cyberagentur

HOCHSCHULE MITTWEIDA
University of Applied Sciences

BACHELOR-ARBEIT

Herr
Hans Daus

Verschlüsseltes Rechnen: Benchmarking vollständig homomorpher Verschlüsselungsalgorithmen

Mittweida, August 2023

Fakultät **Angewandte Computer- und Biowissenschaften**

BACHELORARBEIT

Verschlüsseltes Rechnen: Benchmarking vollständig homomorpher Verschlüsselungsalgorithmen

Autor:
Hans Daus

Studiengang:
Allgemeine und digitale Forensik

Seminargruppe:
FO20w1-B

Erstprüfer:
Prof. Dr. rer. nat. Klaus Dohmen

Zweitprüfer:
Prof. Dr. Jürgen Freudenberger

Einreichung:
Mittweida, 22.08.2023

Verteidigung/Bewertung:
Mittweida, 2023

Faculty of **Applied Computer Sciences and Biosciences**

BACHELOR THESIS

Encrypted Computing: Benchmarking of fully homomorphic Encryption-Algorithms

Author:
Hans Daus

Course of Study:
General and Digital Forensic Science

Seminar Group:
FO20w1-B

First Examiner:
Prof. Dr. rer. nat. Klaus Dohmen

Second Examiner:
Prof. Dr. Jürgen Freudenberger

Submission:
Mittweida, 22.08.2023

Defense/Evaluation:
Mittweida, 2023

Bibliografische Beschreibung:

Daus, Hans:
Verschlüsseltes Rechnen: Benchmarking vollständig homomorpher Verschlüsselungsalgorithmen. – 2023. – 45 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften, Bachelorarbeit, 2023.

Referat:

In Zeiten, in denen Unternehmen rechenintensive Anwendungen auf externe Server auslagern, gewinnt Cloudcomputing immer mehr an Bedeutung. Das Problem dabei ist, um Operationen auf herkömmlichen, verschlüsselten, ausgelagerten Daten ausführen zu können, müssen diese zuerst entschlüsselt werden. Vertrauliche Daten liegen in dem Zeitraum der Bearbeitung unverschlüsselt vor, was zu einem Datenschutz- und Sicherheitsproblem führt. Dieses Problem kann durch vollständig homomorphe Verschlüsselungen gelöst werden. Diese moderne Verschlüsselungstechnik erlaubt das Ausführen von Operationen auf verschlüsselten Daten und wird in dieser Arbeit grundlegend vorgestellt. Weiterhin werden Bibliotheken, die homomorphe Verschlüsselungen implementieren, vorgestellt und mittels Benchmarking miteinander verglichen. Anschließend wird ein Anwendungsbeispiel formuliert, das die Vor- und Nachteile homomorpher Verschlüsselungen simulieren soll. Die Implementierung des Anwendungsbeispiels erfolgt mithilfe der im Benchmarking ermittelten leistungsstärksten Bibliothek - der Microsoft SEAL Bibliothek. Mit den in dieser Arbeit erlangten Erkenntnissen soll der Einstieg in die komplexe Thematik der homomorphen Verschlüsselungen vereinfacht werden und gleichzeitig zur Auseinandersetzung mit dieser Verschlüsselungsmethodik angeregt werden.

Abstract:

At a time when companies are outsourcing compute-intensive applications to external servers, cloud computing is becoming increasingly important. The problem is that in order to perform operations on traditionally encrypted outsourced data, it must first be decrypted. Confidential data is unencrypted while it is being processed, creating a privacy and security problem. This problem can be solved by using fully homomorphic encryption. This modern encryption technique allows to perform operations on encrypted data and is introduced in this thesis. Furthermore, libraries implementing homomorphic encryption are presented and compared by means of benchmarking. Finally, an application example is formulated to simulate the advantages and disadvantages of homomorphic encryption. The implementation of the application example is done using the best performing library identified in the benchmarking, the Microsoft SEAL library. The knowledge gained from this work is intended to simplify the introduction to the complex topic of homomorphic encryption and at the same time to stimulate discussion about this encryption method.

Inhaltsverzeichnis

Abbildungs- und Tabellenverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Vorwort

Im Interesse einer besseren Lesbarkeit der Texte und der Barrierefreiheit wird auf Sonderzeichen als Ausdruck einer gendergerechten Sprache und eine geschlechtsneutrale Differenzierung in dieser Bachelorarbeit verzichtet. Dies impliziert keinesfalls eine Benachteiligung irgendeines Geschlechts. Es wird Wert darauf gelegt, dass sich in allen Fällen Menschen jeden Geschlechts angesprochen fühlen. Gleichzeitig beinhaltet die angewandte Sprachform keine Wertung.

Danksagung

Nachdem ich bereits mein Praktikum bei der Agentur für Innovation in der Cybersicherheit absolvieren durfte, möchte ich mich für die anschließende Betreuung während der Bachelorarbeit bei allen Mitarbeitern bedanken, die mich auf vielfältige Weise unterstützt haben. Vielen Dank für die freundliche und kollegiale Aufnahme, die Einbeziehung in die betrieblichen Abläufe und die Bereitstellung der technischen Ausrüstung. Das gute Arbeitsklima in der Cyberagentur hat mich sehr motiviert.

Ganz besonders bedanke ich mich bei Dr. Matthias Minihold für die fachspezifische Unterstützung in der Kryptologie. Er stand im gesamten Bachelorarbeitszeitraum zur Verfügung und hatte den aktuellen Arbeitsstand immer im Blick. Seine fachlichen Hinweise waren für mich sehr wertvoll.

Ein herzliches Dankeschön geht an Rolf Wittmann für die Betreuung und Unterstützung seit Beginn des Praktikums. Auch während der Bachelorarbeit stand er mir mit Rat und Tat zur Seite und hatte für jedes mathematische Problem einen Lösungsansatz.

Ganz herzlich bedanke ich mich auch bei Prof. Dr. Jürgen Freudenberger als meinen ersten Ansprechpartner in der Cyberagentur. Er organisierte das Praktikum, unterstützte mich bei der Themenwahl der Bachelorarbeit, war während der Bachelorarbeit immer als Ansprechpartner verfügbar und ebnete den Weg für meine weitere Arbeit in der Cyberagentur. Ich freue mich auf die zukünftige Zusammenarbeit.

Weiterhin bedanke ich mich bei Herrn Prof. Dr. Dohmen für die Betreuung von Seiten der Hochschule Mittweida, insbesondere für die gute Verfügbarkeit und die schnelle Beantwortung meiner Rückfragen.

1 Einleitung

Schon vor ca. 2500 Jahren benutzten Spartaner die sogenannte Skytale – einen Holzstab, der es ermöglichte, Nachrichten zu verschlüsseln (siehe Abbildung 1.1). Im Laufe der Zeit haben sich nicht nur die Formen der Kommunikation, sondern auch die darauf angewendeten Verschlüsselungstechniken weiterentwickelt. Neben den klassischen symmetrischen Verschlüsselungsverfahren (wie der Skytale, die aufgrund der Verwendung eines Holzstamms mit gleichem Durchmesser als Schlüssel ein symmetrisches Verfahren ist) etablierten sich auch moderne Formen der Verschlüsselung, die sogar das Rechnen mit verschlüsselten Daten erlauben. In den letzten 30 Jahren hat sich der Kryptographie-Begriff immer weiterentwickelt. Heutzutage werden mit dem Begriff der modernen Kryptographie im Allgemeinen Techniken bezeichnet, „die die Sicherheit digitaler Informationen, Transaktionen und verteilter Rechensysteme schützen [1].“

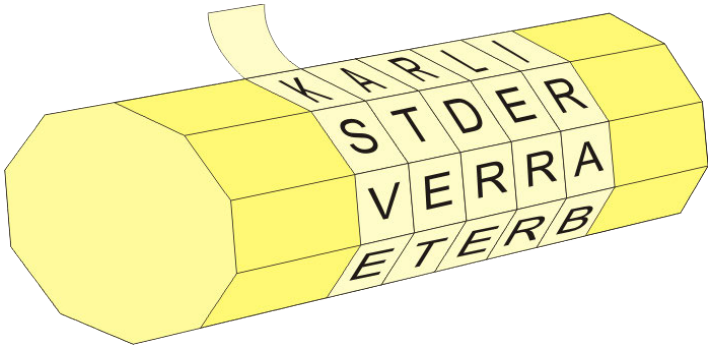

Abbildung 1.1: Verschlüsselung mittels Skytale [2]

Das Ziel von Verschlüsselungen ist es, Vertraulichkeit, Authentizität und Integrität von Daten zu sichern. Der Schutz der Vertraulichkeit bedeutet, dass Daten nur für denjenigen in erkennbarer Form vorliegen, für den sie bestimmt sind. Währenddessen gewährleistet die Authentizität die Echtheit des Absenders. Unter Datenintegrität ist zu verstehen, dass die Nachricht „auf dem Weg vom Absender zum Empfänger nicht unbemerkt durch Dritte verändert werden [3]“ kann.

Grundsätzlich besteht ein Verschlüsselungsverfahren aus zwei Prozessen. Im Verschlüsselungsprozess (engl.: Encryption) wird ein Klartext mithilfe eines Schlüssels in einen Geheimtext umgewandelt. Gegenläufig dazu wird im Entschlüsselungsprozess (engl.: Decryption) ein Geheimtext mithilfe eines Schlüssels in einen Klartext entschlüsselt. In asymmetrischen Verschlüsslungsverfahren ist der Schlüssel für die Verschlüsselung ungleich dem Schlüssel für die Entschlüsselung. Sie bilden die Basis für das sogenannte Public-Key-Verfahren, siehe Abbildung 1.2. Die Besonderheit dieses Verfahrens ist, dass ein Schlüssel öffentlich bekannt gegeben werden kann. Dadurch können zwei Parteien kommunizieren, „ohne dass diese sich zuvor über einen ungesicherten Kanal auf einen gemeinsamen Schlüssel einigen müssen, wie es bei symmetrischen Verfahren notwendig ist [4].“

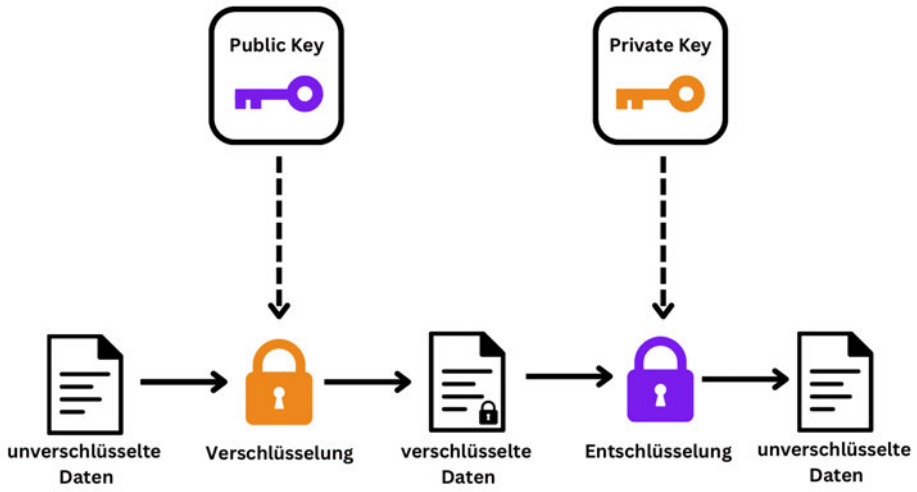

Abbildung 1.2: Public Key Verfahren

Cloud Computing zählt zu einem immer wichtiger werdenden Anwendungsgebiet der Kryptologie. Es ist „das dynamisch an den Bedarf angepasste Anbieten, Nutzen und Abrechnen von IT-Dienstleistungen über ein Netz.[...] Die Spannbreite der im Rahmen von Cloud Computing angebotenen Dienstleistungen umfasst das komplette Spektrum der Informationstechnik und beinhaltet unter anderem Infrastruktur (z. B. Rechenleistung, Speicherplatz), Plattformen und Software [5].“
Sofern die Daten unverschlüsselt vorliegen, ist der Cloud-Dienst in der Lage, Dateninhalte herauszufinden. Dies gefährdet die Vertraulichkeit der Daten, weshalb die Daten verschlüsselt werden müssen. In vielen Fällen müssen Server aber auf den ausgelagerten Daten Operationen ausführen. Mit einem herkömmlichen Verschlüsselungsschema würde dies bedeuten, dass die verschlüsselten Daten erst entschlüsselt werden müssen, damit Operationen, wie z.B. Additionen, auf ihnen ausgeführt werden können. Dadurch ist der Schutz der Vertraulichkeit nicht mehr gewährleistet, da die Verschlüsselung nicht mehr angewendet wird. Um diese Problematik zu lösen, empfiehlt sich die Verwendung von homomorphen Verschlüsselungen. Diese Form der Verschlüsselung ermöglicht das Ausführen von Operationen auf Verschlüsselungen, ohne dass diese vorher entschlüsselt werden müssen.

