daten mit 336 Messungen. $k = 3$ Cluster, $d = 336$ Dimensionen.