In dieser Bachelorarbeit „Verschlüsseltes Rechnen: Benchmarking vollständiger homomorpher Verschlüsselungsalgorithmen“ soll ein grundlegender Überblick über die weitläufige Thematik der homomorphen Verschlüsselungen (eng. Homomorphic Encryption) dargeboten werden. Des Weiteren werden zwei Bibliotheken für homomorphe Verschlüsselungen vorgestellt und anhand ihrer bereitgestellten Funktionen und der benötigten Laufzeiten der einzelnen Verfahren miteinander verglichen. Ziel des Vergleiches ist es, eine Bibliothek zu finden, welche den Usecase einer Online-Wahl besser und schneller implementiert. Mit der Simulation einer Online-Wahl wird ein Ausblick auf einen möglichen Anwendungsbereich von homomorphen Verschlüsselungen gegeben. Außerdem werden Chancen und Risiken der Durchführung von Online-Wahlen anstelle von herkömmlichen Papier-Wahlen benannt.

2 Grundlagen homomorpher Verschlüsselungen

Im Folgenden werden die homomorphen Verschlüsselungen grundlegend erklärt. Zuerst wird ein Überblick über die in der Arbeit verwendeten Fachbegriffe und Schreibweisen gegeben. Danach werden die mathematischen Grundlagen die zum Verständnis von Homomorphen Verschlüsselungen benötigt werden erläutert. Weiterhin wird auf die Entstehung der Homomorphen Verschlüsselungen eingegangen und eine damit einhergehende Abstufung beleuchtet. Zum Schluss wird das zugrunde liegende mathematische Problem des Verschlüsselungsprozesses, das Learning with Errors erklärt.

2.1 Notation

Im Tabelle 2.1 werden die in der Arbeit verwendeten Fachbegriffe und Schreibweisen dargestellt.

Begriff	Definition
Kryptologie	Die Kryptologie ist die Kunst oder Wissenschaft der Konstruktion und des Angreifens von Verfahren zur Geheimhaltung. Die Teilgebiete der Kryptologie umfassen die Kryptographie, Kryptoanalyse und Authentifizierung [6].
Kryptographie	Kunst oder Wissenschaft der Verschlüsselung [6]
Kryptoanalyse	Kunst oder Wissenschaft des Angreifens [6]
Plaintext P	Klartext
Ciphertext C	Geheimtext
Public Key (PK)	Der öffentliche Schlüssel wird für den Verschlüsselungsprozess verwendet.
Secret Key (SK)	Der geheime Schlüssel wird für den Entschlüsselungsprozess verwendet.
Evaluation Key (EK)	Der Evaluierungsschlüssel wird zur Auswertung auf Ciphertexten ausgeführter Operationen verwendet.
KeyGen	Algorithmus zur Schlüsselgenerierung $\rightarrow$ SK, PK und EK werden mithilfe der vorher definierten Verschlüsselungsparameter erzeugt.
Enc(PK,P)$\rightarrow C$	Verschlüsselungsalgorithmus
Dec(SK,C)$\rightarrow P$	Entschlüsselungsalgorithmus
Rot	Blinde Rotation (siehe Definition 2.5)
$\in$	Element von
$\mathbb{Z}$	Ganze Zahlen $\{\dots, -1, 0, 1, \dots\}$
$\mathbb{N}$	Natürliche Zahlen $\{1, 2, \dots\}$
$\mathbb{N}_0$	Nicht-negative Ganzzahlen $\{0, 1, 2, \dots\}$
$\langle a, b \rangle$	Skalarprodukt oder auch inneres Produkt der Vektoren a und b
mod	Modulo Rest (Division mit Rest)
$\lfloor x \rceil$	x auf die nächste Ganzzahl gerundet
$\mathcal{P}$	Plaintextraum
$\mathcal{C}$	Ciphertextraum
$\mathbf{v}$	Vektor

Tabelle 2.1: Fachbegriffe und Schreibweisen

2.2 Algebraische Grundlagen

Die Kryptografie gilt als Teilgebiet der diskreten Mathematik. Im folgenden wird auf die algebraischen Grundlagen von homomorphen Verschlüsselungen näher eingegangen. Die folgenden Definitionen sind direkte Zitate und die dazugehörige Quelle wird vorher genannt.

Definition 2.1 (Gruppe)
[7, Def. 3.22] Sei G eine Menge mit einer Verknüpfung, die je zwei Elementen $a, b \in G$ ein Element $a \circ b \in G$ zuordnet. Dann wird $(G, \circ)$ eine Gruppe genannt, wenn folgendes gilt:

1. Es gilt $(a \circ b) \circ c = a \circ (b \circ c)$ für alle $a, b, c \in G$ (Assoziativgesetz).
2. Es gibt ein neutrales Element $n \in G$, dass $n \circ a = a \circ n = a$ für alle $a \in G$ erfüllt.
3. Zu jedem $a \in G$ gibt es ein inverses Element $i(a) \in G$, das $a \circ i(a) = i(a) \circ a = n$ erfüllt.
4. Gilt zusätzlich, $a \circ b = b \circ a$ für alle $a, b \in G$ (Kommutativgesetz), so spricht man von einer kommutativen oder abelschen Gruppe.

Definition 2.2 (Ring)
[7, Def. 3.27] Eine Menge R mit zwei Verknüpfungen $+$ und $\cdot$, geschrieben $(R, +, \cdot)$, heißt Ring, wenn folgendes gilt:

1. $(R, +)$ ist eine kommutative Gruppe mit neutralem Element 0.
2. Für alle $a, b, c \in R$ gilt: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ (Assoziativgesetz).
3. Für alle $a, b, c \in R$ gilt: $a \cdot b + a \cdot c = a \cdot (b + c)$ (Distributivgesetz).
4. Gilt zusätzlich das Kommutativgesetz $a \cdot b = b \cdot a$ für alle $a, b \in R$, so spricht man von einem kommutativen Ring
5. Wenn darüber hinaus ein neutrales Element 1 für die Multiplikation existiert, also $a \cdot 1 = 1 \cdot a = a$ für alle $a \in R$, so spricht man von einem kommutativen Ring mit Eins.

Definition 2.3 (Gruppenhomomorphismus)
[8, Def. 2.1] Unter einem Gruppenhomomorphismus ist eine strukturerhaltende Abbildung zwischen zwei oder mehreren algebraischen Strukturen zu verstehen. Die Operanden der ersten Menge bilden Operationen auf die Operanden der zweiten Menge ab. Ein Gruppenhomomorphismus lässt sich belegen, indem die Gruppen $(P, \oplus)$, $(C, \otimes)$ und die Funktion $f : P \to C$ in folgender Gleichung dargestellt werden:

$$\forall a, b \in P: \quad f(a \oplus b) = f(a) \otimes f(b) \in C.$$

Definition 2.4 (Ringhomomorphismus)
[9, S. 1] Es seien $(R, +, \cdot)$ und $(S, \oplus, \otimes)$ zwei Ringe. Die Abbildung $\varphi\colon R \to S$ mit den beiden Eigenschaften

1. $\varphi(a + b) = \varphi(a) \oplus \varphi(b) \quad \forall a, b \in R$
   und
2. $\varphi(a \cdot b) = \varphi(a) \otimes \varphi(b) \quad \forall a, b \in R$
   wird Ringhomomorphismus genannt.

Definition 2.5 (Blindrotation)
Sei C ein Ciphertext, welcher den Plaintextvektor $(p_0, p_1, \ldots, p_{n-1})$ chiffriert (d.h. $\mathrm{Dec}(C) = (p_0, p_1, \ldots, p_{n-1})$, dann ist die Abbildung $\mathrm{Rot}\colon \mathcal{C} \times \mathbb{Z} \to \mathcal{C}$ definiert als $\mathrm{Rot}(C, k) = C'$, wobei C' den Plaintextvektor $(p_{k \bmod n}, p_{k+1 \bmod n}, \ldots, p_{k+n-1 \bmod n})$ chiffriert.

2.3 Entstehung und Abstufung homomorpher Verschlüsselungen

In diesem Abschnitt werden die einzelnen Abstufungen homomorpher Verschlüsselungen anhand ihrer historischen Entstehung vorgestellt.

2.3.1 Partially Homomorphic Encryption

Erste Ideen wie Daten, trotz darauf ausgeführten Operationen, sicher verschlüsselt werden können, formulierten die Informatiker Ronald L. Rivest, Leonard Adleman und Michael L. Dertouzos. In ihrer Veröffentlichung von 1978 legten sie erste Beispiele für „privacy homomorphisms“ (z. Dt. Datenschutzhomomorphismen) dar [10]. Die Datenschutzhomomorphismen stellen den Grundstein für partiell homomorphe Verschlüsselungsverfahren (Partially Homomorphic Encryption - PHE) dar. Partiell homomorphe Verschlüsselungen basieren auf Gruppenhomomorphismen und können somit nur eine bestimmte Operation mehrmals auf Ciphertexten ausführen. Weiterhin gelten sie als die erste und einfachste Form homomorph zu verschlüsseln [11]. Nach der ersten Veröffentlichung 1978 folgten viele weitere partiell homomorphe Verschlüsselungsverfahren. In Tabelle 2.2 wird ein Überblick über die bekanntesten PHE-Veröffentlichungen gegeben.

Kryptosysteme	homomorphe Eigenschaft	Veröffentlichungsjahr
RSA (Rivest, Shamir und Adleman)	Multiplikativ	1978
GM (Goldwasser und Micali)	Additiv	1982
El-Gamal	Multiplikativ	1985
Benaloh	Additiv	1994
Naccache und Stern	Additiv	1998
Okamoto und Uchiyama	Additiv	1998
Paillier	Additiv	1999
Damgård und Jurik	Additiv	2001
KTX (Kawachi, Tanaka und Xagawa)	Additiv	2007

Tabelle 2.2: Partiell homomorphe Verschlüsselungsverfahren [11]

2.3.2 Somewhat Homomorphic Encryption

Die Grundlage von vielen Somewhat Homomorphic Encryption Schemata bilden Ringhomomorphismen. Einen Sonderfall stellt das von Boneh, Goh und Nissim im Jahr 2005 untersuchte Verfahren dar, mit welchem es möglich ist, beliebig oft zu addieren, aber unabhängig von der konkreten Parameterwahl, immer nur eine Multiplikation möglich ist [12].

Ursächlich für die Einschränkung der Anzahl an Operationen in SHE ist, ein kleiner numerischer Fehler (auch *Noise* oder Rauschen genannt) der während des Verschlüsselungsprozesses an den Plaintext angehängt wird. Homomorphe Multiplikationen erzeugen größere Fehler als Additionen. Im Entschlüsselungsprozess wird der Fehler dann wieder korrigiert. Dieser Prozess findet aber nur bei Abweichungen, die klein genug sind, statt. Ist der Fehler zu groß, dann lässt sich der Ciphertext nicht mehr richtig entschlüsseln [13]. Das SHE Konzept kann kurz am Beispiel der Verschlüsselung mit Ganzzahlen dargestellt werden. Folgendes Verfahren wurde der Quelle [14, S.50 ff.] entnommen.

Beispiel der SHE Verschlüsselung mit Ganzzahlen Als erster Schritt im Verschlüsselungsprozess werden die Schlüssel erzeugt. In einem Public Key Verfahren wird ein Public Key und ein Secret Key benötigt. Da dieses Beispiel ein Secret Key Verfahren veranschaulicht, wird lediglich der Secret Key SK generiert:

$$\text{KeyGen}: \quad SK \in [2^{\eta-1}, 2^{\eta}]$$

Um ein Bit $m \in \{1, 0\}$ zu verschlüsseln, wird wie folgt vorgegangen:

$$\text{Enc}(SK, m): \quad c = SKq + 2r + m$$

Hierbei steht c für das Ergebnis der Verschlüsselung, den Ciphertext. Der private Schlüssel $SK \in [2^{\eta-1}, 2^{\eta}]$ ist ein vorher geheim generierter Integer-Wert. Die Platzhalter q und r stehen für zwei zufällig gewählte Integer-Werte, wobei $2r < \frac{SK}{2}$ als Bedingung erfüllt sein muss. Um den erzeugten Ciphertext zu entschlüsseln, werden der private Schlüssel und die Gleichung

$$\text{Dec}(SK, c): \quad m = (c(\text{mod } SK))(\text{mod } 2)$$

benötigt. Es folgt nun ein ausführliches Beispiel zur Verschlüsselung von zwei Plaintexten und der dazugehörige Addition und Multiplikation der zwei Ciphertexte. Gegeben seien die Verschlüsselungen:

$$c_1 = 17 \cdot 1 + 2 \cdot 1 + 0 = 19,$$
$$c_2 = 17 \cdot 2 + 2 \cdot 2 + 1 = 39.$$

Für beide Verschlüsselungen wurde der private Schlüssel $SK = 17$ verwendet. Die Variablen $q_1 = 1$, $q_2 = 2$, $r_1 = 1$ und $r_2 = 2$ wurden zufällig gewählt. Die zu verschlüsselnden Plaintexte sind jeweils ein Bit ($m_1 = 0$ und $m_2 = 1$) groß. Nun werden beide Ciphertexte $c_1 = 19$ und $c_2 = 39$ addiert und das Ergebnis entschlüsselt:

$$c_1 + c_2 = 19 + 39 = 58,$$
$$(58 \,(\text{mod } 17))(\text{mod } 2) = 1 = 0 + 1 = m_1 + m_2.$$

Das Ergebnis der Addition der beiden Ciphertexte stimmt mit dem Ergebnis der Addition der beiden Plaintexte überein. Dies ist ein Beispiel, dass zwei Ciphertexte addiert werden können. Es folgt ein Beispiel, dass zwei Ciphertexte multipliziert werden können:

$$c_1 \cdot c_2 = 19 \cdot 39 = 741,$$
$$(741 \,(\text{mod } 17))(\text{mod } 2) = 0 = 0 \cdot 1 = m_1 \cdot m_2.$$

Weitere Somewhat-Homomorphen Kryptosysteme sind in Tabelle 2.3 aufgelistet.

Kryptosysteme	Besonderheiten	Jahr
Yao	Entwickung des Garbled Circuit Protocol	1982
Poly Cracker (Fellows und Koblitz)	erlaubt sowohl Additionen und Multiplikationen	1994
SYY (Sander, Young und Young)	erste SHE basierend auf Halbgruppen[1]	1999
BGN (Boneh, Goh und Nissim)	beliebig viele Additionen (aber nur eine Multiplikation)	2005
IP (Ishai und Paskin)	binäre Entscheidungsdiagramme werden auf den Verschlüsselungen ausgewertet	2007

Tabelle 2.3: Somewhat homomorphe Verschlüsselungsverfahren [11]

[1] Halbgruppen gelten als Verallgemeinerung von Gruppen. Es gilt das Assoziativgesetz (1. Eigenschaft von Gruppen in Definition 2.1)

2.3.3 Leveled Homomorphic Encryption

Die vorletzte Abstufung bildet die „Leveled“ Homomorphic Encryption (LHE). Sie ähnelt der SHE und unterstützt ebenfalls mehrere Operationen (z.B. Addition und Subtraktion) mit Einschränkungen. Durch im Vorfeld angepasste Parameterwahl können die Einschränkungen aber begrenzt werden. Dadurch ist es möglich, die nötige Anzahl an Operationen auf dem Ciphertext durchzuführen. Die negativen Aspekte der angepassten Parameterwahl sind längere Schlüssel, größere Ciphertexte und längere Laufzeiten [15]. LHE findet vor allem dann Anwendung, wenn die maximale Anzahl der auszuführenden Rechenoperationen bekannt ist und die Parameter effektiv gewählt werden können.

2.3.4 Fully Homomorphic Encryption

Craig Gentry konstruierte im Jahr 2009 den ersten vollständig homomorphen Algorithmus, der es ermöglicht, theoretisch unbegrenzt viele Rechenoperationen auf einem Ciphertext auszuführen. Die Basis dieser Entwicklung bildet ein SHE-Schema. Es ermöglicht das Ausführen von Additionen und Multiplikationen unter der Annahme, dass sich die Noise beim Ausführen von Operationen vergrößert (siehe Abschnitt 2.3.2) Das damit einhergehende Problem des begrenzten Noise-Budgets N und der damit verknüpften begrenzten Anzahl an ausführbaren Rechenoperationen auf einen Ciphertext, löst Gentry durch eine Recrypt-Methode. Die Recrypt-Methode verkleinert die Noise $N' < N$ eines Ciphertextes durch Umschlüsselung so, dass der neuverschlüsselte Ciphertext eine Noise, die um $\sqrt{N}$ kleiner als das Noise-Budget ist, besitzt. Die Recrypt-Methode kann beliebig oft durchgeführt werden und wird auch als Bootstrapping-Methode bezeichnet [16].

Neben dem benannten Vorteil der Durchführung von beliebig vielen Rechenoperationen auf Ciphertexten, ist anzumerken, dass die zusätzliche Umschlüsselung des Ciphertextes zusätzlichen Rechenaufwand benötigt. Aufgrund dessen ist davon auszugehen, dass FHE bei einer begrenzten Anzahl an Rechenoperationen mehr Rechenzeit benötigt als LHE. Vollständig homomorphe Verschlüsselungsverfahren sollten deshalb in einem Kontext verwendet werden, in dem die maximale Anzahl auszuführender Operationen unbekannt ist und LHE deshalb nicht verwendet werden kann. Da in dieser Arbeit hauptsächlich Anwendungsgebiete mit einer im Vorfeld bekannten maximalen Anzahl an Rechenoperationen beschrieben werden, wird im praktischen Teil dieser Arbeit hauptsächlich LHE statt FHE verwendet.

2.4 Learning with Errors (LWE)

Learning with Errors bezeichnet ein mathematisches Problem, welches entsprechend parametrisiert und ohne zusätzliche Information als schwer zu lösen gilt. Die meisten gängigen homomorphen Verschlüsselungsverfahren basieren auf LWE. Die Generierung von LWE-Stichproben lässt sich durch folgende mathematische Gleichung darstellen [17].

$$(\mathbf{a}, c) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q \tag{2.1}$$

Die Variable $\mathbf{s}$ ist ein zufällig gewählter Vektor der Länge n über einem Ring aus Ganzzahlen modulo q ($\mathbb{Z}_q^n$). Die Variable c wird aus dem Skalarprodukt von $\mathbf{a}$, $\mathbf{s}$ und dem Fehler e als Summanden erstellt. Die Variable $\mathbf{a}$ wird zufällig aus $\mathbb{Z}_q^n$ gezogen. Der Fehler e wird auch als Noise bezeichnet und wird aus einer Fehlerverteilung χ gezogen. Ziel ist zu erkennen, ob c wie in Gleichung 2.1 erstellt wurde, oder zufällig in $\mathbb{Z}_q^n$ gewählt wurde [17], [18].

Ein Ciphertext, der durch Learning with Errors erstellt wurde, lässt sich ohne zusätzliche Informationen sehr schwer, aber mit geheimen Schlüssel sehr leicht entschlüsseln. Neuere Schemata nutzen Varianten dieses Problems, wie zum Beispiel Ring-LWE [19].

3 Hinführung zum Benchmarking

Das Ziel dieses Kapitels ist die Vorstellung der im Benchmarking (Abschnitt 4) verglichenen Bibliotheken. Dazu werden die zwei betrachteten Libraries kurz vorgestellt und miteinander verglichen. Danach werden die im Benchmarking getesteten Schemata vorgestellt und ihre Funktion veranschaulicht. Zum Schluss werden die wichtigsten durch die Bibliotheken bereitgestellten Encoding-Verfahren kurz vorgestellt.

3.1 Vorstellung der Bibliotheken

Im folgenden Abschnitt werden die Bibliotheken Microsoft SEAL und die OpenFHE vorgestellt. Bei der Auswahl wurde darauf geachtet, dass beide Bibliotheken auf einem aktuellen Stand sind und regelmäßig aktualisiert werden. Alle betrachteten Libraries sind in der Programmiersprache C++ frei auf Github verfügbar und mit standardmäßigen Toolchains kompilier- und ausführbar.

3.1.1 Microsoft SEAL

„Microsoft SEAL ist eine einfach-benutzbare Open-Source-Bibliothek, welche von der Cryptography and Privacy Group von Microsoft entwickelt wurde [20].“ Neben der C++ Library ist SEAL auch mit anderen Programmiersprachen kompatibel. Mithilfe des Pakets „pybind11“ kann die C++ Bibliothek in Python eingebunden werden [21]. Weiterhin verfügt SEAL über viele Code-Beispiele und eine gute Dokumentation. Es werden die Schemata BGV, BFV und CKKS unterstützt, die Verfahren GSW, FHEW und TFHE sind nicht implementiert. Bootstrapping wird als Option von SEAL nicht zur Verfügung gestellt. Dafür ist aber das Nutzen einer LHE-Option möglich [22]. Die Microsoft SEAL Bibliothek wurde in der Version 4.1.1 verwendet.

3.1.2 OpenFHE

„OpenFHE ist ein Open-Source-Projekt, das effiziente erweiterbare Implementationen von den führenden post-quantum FHE-Schemata bereitstellt [23].“ Es ist der Nachfolger der Palisade-Bibliothek und stellt in der Version 1.0.3 alle Funktionen des Vorgängers zur Verfügung. Im Genaueren betrachtet bedeutet dies, dass OpenFHE BGV, BFV, CKKS, FHEW, TFHE und GSW bereitstellt. Auch die Bootstrapping und Level Option ist in der Bibliothek integriert. Die Dokumentation von OpenFHE ist noch ausbaufähig, deshalb empfiehlt es sich, auf die ausführliche Dokumentation und Beispiele von Palisade zurückzugreifen. Die OpenFHE Bibliothek wurde in dieser Arbeit in der Version 1.0.3 benutzt.

3.1.3 Vergleich der Funktionen von SEAL und OpenFHE

Es folgt ein tabellarischer Vergleich über die Funktionen der beiden Bibliotheken. Dazu wurden die Quellen [20], [24] und [25] verwendet. In Tabelle 3.1 wird die Bereitstellung der bekanntesten Schemata in den Bibliotheken Microsoft SEAL und OpenFHE aufgezeigt. Auffällig ist, dass OpenFHE alle in der Tabelle aufgelisteten Schemata bereitstellt, während Microsoft SEAL keine boolschen Schemata (siehe Abschnitt 3.2) unterstützt. Erwähnenswert ist, dass das BGV Schema ab SEAL Version 4.0 zur Verfügung gestellt wird. Beim Nutzen dieser Bibliothek ist also auf die Version zu achten.

	Bibliotheken	
Schemata	SEAL	OpenFHE
BGV	✓	✓
BFV	✓	✓
CKKS	✓	✓
FHEW	✗	✓
TFHE	✗	✓
GSW	✗	✓

Tabelle 3.1: Vergleich Bereitstellung Schemata

3.2 Vorstellung der Schemata

Homomorphe Schemata lassen sich anhand ihrer Arithmetik in mehrere Klassen unterteilen. Die bekanntesten drei Klassen, die dargestellt werden können, sind „boolean circuit“, „modular integer“ und „approximate reals“. Bei „boolean circuit“ (z. Dt. Boolesche Schaltkreise) werden Plaintexte als eine Darstellung von Bits betrachtet. Ein Boolscher Schaltkreis ist eine Menge logischen Operationen wie z.B. XOR und AND, mithilfe dieser eine beliebige Funktion konstruiert werden kann. Bekannte Vertreter dieser Klasse sind „Fastest Homomorphic Encryption in the West“ (FHEW) und „Fast Fully Homomorphic Encryption over the Torus“(TFHE) [19]. Homomorphe Verschlüsselungsschemata, die ganzzahlige Plaintexte chiffrieren und mit Modulo-Operationen encodieren, werden auch als „modular integer“ bezeichnet. Die bekanntesten Vertreter dieser Klasse sind das BGV- und das BFV-Schema. Die Klasse der „approximate reals“ stellt reelle Zahlenwerte bis auf einen kleinen Fehler dar. Dadurch ist es möglich, zu verschlüsselnde reelle Zahlen bis zu einer bestimmten Nachkommastelle abzubilden. Anwendung findet diese Klasse beispielsweise im CKKS-Schema [19]. Im Folgenden werden das BFV- und das CKKS-Schema genauer betrachtet.

3.2.1 Brakerski-Fan-Vercauteren (BFV)

Das BFV Schema wurde von Zvika Brakerski, Junfeng Fan und Frederik Vercauteren entwickelt und 2012 veröffentlicht. Es nutzt die in Abschnitt 2.2 beschriebenen Homomorphismen als mathematische Grundlage, um Plaintextnachrichten (M) verschlüsseln zu können. Dadurch ist es möglich, Operationen auf einem Ciphertext durchzuführen, ohne den für den Entschlüsselungsprozess notwendigen Secret Key zu besitzen. Die Sicherheitsannahme von BFV basiert auf dem mathematischen „Ring-Learning with Errors“ (RLWE)-Problem, einer Variante des

schon in Abschnitt 2.4 beschriebenen LWE-Problems [26]. Im BFV-Schema wird der Plaintextraum als Ring $\mathcal{P}$ bezeichnet. Dieser Ring enthält den Klartext in lesbarer und in encodeter (siehe Abschnitt 3.3 für Encoding) Form und wird als Polynomring $\mathcal{P} = R_t = \mathbb{Z}_t[x]/(x^n + 1)$ definiert [27, S.5]. Der zweite Ring ist der Ciphertextraum $\mathcal{C}$, welcher den Geheimtext enthält. Er ist ebenfalls ein Polynomring und wird als $\mathcal{C} = R_q \times R_q$ definiert, wobei $R_q = \mathbb{Z}_q[x]/(x^n+1)$ ist. Die Variable $n \in \mathbb{Z}$ bezeichnet dabei die Ringdimension. Die Ringdimension nimmt normalerweise den Wert einer Zweierpotenz an. Die Variable q stellt den Ciphertext-Koeffizienten dar, während t für den Koeffizienten des Plaintextes steht. Für beide Koeffizienten gilt $q, t \in \mathbb{Z}$. Der Ciphertext-Koeffizient und damit auch die Mächtigkeit des Ciphertextraumes $\mathcal{C}$ ist größer als der Plaintext-Koeffizient und dessen Plaintextraum $\mathcal{P}$. Dadurch ist es möglich, dass ein Plaintext auf mehrere Ciphertexte abgebildet werden kann [13],[28]. Zusätzlich zu den vorgestellten Variablen wird die Zufallsverteilung $\mathcal{X}$ im BFV-Schema verwendet. Aus ihr können für den Verschlüsselungsprozess notwendige Fehlerpolynome gezogen werden können [13].

Die für die Ver- und Entschlüsselung notwendigen Secret Key (SK) und Public Key (PK), werden während der Schlüsselgenerierung (KeyGen) im BFV-Schema erzeugt. Der SK wird dabei aus einer aus $\mathcal{X}$ gezogenen Stichprobe $\mathbf{s}$ generiert [13, S.4]. Der PK, wird aus den zwei Teilschlüsseln PK_1 und PK_2 zusammengesetzt. Die Gleichung zur Erstellung der beiden Teilschlüssel wurde aus dem Blog [28] unter Berücksichtigung der Originalquelle [13] entnommen:

$$\begin{aligned} \mathrm{PK}_1 &= [-1(\mathbf{a} \cdot \mathrm{SK} + \mathbf{e})]_q \\ \mathrm{PK}_2 &= \mathbf{a} \end{aligned}$$

Die Variable $\mathbf{e}$ steht für ein zufälliges Fehlerpolynom, das mithilfe der Fehlerverteilung $\mathcal{X}$ erzeugt wurde, und die Variable $\mathbf{a}$ ist eine aus R_q gezogene Stichprobe [13, S.4].

Um eine Klartextnachricht M im Plaintextraum $\mathcal{P}$ zu verschlüsseln, sind zwei Teilchiffren C_1 und C_2 notwendig. Die Gleichungen zum Generieren von C(C_1,C_2) wurden unter Berücksichtigung der Originalquelle [13] aus dem Blog [28] entnommen:

$$\begin{aligned} \mathrm{C}_1 &= [\mathrm{PK}_1 \cdot \mathbf{u} + \mathbf{e}_1 + \Delta \mathrm{M}]_q \\ \mathrm{C}_2 &= [\mathrm{PK}_2 \cdot \mathbf{u} + \mathbf{e}_2]_q \end{aligned} \tag{3.1}$$

Zuerst werden zufällige Polynome $\mathbf{u}$ aus Verteilung $\mathcal{X}$ gezogen. Weiterhin werden zwei Fehlerpolynome $\mathbf{e}_1$ und $\mathbf{e}_2$ aus $\mathcal{X}$ generiert. Die Variable Δ soll die Größe der zu verschlüsselnden Nachricht M skalieren, indem sie q durch t teilt [13, S.4]. Um einen verschlüsselten Text in eine lesbare Nachricht zu entschlüsseln, ist folgende Gleichung notwendig [28]:

$$\mathrm{M} = \left[\left\lfloor \frac{t[\mathrm{C}_1 + \mathrm{C}_2 \cdot \mathrm{SK}]_q}{q} \right\rceil \right]_t$$

Um so kleiner die Variablen $\mathbf{u}$ und $\mathbf{s}\rightarrow$SK gewählt werden, desto kleiner ist die Noise $\mathbf{v} = \mathbf{e} \cdot \mathbf{u} + \mathbf{e}_1 + \mathbf{e}_2 \cdot \mathbf{s}$ [13, S.5].

Die für das Benchmarking interessanten Operationen Addition und Multiplikation werden im Folgenden dargestellt.
Die Addition von zwei Ciphertexten wird als

$$\begin{aligned}\text{EvalAdd}(\mathrm{C}^{(1)}, \mathrm{C}^{(2)}) &= ([\mathrm{C}_1^{(1)} + \mathrm{C}_1^{(2)}]_q, [\mathrm{C}_2^{(1)} + \mathrm{C}_2^{(2)}]_q) \\ &= (\mathrm{C}_1^{(3)} + \mathrm{C}_2^{(3)}) \\ &= \mathrm{C}^{(3)}\end{aligned} \tag{3.2}$$

dargestellt [13],[28].
Der in Gleichung 3.1 aus zwei Teilen bestehende Ciphertext wird, für die Bestimmung der Noise, als ein ganzer Ciphertext $[\mathrm{C}(s)]_q = \Delta \cdot \mathrm{M} + \mathbf{v}$ betrachtet. Durch die Addition der beiden Ciphertexte

$$\text{EvalAdd} = \Delta \cdot [\mathrm{M}_1 + \mathrm{M}_2]_t + \mathbf{v}_1 + \mathbf{v}_2 - \epsilon \cdot t \cdot \mathbf{r}$$

wird die Noise $\mathbf{v}$ der beiden Ciphertexte addiert. Der durch Subtraktion angefügte Term $\epsilon \cdot t \cdot \mathbf{r}$ wird in Quelle [13, S.6] erklärt und beweist, dass das Noise-Wachstum additiv um das Maximum des Plaintextkoeffizienten t erfolgt.

Die Multiplikation von zwei Ciphertexten wird als

$$\begin{aligned}\text{EvalMult}(\mathrm{C}^{(1)}, \mathrm{C}^{(2)}) = \Bigg(&\left[\left\lfloor \frac{t(\mathrm{C}_1^{(1)} \cdot \mathrm{C}_1^{(2)})}{q} \right\rceil\right]_q, \left[\left\lfloor \frac{t(\mathrm{C}_1^{(1)} \cdot \mathrm{C}_2^{(2)} + \mathrm{C}_2^{(1)} \cdot \mathrm{C}_1^{(2)})}{q} \right\rceil\right]_q, \\ &\left[\left\lfloor \frac{t(\mathrm{C}_2^{(1)} \cdot \mathrm{C}_2^{(2)})}{q} \right\rceil\right]_q \Bigg)\end{aligned} \tag{3.3}$$

dargestellt. Das Ausführen einer Multiplikation auf Ciphertexte vergrößert die Anzahl der Terme des resultierenden Ciphertextes von 2 auf 3. Um den Ciphertext wieder auf zwei Terme zu reduzieren und dadurch entschlüsseln zu können, muss Relinearisierung verwendet werden. Relinearisierung formt den durch Multiplikation entstandenen Term in $R_q \times R_q \times R_q$ in einen zugehörigen Ciphertext aus $\mathcal{C} = R_q \times R_q$ um. Die Noise vergrößert sich bei Multiplikationen um den Faktor $2 \cdot t \cdot n^2 \cdot ||\mathrm{SK}||$. Die Gleichung 3.3 und Inhalt zur Multiplikation wurden unter Berücksichtigung der Originalquellen [13] und [26] vom Blogeintrag [28] entnommen.

3.2.2 Cheon-Kim-Kim-Song (CKKS)

Das CKKS-Schema wurde 2017 auf der Asiacrypt 2017 von J.H. Cheon, A. Kim, M. Kim, Y. Song veröffentlicht. Es wird auch als HEAN (Homomorphic Encryption for Arithmetic of Approximate Numbers) bezeichnet. Mithilfe dieses Schemas ist es möglich homomorphe Operationen (wie z.B. Additionen und Multiplikationen) auf reellen Zahlen auszuführen. Dabei werden reelle Zahlen so weit wie möglich approximiert, das heißt, bis zu einer vorher festgelegten Nachkommastelle abgebildet [29].

Der Hauptunterschied zwischen dem im Abschnitt 3.2.1 vorgestellten BFV-Schema und dem CKKS-Schema besteht in der Anordnung von Plaintextspace t und der Noise v. Während das BFV-Schema beide Variablen getrennt voneinander betrachtet, wird im CKKS-Schema die

Noise zu signifikanten Teilen des Plaintextes hinzugefügt. Die Noise wird aber trotzdem als ein Teil des Fehlers betrachtet, welcher zusammen mit dem Plaintext durch das Rescaling-Verfahren reduziert werden kann [29].
Das CKKS-Schema nutzt, wie auch das BFV-Schema, das RLWE-Problem, was zur Folge hat, dass Plaintextspace t und Ciphertextspace q analog zu den in Abschnitt 3.2.1 definierten Ringen dargestellt werden. CKKS wird im Folgenden als LHE-Schema betrachtet, was zur Folge hat, dass q_l den Koeffizienten Modulus in dem entsprechenden Level angibt [29].

Die KeyGen des CKKS-Schemas ist analog zur Schlüsselgenerierung des BFV-Schemas zu betrachten. Der einzige Unterschied besteht in der Generierung des PK. Hier wird in PK_1 die Äquivalenzklasse von q_l statt von q betrachtet:

$$\mathrm{PK}_1 = [-\mathbf{a} \cdot \mathrm{SK} + \mathbf{e})]_{q_l}$$

Der zweite Teilschlüssel PK_2 entspricht dem im Abschnitt 3.2.1 definierten Teilschlüssel. Für die Schlüsselgeneration wurde der Blogbeitrag [30] unter Verwendung der Originalquelle [29] verwendet.

Auch der Ver- und Entschlüsselungsprozess im CKKS-Schema ähnelt dem gleichnamigen Prozess im BFV-Schema. Der Verschlüsselungsprozess im CKKS-Schema besteht ebenfalls aus zwei Teilchiffren C_1 und C_2. Im Folgenden wird der Verschlüsselungsprozess aus dem Blogbeitrag [30] unter Berücksichtigung der Originalquelle [29] dargestellt:

$$\mathrm{C}_1 = [\mathrm{PK}_1 \cdot \mathbf{u} + \mathbf{e}_1 + M]_{q_l}$$
$$\mathrm{C}_2 = [\mathrm{PK}_2 \cdot \mathbf{u} + \mathbf{e}_2]_{q_l}$$

Die Entschlüsselung des Ciphertext C mit dem Level l lässt sich durch

$$\mathrm{M}' = [(\mathrm{C}, \mathrm{SK})]_{q_l}$$

darstellen [29, S.9].

Die Addition im CKKS-Schema erfolgt analog zur Addition im BFV-Schema (Gleichung 3.2). Statt der Äquivalenzklasse von q wird in der Addition von CKKS q_l betrachtet. Außerdem müssen alle Summanden die selbe Skalierung besitzen [29]. Eine Analogie dazu kann zur Addition von Brüchen durch

$$\frac{a}{\Delta} + \frac{c}{\Delta} = \frac{a+c}{\Delta}$$

hergestellt werden. Die Zähler können nur addiert werden, wenn die Nenner gleich sind.

Die Multiplikation von Ciphertexten im CKKS-Schema lässt sich vereinfacht als

$$\mathrm{EvalMult}((\mathrm{C}_1, l, \Delta), (\mathrm{C}_2, l, \Delta)) \rightarrow (\mathrm{C}_3, l, \Delta^2)$$

abbilden [31]. Da das Ergebnis der Multiplikation analog zur Multiplikation im BFV-Schema aus einem Term mit drei Elementen besteht, der nicht die Bedingung des Ciphertextraum $\mathcal{C}$ erfüllt, muss nach der Multiplikation relinearisiert werden. Die Relinearisierung lässt sich durch

$$(\mathrm{C}_3, l, \Delta^2) \rightarrow (\mathrm{C}'_3, l, \Delta^2)$$

darstellen [31]. Da sich bei der Multiplikation von zwei Ciphertexten die Skalierung Δ zu Δ^2 quadriert, muss auf den nach der Relinearisierung entstandenen Ciphertext das Rescaling-Verfahren angewendet werden. Das Rescaling-Verfahren lässt sich durch

$$(\mathrm{C}'_3, l, \Delta^2) \rightarrow (\mathrm{C}'_3, l, \Delta)$$

beschreiben [31]. Durch das Rescaling-Verfahren wird die Skalierung Δ des entstandenen Ciphertext C'_3 wieder auf die ursprüngliche Skalierung reduziert. Weitere Informationen zum CKKS-Schema und des darin bereitgestellten Rescaling-Verfahrens können in den Quellen [29] und [31] gewonnen werden.

3.3 Encoding-Techniken

Einer der wichtigste Prozesse einer homomorphen Verschlüsselung ist der Encoding-Prozess. Das Encoding (z.Dt. die Kodierung) ist für das Umwandeln von Eingaben (z.B reelle Zahlen oder Ganzzahlen) in die für die Verschlüsselung notwendigen Elemente des Plaintextspaces $\mathcal{P} = \mathrm{R}_t$ verantwortlich. Meistens wird $\mathbb{Z}$ als hinreichend große, endliche Menge betrachtet und versucht diese Menge injektiv[2] auf R_t abzubilden [32]. Das Decoding (z.Dt. Dekodierung) ist die Umkehroperation des Encodings. Beide Operationen werden durch den Encoder ausgeführt.

Es gibt unterschiedliche Encoder für verschiedene Aufgaben. Im Folgenden werden verschiedene Encoding-Verfahren vorgestellt und es wird aufgelistet, welche Encoder in der Microsoft SEAL Bibliothek und in der OpenFHE Bibliothek bereitgestellt werden.

Der einfachste Encoder ist der Skalar Encoder. Er kann Ganzzahlen kodieren. Die Ganzzahl a wird dabei „als das konstantes Polynom $a \in \mathrm{R}_t$ [32, Scalar Encoder]“ betrachtet. Der Encoder ist nicht in der Lage, Modulo-Arithmetik auszuführen, was zur Folge hat, dass t möglichst groß gewählt werden muss und dadurch Probleme mit dem Noise-Wachstum entstehen [32]. Der Skalar Encoder wird in den neueren Versionen von Microsoft SEAL und OpenFHE nicht mehr bereitgestellt.

Ein im Vergleich zum Skalar Encoder deutlich effizienterer Encoder ist der Integer Encoder. Der Integer Encoder ist in der Microsoft SEAL Bibliothek implementiert und die genaue Funktionsweise ist unter [32, Integer Encoder] zu finden.

[2] „f heißt injektiv, wenn verschiedene Elemente von D auf verschiedene Elemente von $f(\mathrm{D})$ abgebildet werden, kurz: $x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)$ für alle $x_1, x_2 \in \mathrm{D}$ [7].“

Die OpenFHE Bibliothek stellt keinen Integer Encoder zur Verfügung, dafür wird aber aber ein String-Encoding bereitgestellt. Mit String-Encoding ist es möglich, Zeichenketten in Polynome zu kodieren und anschließend zu verschlüsseln. Nachteil dieser Encoding-Variante ist aber, dass keine homomorphen Operationen mit String-Encoding möglich sind [33].

Um rationale Zahlen $\mathbb{Q}$ kodieren zu können, kann der Fractional Encoder in der Microsoft SEAL Bibliothek verwendet werden. Dieser Encoder skaliert alle rationalen Zahlen in Ganzzahlen. Von da aus können die Ganzzahlen dann mit dem Integer Encoder kodiert werden. Fractional Encoding wurde auch in der Palisade Bibliothek, dem Vorgänger der OpenFHE Bibliothek bereitgestellt, wird aber nicht in der Dokumentation der OpenFHE Bibliothek erwähnt [34].

Das (für das BFV-Schema) im Benchmarking (Kapitel 4) und im Usecase (Kapitel 5) verwendete Encoding ist das Packed-Encoding. Es zählt zu den leistungsstärksten Encodingverfahren. Es wird in der SEAL durch das CRT Batching und den Batch-Encoder bereitgestellt. Dieser ermöglicht es, mehrere Ganzzahlen n mod t in ein Klartextpolynom einzufügen. Dadurch kann auf mehreren Daten gleichzeitig eine Anweisung ausgeführt werden. Diese Methode wird auch als SIMD (Single Instruction, Multiple Data) bezeichnet. Weiterhin ermöglicht der Batch-Encoder die Durchführung von Rotationen (Definition 2.5) [32]. Die OpenFHE Bibliothek stellt ebenfalls das Packed-Encoding mit SIMD und Rotation bereit. Außer einer kurzen Vorstellung des Packed-Encodings unter Palisade [33], wurde keine ausführliche Dokumentation des Packed-Encodings unter OpenFHE gefunden. Deswegen ist ein Vergleich zwischen Batch-Encoder (Microsoft SEAL) und Packed-Encoding (OpenFHE) derzeit nicht möglich.

Um auch im CKKS-Schema kodieren zu können, muss eine angepasste Form des Packed-Encodings verwendet werden. In OpenFHE ist das CKKS-Packed-Encoding implementiert, welches auch im Benchmarking der OpenFHE Bibliothek verwendet wird. Das CKKS-Packed-Encoding unterscheidet sich zu dem Packed-Encoding nur im encodierten Zahlentyp. Während Packed-Encoding Ganzzahlen kodiert, werden beim CKKS-Packed-Encoding reelle Zahlen näherungsweise kodiert [33].

Auch in der Microsoft SEAL Bibliothek wird ein CKKS-Encoder bereitgestellt. Da dieser die selben Funktionen wie der CKKS-Packed-Encoder ausführen kann und keine ausführliche Dokumentation zu dem Encoder gefunden wurde, kann nicht näher auf die Funktionsweise des CKKS-Encoder eingegangen werden.

Erwähnenswert ist, dass auf der Eurocrypt 2022 FIMD (Field Instruction Multiple Data) eine Erweiterung zu dem SIMD vorgestellt wurde. Die Vorteile der FIMD-Methode werden in dem Konferenz-Paper [35] beschrieben.

4 Benchmarking

In diesem Kapitel werden Benchmarks für die Microsoft SEAL Bibliothek und die OpenFHE Bibliothek durchgeführt. Es wurden jeweils zwei Benchmarking-Tests pro Library für das BFV-Schema und das CKKS-Schema durchgeführt. Betrachtet werden die Addition und die Multiplikation von Ciphertexten. Zuerst wird der Aufbau des Versuches beschrieben. Dabei wird die verwendete Hardware aufgelistet, die Parameterwahl beschrieben und die Vergleichbarkeit thematisiert. In Abschnitt 4.2 werden schlussendlich die Benchmarking-Resultate der Bibliotheken ausgewertet und miteinander verglichen.

4.1 Versuchsbeschreibung

Dieser Abschnitt thematisiert zuerst die in der praktischen Arbeit verwendete Hardware. Anschließend wird die getroffene Parameterauswahl und der dazugehörige Versuchsaufbau für Bibliotheken und Schemata erläutert. Zum Schluss wird auf die Vergleichbarkeit der Benchmarks eingegangen.

4.1.1 Hardware Spezifikationen

Alle Benchmarks und Anwendungsbeispiele (aus Kapitel 5) wurden auf einem Surface Laptop 4 durchgeführt. Der Laptop wurde von der Agentur für Innovation in der Cybersicherheit für die Bachelorarbeit bereitgestellt und besitzt folgende Hardware:

Als Prozessor wurde der Intel Core i7-1185G7 verbaut. Er besitzt vier Kerne und eine Basistaktfrequenz von 3 GHz. Der im Laptop verbaute Arbeitsspeicher ist 16 GB DDR4 RAM mit einer Taktfrequenz mit 4267 MHz. Im Surface Laptop 4 ist außerdem eine Grafikeinheit verbaut. Als Grafikkarte besitzt der Laptop eine Intel Iris XE Grafikkarte.

Softwaretechnisch wurde der Laptop mit Windows 11 Pro ausgestattet. Die Benchmarks und Anwendungsbeispiele wurden aber auf einem Windows Subsystem für Linux (WSL2) in Ubuntu ausgeführt. Die Hardwareauslastung wurde dabei vom System automatisch festgelegt und je nach Bedarf angepasst. Wichtig anzumerken ist, dass WSL2 noch nicht in der Lage ist die vollständige Leistung einer nativen Linux-Distribution zu erbringen. Weiterhin ist zu erwähnen, dass es während der Messungen zu Hintergrundprozessen in Windows gekommen ist und diese die Messwerte beeinflussen können.

4.1.2 Parameterwahl und Versuchsaufbau

Im folgenden wird auf die Parameterwahl der einzelnen Versuchsaufbauten eingegangen. Grundsätzlich wurde versucht, optimale und vergleichbare Parameter für das Benchmarking zu wählen. Alle im Benchmarking gesetzten Parameter werden in Tabelle 4.1 dargestellt. Im Folgenden werden die Parameter erklärt und danach werden die einzelnen Versuchsaufbauten beschrieben.

	Parameter				
Versuch	n	q	p	Security-Level	Encoder
1.Versuch SEAL	8192	218	536 690 689	128	Batch-Encoder
1.Versuch OpenFHE	8192	120	536 690 689	128	Packed-Encoder
2.Versuch SEAL	32 768	881	536 608 769	128	Batch-Encoder
2.Versuch OpenFHE	32 768	540	536 608 769	128	Packed-Encoder
3.Versuch SEAL	8192	200	-	128	CKKS-Encoder
3.Versuch OpenFHE	8192	110	-	128	CKKS-Packed-Encoder
4.Versuch SEAL	32 768	560	-	128	CKKS-Encoder
4.Versuch OpenFHE	32 768	560	-	128	CKKS-Packed-Encoder

Tabelle 4.1: Auflistung der Parameter

Erklärung der Parameter aus Tabelle 4.1

Im Folgenden werden die in Tabelle 4.1 aufgelisteten Parameter erklärt. Der Parameter n steht für den Poly-Modulus-Degree. Der Poly-Modulus-Degree ist eine positive Zweierpotenz und stellt den Grad eines zyklotomischen Polynoms dar. Umso größer der Poly-Modulus-Degree, desto größer wird der Ciphertext. Die Größe des Ciphertextes beeinflusst die Laufzeit und die Komplexität der Berechnungen. Ein großer Poly-Modulus-Degree steht für eine längere Laufzeit und ermöglicht komplexere Operationen als ein kleiner Poly-Modulus-Degree. Mögliche Werte für n sind 1024, 2048, 4096, 8192, 16 384, sowie 32 768. Der Parameter q steht für die Koeffizienten-Modulus-Bitlänge. Der Koeffizienten-Modulus ist eine große Ganzzahl und ein Produkt aus mehreren Primzahlen. Der Koeffizienten-Modulus gibt die Größe des Noise-Budgets an. Umso größer der Koeffizienten-Modulus, desto größer das Noise-Budget und die damit verbundene Anzahl an Operationen, die ausgeführt werden können. Der Parameter p steht für den Plaintext Modulus. Im CKKS-Schema gibt es keinen Plaintext Modulus, sondern das Noisbudget ist von der Skalierung des Ciphertext abhängig. Der Skalierungsfaktor wurde bei beiden Bibliotheken auf 40 bit gesetzt. Das Security Level ist bei allen Versuchen standardmäßig auf 128 Bit gesetzt und gewährleistet die Sicherheit der Verschlüsselung. Das Encoding ist je nach Bibliothek und Schema unterschiedlich. Die unterschiedlichen Arten von Encodern werden in Abschnitt 3.3 vorgestellt.

Versuch 1: Addition mit BFV

Im ersten Benchmarking Test wurde der Zeitaufwand der Microsoft SEAL Bibliothek und der OpenFHE Bibliothek bei der Ausführung von Additionen im BFV-Schema ermittelt. Die Testung wurde für jede Bibliothek separat durchgeführt. Es wurden jeweils 21 Messungen vorgenommen, die um Messfehler zu verhindern insgesamt 19 mal wiederholt werden. Die jeweilige Anzahl an Additionen pro Messung wird in Tabelle 4.2 dargestellt. Jede Messung besteht aus drei Zeitmessungen. Gemessen werden die Verschlüsselungszeit, die Entschlüsselungszeit und die Zeit, die für die jeweilige Anzahl an Additionen benötigt wird.

Nummer der Messung	Anzahl der Additionen
0	100
1	200
2	400
3	600
4	800
5	1000
6	2000
7	4000
8	6000
9	8000
10	10 000
11	20 000
12	40 000
13	60 000
14	80 000
15	100 000
16	200 000
17	400 000
18	600 000
19	800 000
20	1 000 000

Tabelle 4.2: Auflistung Additionen pro Messung

Versuch 2: Multiplikation mit BFV

In der zweiten Benchmarking Testreihe wurde der Zeitaufwand beider Bibliotheken bei der Ausführung von Multiplikationen im BFV-Schema getestet. Jede Bibliothek wurde separat getestet. Für jede Bibliothek wurden 10 Messungen durchgeführt und diese 99 mal wiederholt. Die Anzahl der Multiplikationen pro Messung wird in Tabelle 4.3 dargestellt. Jede Messung besteht aus drei Zeitmessungen. Gemessen werden die Verschlüsselungszeit, die Entschlüsselungszeit und die Zeit, die für die jeweilige Anzahl an Multiplikationen benötigt wird.

Nummer der Messung	0	1	2	3	4	5	6	7	8	9
Anzahl der Multiplikationen	1	2	3	4	5	6	7	8	9	10

Tabelle 4.3: Auflistung Multiplikationen pro Messung

Versuch 3: Addition mit CKKS

Im dritten Benchmarking Test wurden Additionen im CKKS Schema ausgeführt und die Dauer der Ausführung gemessen. Es wurden, wie in Versuch 1, jeweils 21 Messungen vorgenommen, die insgesamt 19 mal wiederholt wurden, um Messfehler zu verhindern. Die jeweilige Anzahl an Additionen pro Messung ist gleich geblieben und wird in Tabelle 4.2 dargestellt. Jede Messung besteht aus drei Zeitmessungen. Gemessen werden die Verschlüsselungszeit, die Entschlüsselungszeit und die Zeit, die für die jeweilige Anzahl an Additionen benötigt wird.

Versuch 4: Multiplikation mit CKKS

Im vierten Benchmarking Test wird der Zeitaufwand der Microsoft SEAL Bibliothek und der OpenFHE Bibliothek bei der Ausführung von Multiplikationen im CKKS-Schema ermittelt. Dafür wurden (wie in Versuch 2) für jede Bibliothek 10 Messungen durchgeführt und diese 99 mal wiederholt. Die Anzahl der Multiplikationen pro Messung werden in Tabelle 4.3 dargestellt. Jede Messung besteht aus drei Zeitmessungen. Gemessen werden die Verschlüsselungszeit, die Entschlüsselungszeit und die Zeit, die für die jeweilige Anzahl an Multiplikationen benötigt wird.

4.1.3 Vergleichbarkeit

Um eine Vergleichbarkeit zwischen beiden Bibliotheken zu gewährleisten, wurde im Benchmarking Prozess darauf geachtet, möglichst ähnliche Parameter in den jeweiligen Messungen zu wählen. Die hier im Benchmarking gewählten Parameter sind das Resultat mehrerer Testläufe mit unterschiedlichen Parametern.

Um eine Vergleichbarkeit zwischen den Schemata zu gewährleisten, wurde in den Versuchen die selbe Anzahl an Messungen und Operationen ausgeführt. Des Weiteren wurden gleiche Poly-Modulus-Degree Werte gewählt.

Ein für die Vergleichbarkeit entscheidender Punkt ist das Security-Level. Das Security Level gewährleistet die Sicherheit der Verschlüsselung und ist in allen Testläufen standardmäßig auf 128 Bit gesetzt. In Quelle [36] wurden Attacken auf homomorphe Verschlüsselungen mit klassischen Algorithmen und Quantenalgorithmen berechnet. Angriffe auf Verschlüsselungen mit Security-Level 128 benötigen immer mindestens 2^{128} Versuche, egal welcher von den beschriebenen Angriffen durchgeführt wird. Damit gelten sie als sicher gegen klassische Angriffe, aber auch gegen Angriffe mithilfe von Quantencomputern.

4.2 Ergebnisvorstellung

Im Folgenden werden die Ergebnisse des Benchmarkingprozesses vorgestellt. Die aus dem Benchmark resultierenden Testreihen wurden exportiert und mit dem Paket Matplotlib in Python in Form von Log-Log-Diagrammen[3] ausgewertet. In den Diagrammen wird die Latenz in Clockcycles auf der y-Achse und die Anzahl der ausgeführten Operationen auf der x-Achse dargestellt. Die mit der Microsoft SEAL erzielten Ergebnisse werden durch orange, mit einer Linie verbundene Punkte dargestellt. Im Gegensatz dazu werden die Ergebnisse der OpenFHE Bibliothek durch blaue, durch eine Linie verbundene Punkte repräsentiert. Für beide Bibliotheken wurden Regressiongeraden eingezeichnet, die den Anstieg des jeweiligen Graphen veranschaulichen sollen. Für die Microsoft SEAL Bibliothek wird eine grüne Regressionsgerade gezeichnet, während die Regressionsgerade für OpenFHE rot ist.

[3] In einem Log-Log-Diagramm erfolgt die Achsenbeschriftung beider Achsen mit dem Logarithmus des jeweiligen Zahlenwertes.

Nach der Vorstellung aller Ergebnisse wird ein Überblick über die Latenz beider Bibliotheken in Form eines Säulendiagrammes gegeben. Hierbei werden neben den Operationen auch die Entschlüsselungs- und Verschlüsselungsdauer betrachtet. Anschließend folgt eine Schlussfolgerung.

4.2.1 Versuch 1: Addition BFV

In Versuch 1 wurde die Abhängigkeit der Latenz bei der Ausführung von Additionen im BFV-Schema untersucht. Dabei wurde die Zeitmessung der Additionen, wie unter 4.1.2 beschrieben, durchgeführt. Um Messfehler zu umgehen, wurde aus den aufgezeichneten 20 Testreihen das Minimum und das Maximum für jeden Messpunkt bestimmt und anschließend entfernt. Die verbleibenden Messwerte werden für jeden Messpunkt gemittelt und in den Graphen SEAL und OpenFHE repräsentiert (Abbildung 4.1). Auffällig ist, dass die Messwerte der Microsoft SEAL Bibliothek einen deutlich kleineren Abstand zur x-Achse besitzen als die Messwerte der OpenFHE Bibliothek. Daraus ist zu schlussfolgern, dass die Microsoft SEAL Bibliothek schneller im BFV-Schema Additionen ausführt als die OpenFHE Bibliothek. Auffällig ist weiterhin, dass beide Graphen einen Anstieg von 1,0 haben. Im Log-Log-Diagramm bedeutet dies, dass ein linearer Anstieg vorliegt.

Die hellblau gekennzeichnete Fläche steht für die Messabweichung der SEAL Bibliothek und die hell-orange Fläche gibt die Messabweichung der OpenFHE Bibliothek an. Gemessen wird die Messabweichung am Minimum und Maximum jedes Messpunktes. In der Abbildung fällt auf, dass bei den Messungen der Microsoft SEAL Bibliothek wenige Messabweisungen erkennbar sind. Dies spricht für eine sehr stabile Implementation des Schemas durch die Bibliothek. Bei der OpenFHE Messung sind mehr Messabweichungen erkennbar, was als Nachteil gegenüber SEAL anzusehen ist. Aus den beschriebenen Benchmarking-Resultaten ist zu schlussfolgern, dass die Microsoft SEAL Bibliothek schneller Additionen auf ganzzahligen Ciphertexten ausführen kann als die OpenFHE Bibliothek.

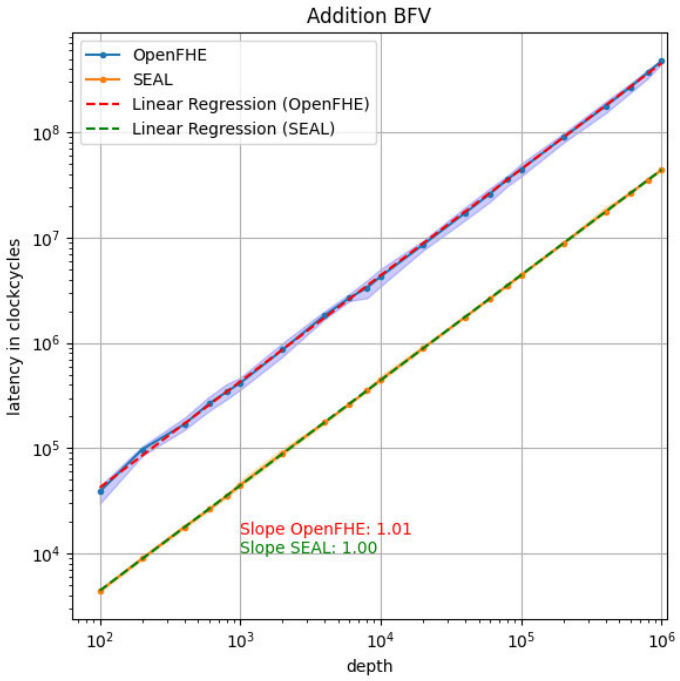

Abbildung 4.1: Laufzeiten Addition BFV

4.2.2 Versuch 2: Multiplikation BFV

In Versuch 2 wurde die Abhängigkeit der Latenz bei der Ausführung von Multiplikationen im BFV-Schema untersucht. Die Zeitmessung der durchzuführenden Multiplikationen erfolgt dabei, wie in Abschnitt 4.1.2 beschrieben. Messfehler, die in insgesamt 100 aufgezeichneten Testreihen entstanden sind, werden als Minimum- und Maximum-Werte betrachtet, detektiert und entfernt. Die verbleibenden Messwerte werden für jeden Messpunkt gemittelt und der Mittelwert wird in den Graphen SEAL und OpenFHE repräsentiert (Abbildung 4.2). Die Messwerte der Microsoft SEAL Bibliothek sind deutlich kleiner als die Messwerte der OpenFHE Bibliothek. Das bedeutet, dass die SEAL Bibliothek auch bei der Ausführung von Multiplikationen schneller als die OpenFHE Bibliothek ist. Der Anstieg beider Bibliotheken liegt bei 0,99. Daraus lässt sich schlussfolgern, dass auch bei der Multiplikation ein linearer Anstieg vorliegt. Die hellblaue und hell-orange Messabweichung ist bei der Multiplikation bei beiden Bibliotheken deutlich höher als bei Versuch 1 (siehe Abschnitt 4.2.1). Der Grund dafür könnte die vergleichsweise höhere Anzahl an aufgezeichneten Testreihen sein. Die Messabweichung der Microsoft SEAL Bibliothek ist aber auch in Versuch deutlich geringer als bei OpenFHE, weshalb geschlussfolgert werden kann, dass die Microsoft SEAL Bibliothek auch beim Durchführen von Multiplikationen im BFV-Schema bessere Benchmarking-Ergebnisse liefert.

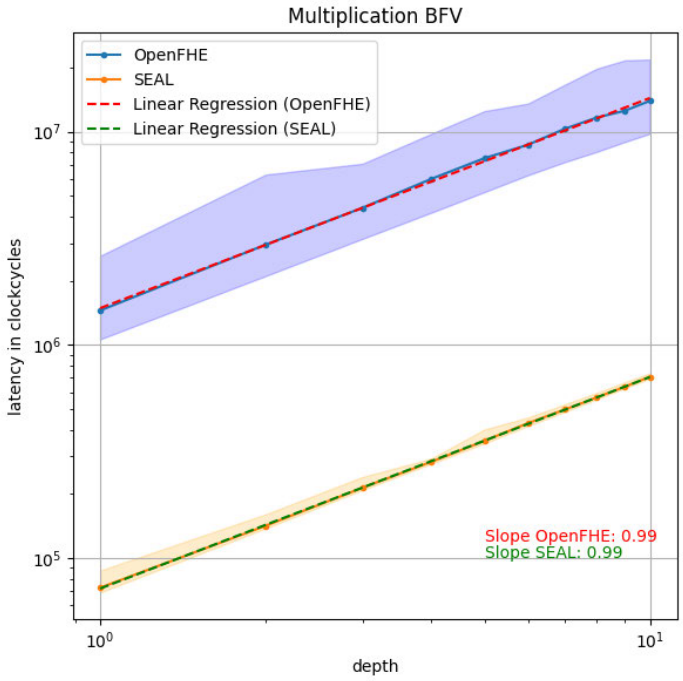

Abbildung 4.2: Laufzeiten Multiplikation BFV

4.2.3 Versuch 3: Addition CKKS

Im dritten Versuch wurde die Abhängigkeit der Latenz bei der Ausführung von Additionen im CKKS-Schema untersucht. Das Benchmarking wurde wie unter 4.1.2 beschrieben durchgeführt. Aus den daraus resultierenden 20 Testreihen wurde für jeden Messpunkt das Minimum und das Maximum ermittelt und gelöscht. Der Mittelwert aus den verbleibenden Messwerten soll die im Benchmarking entstandenen Messfehler ausgleichen und wird in den Graphen SEAL und OpenFHE im Log-Log-Diagramm abgebildet (Abbildung 4.3). Auffällig ist, dass die Messwerte des SEAL Graphen deutlich kleiner sind. Daraus ist zu schlussfolgern, dass die Addition im CKKS-Schema schneller in der Microsoft SEAL Bibliothek als in der OpenFHE Bibliothek abläuft. Beide Bibliotheken weisen eine Anstieg von 1,0 auf, woraus erkennbar ist, dass beide Messungen einen linearen Anstieg besitzen. Bei der hellblauen sowie der hell-orange gekennzeichneten Messabweichung sind keine Unterschiede erkennbar. In beiden Messungen sind kaum erkennbare Messfehler aufgetreten. Dies ist ein positives Indiz für die Stabilität beider Bibliotheken im CKKS-Schema. Trotz dessen schneidet die Microsoft SEAL Bibliothek bei der Addition im CKKS-Schema insgesamt besser ab. Im Vergleich zur Addition im BFV-Schema ist die Addition im CKKS-Schema deutlich langsamer. Ursache dafür ist die aufwendigere float-Arithmetik des CKKS-Schemas.

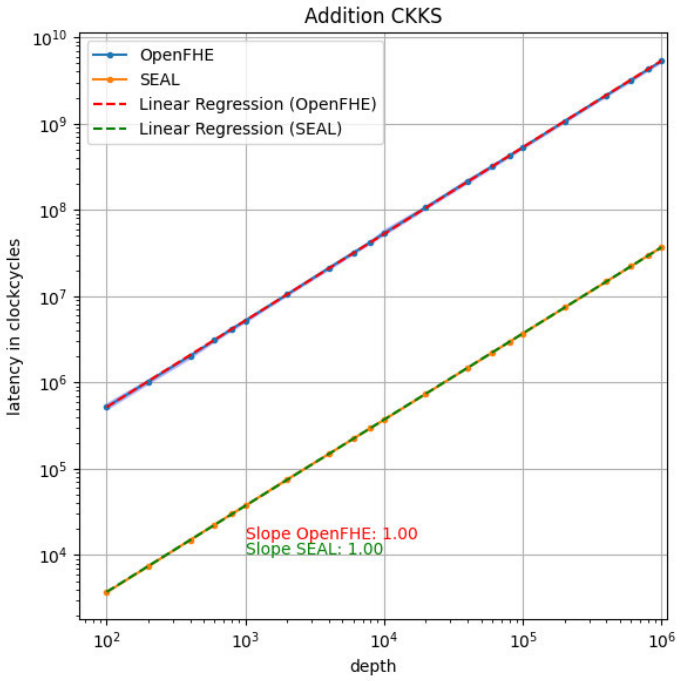

Abbildung 4.3: Laufzeiten Addition CKKS

4.2.4 Versuch 4: Multiplikation CKKS

In Versuch 4 wurde die Abhängigkeit der Latenz bei der Ausführung von Multiplikationen im CKKS-Schema untersucht. Das Benchmarking wurde, wie unter 4.1.2 beschrieben, durchgeführt. In den insgesamt 100 resultierenden Testreihen wurde für jeden Messpunkt das Minimum und das Maximum bestimmt und entfernt. Danach wurde der Mittelwert für jeden Messpunkt aus den verbleibenden Werten bestimmt, um die im Benchmarking entstandenen Messfehler auszugleichen. Die von Messfehlern bereinigten Benchmarking-Resultate werden in den Graphen SEAL und OpenFHE im Log-Log-Diagramm abgebildet (Abbildung 4.4). Auch im vierten Versuch liegen die Messwerte von SEAL deutlich näher an der x-Achse als die Messpunkte von OpenFHE. Daraus ist zu schlussfolgern, dass die Microsoft SEAL Bibliothek weniger Clockcycles für Multiplikation als die OpenFHE Bibliothek benötigt. Auch die Messabweichungen sind bei der der Microsoft SEAL Bibliothek deutlich geringer im Vergleich zur OpenFHE Bibliothek. Besonders auffällig ist der vergleichsweise niedrige Anstieg beider Geraden. Sowohl bei SEAL als auch bei OpenFHE sind teils große Abstände zwischen Regressionsgerade und Messpunkt erkennbar. Grundsätzlich ist aber trotzdem ein linearer Anstieg beider Graphen erkennbar. Insgesamt liefert die Microsoft SEAL Bibliothek bessere Benchmarking-Resultate bei der Multiplikation im CKKS-Schema.

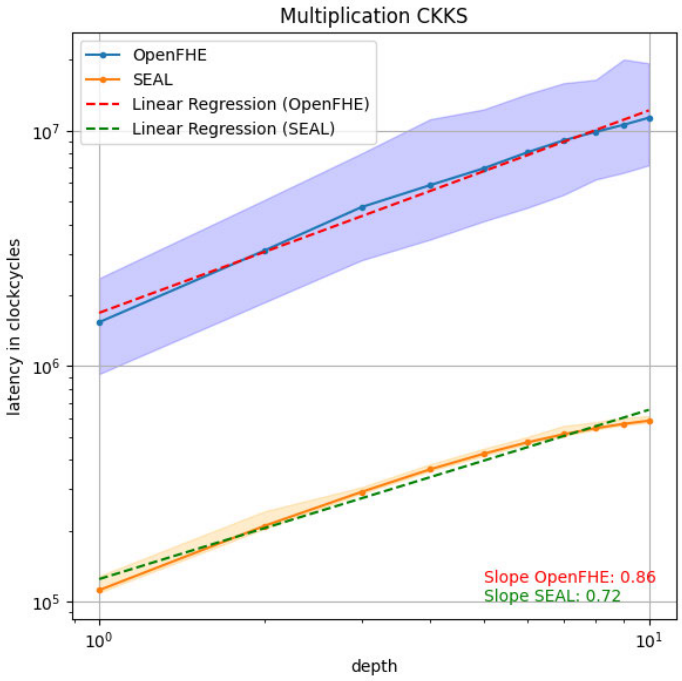

Abbildung 4.4: Laufzeiten Multiplikation CKKS

4.2.5 Überblick

Es folgt eine Übersicht, die einen schnellen und eindeutigen Überblick über alle erzielten Benchmarking-Resultate geben soll (Abbildung 4.5). Die Abbildung zeigt ein viergeteiltes Säulendiagramm, wobei in jedem der vier Diagrammteile Verschlüsselungs-, Entschlüsselungs-, Additions- und Multiplikationslatenz eines Schema in einer Bibliothek auf der x-Achse dargestellt wird. Als zu betrachtende Schemata wurde wie in Versuch 1-4 das BFV-Schema und dass CKKS-Schema gewählt. Die in der Abbildung zu vergleichenden Bibliotheken sind die Microsoft SEAL Bibliothek und die OpenFHE Bibliothek. Auf der y-Achse wird die Latenz in Clockcycles angegeben. Die Verschlüsselungszeit wurde für das Verschlüsseln eines Plaintextes aus den Versuchen 1-4 gemessen. Außerdem wurden Minimum und Maximum der resultierenden Messergebnisse bestimmt und entfernt. Anschließend wurde die Latenz in Clockcycles gemittelt und im Säulendiagramm dargestellt. Für die Entschlüsselungszeit wurde der gleiche Prozess durchlaufen. Sowohl Additionszeit als auch Multiplikationszeit stammen aus dem letzten Messpunkt der durchgeführten Versuche 1-4. Das bedeutet, dass in dem Säulendiagramm nur die Zeitmessungen mit 1 000 000 Additionen (Tabelle 4.2) und 10 durchgeführten Multiplikationen (Tabelle 4.3) betrachtet werden. Aufgrund der größeren Anzahl an Operationen ist die Additionslatenz insgesamt deutlich höher als die Multiplikationslatenz. Wird aber die Latenz von einer einzelnen Operation betrachtet, so ist die Latenz von Additionen deutlich niedriger als bei Multiplikationen. Auffällig ist, dass die Microsoft SEAL Bibliothek in beiden Schemata und allen Vergleichspunkten die bessere Resultate liefert als die OpenFHE Bibliothek. Weiterhin fällt auf, dass die Verschlüsselungszeit immer größer ist als die Entschlüsselungszeit.

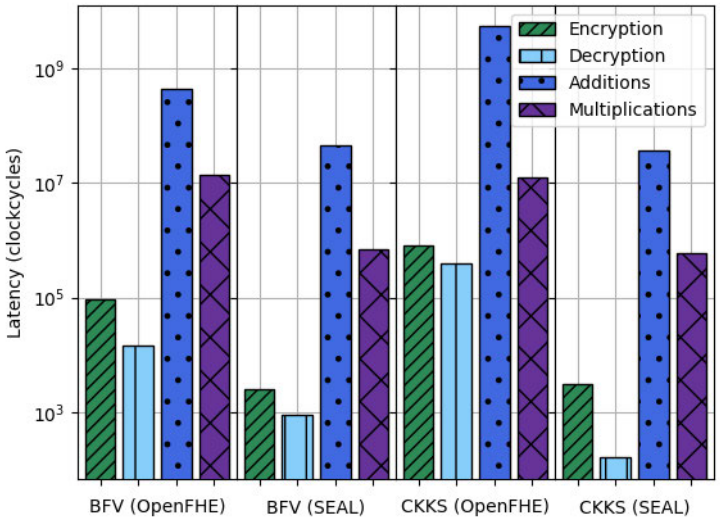

Abbildung 4.5: Vergleich SEAL vs. OpenFHE

4.2.6 Schlussfolgerung

Nach der Auswertung der im Benchmarking entstandenen Resultate ist zu schlussfolgern, dass die Microsoft SEAL Bibliothek die leistungsstärkere Bibliothek ist. Sie weist geringer Latenzen bei Additionen und Multiplikationen im BFV- und CKKS-Schema als die OpenFHE Bibliothek auf. Des Weiteren sind auch die Entschlüsselung und Verschlüsselung deutlich weniger zeitintensiv als bei der OpenFHE Bibliothek. Auch die vergleichsweise geringeren Messabweichungen der SEAL Bibliothek sind bemerkenswert. Die insgesamt sehr guten Benchmarking-Reslutate und die ausführliche, bereitgestellte Notation sind der Grund, weshalb das Anwendungsbeispiel in Kapitel 5 mithilfe der Microsoft SEAL Bibliothek umgesetzt wird.

5 Anwendungsbeispiel Onlinewahlen

Dieses Kapitel soll einen praktischen Einblick in mögliche Anwendungsbereiche von homomorphen Verschlüsselungen bieten. Mithilfe der in Kapitel 4 gesammelten Erkenntnisse wird deshalb ein Anwendungsbeispiel für homomorphe Verschlüsselungen simuliert. Als erstes werden dafür mögliche Anwendungsgebiete von homomorphen Verschlüsselungen beschrieben. Danach wird der Fokus auf das Anwendungsgebiet der Onlinewahlen gelegt und mögliche Richtlinien für Onlinewahlen benannt. Anschließend wird der Aufbau der Onlinewahl-Umgebung erläutert. Zum Schluss wird die Simulation für verschiedenen Szenarien durchgeführt und ein Fazit zum vorgestellten Usecase gezogen.

5.1 Formulierung des Anwendungsbeispiels

In folgendem Abschnitt wird ein praktisches Anwendungsbeispiel formuliert. Dazu werden als erstes mögliche Anwendungsgebiete homomorpher Verschlüsselungen vorgestellt. Danach wird die Thematik der Onlinewahlen aufgegriffen und näher beleuchtet. Zum Schluss wird eine Richtlinie für Onlinewahlen vorgestellt und der Umfang des Anwendungsbeispiels in dieser Arbeit eingegrenzt.

5.1.1 Anwendungsgebiete von homomorphen Verschlüsselungen

Dank der Weiterentwicklung des Internets bildeten sich auch neue Konzepte der Datenverarbeitung. Cloud-Computing[4] ist eine immer häufiger verwendete Form der Datenverarbeitung und deshalb auch einer der am häufigsten verwendeten Begriffe in der Informationstechnik [37]. Cloud-Computing bietet viele Vorteile, wie zum Beispiel Virtualisierung, Mehrnutzerzugriff und die Möglichkeit des Auslagerns von leistungsfordernden Programmen auf kostengünstigere und leistungsstärkere externe Computerressourcen. Gleichzeitig werden aber auch Sicherheits- und Datenschutzprobleme vergrößert, weil bei herkömmlichen Cloud Computing Anwendungen nur im entschlüsselten Zustand ein auf der Cloud gespeichertes Objekt bearbeitet werden kann und Cloud-Besitzer dadurch Zugriff auf unverschlüsselte Daten erhalten. Dieses Problem kann aber durch den Einsatz von homomorphen Verschlüsselungen minimiert werden, denn homomorph-verschlüsselte Cloud-Systeme sind dagegen in der Lage, Operationen auf verschlüsselten Objekten auszuführen (siehe Abbildung 5.1). Die auf der Cloud gespeicherten Objekte liegen zu keinem Zeitpunkt unverschlüsselt vor und sind somit nur mit dem entsprechenden Private Key im Klartext zugänglich [37].

Im folgenden werden Anwendungsgebiete von homomorphen Verschlüsselungen benannt.

[4] „Cloud Computing bezeichnet das dynamisch an den Bedarf angepasste Anbieten, Nutzen und Abrechnen von IT-Dienstleistungen über ein Netz. [...] Die Spannbreite der im Rahmen von Cloud Computing angebotenen Dienstleistungen umfasst das komplette Spektrum der Informationstechnik und beinhaltet unter anderem Infrastruktur (z. B. Rechenleistung, Speicherplatz), Plattformen und Software [5].“

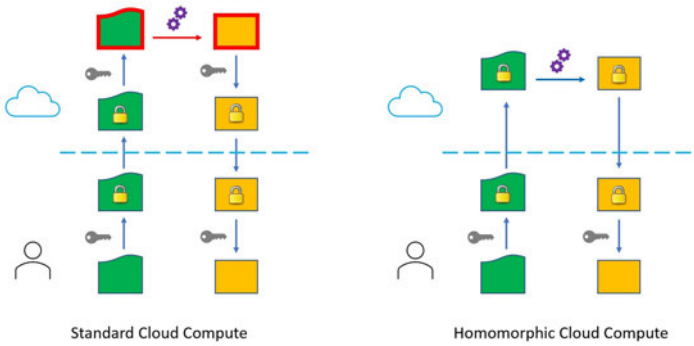

Abbildung 5.1: Herkömmliches vs. homomorphes Cloudcomputing [38]

Verschlüsselte Gendatenbanken

Aufgrund der rapiden Entwicklung von Genomsequenzierungstechnologien wird davon ausgegangen, dass in Zukunft große Menge an Genomdatensätzen vor allem bei Forschungsinstitutionen entstehen. Genomdaten finden sehr viele Anwendungsgebiete, wie zum Beispiel in Forschung, Gesundheitswesen und allgemeiner Forensik. Bei Genomdaten handelt es sich aber um sensible Daten, die Datenschutzprobleme erzeugen können. Es ist sogar möglich, Identitätsinformationen aus unbeschrifteten Genomdaten zu extrahieren. Um dies zu verhindern, sollten Genomdaten verschlüsselt werden. Damit die Forschung und der Austausch zwischen Forschungsinstituten nicht gefährdet werden, müssen die Arbeit an Genomdatensätzen ermöglicht und gleichzeitig die sensiblen Daten geschützt werden. Hierfür gibt es schon Ansätze, die auf homomorphen Verschlüsselungen basieren [39].

Verschlüsselter forensischer Bildabgleich

Forensische Bilderkennung ist ein probates Mittel, um illegale Bilder auf Festplatten oder Netzwerkspeichern zu erkennen. Damit illegale Bilder identifiziert werden können, wird von Ermittlungsbehörden ein Hashwertdatensatz aus Hashwerten illegaler Bilder zum Abgleich verwendet. Der zu durchsuchende Datensatz wird mit dem Hashwertdatensatz der Ermittlungsbehörde verglichen. Sollte es dabei zu Übereinstimmungen zwischen den beiden Datensätzen kommen, beinhaltet der zu durchsuchende Datensatz illegale Bilder. Damit Täter keinen Zugriff auf den Hashwertdatensatz haben und prüfen können, ob ihre Bilder als illegal erkannt werden können, muss der Hashwertdatensatz verschlüsselt werden. Homomorphe Verschlüsselungen ermöglichen eine Verschlüsselung des Hashwertdatensatzes und einen Vergleich der zu durchsuchenden Datensätze mit dem Hashwertdatensatz im verschlüsselten Zustand. Der Vorteil dabei ist, dass niemand ohne private Key Zugriff auf den Hashwertdatensatz hat. Gleichzeitig hat die Ermittlungsbehörde nur Kenntnis über die für sie relevanten Daten (das Vorkommen illegaler Bilder im zu durchsuchenden Datensatz) [40].

Künstliche Intelligenz unter Wahrung der Privatsphäre

Künstliche Intelligenz (kurz KI) wird immer häufiger zum Ausführen verschiedener rechenintensiver Dienstleistungen, in verschiedenen Bereichen, wie z.B. Gesundheitswesen, Finanzwesen und Cybersicherheit, eingesetzt. Dabei werden rechenintensive Prozesse, in denen mit privaten und sensiblen Daten gearbeitet wird, meist auf Clouds ausgelagert. Das Problem ist, dass die sensiblen Nutzerdaten dabei von Cloud- und Modelleigentümern einsehbar sind. Um dies zu verhindern, können homomorphe Verschlüsselungen verwendet werden. Diese Form der Verschlüsselung ermöglicht beispielsweise Modell-Training und Inferenz[5] auf verschlüsselten Daten. Dadurch kann die Privatsphäre der Dateninhaber geschützt werden [42].

5.1.2 Homomorphe Verschlüsselungen bei Onlinewahlen

„Die Ausübung der Staatsgewalt geht vom Volk aus und wird durch Wahlen ermöglicht [43]." Während der Corona-Pandemie stieg der Bedarf für die Digitalisierung von Verwaltungsprozessen. Dies betrifft auch die Wahlen verschiedenster Ämter, weshalb Unternehmen vermehrt auf die Durchführung von Online-Wahlen setzen [44]. Ein Vorteil von Onlinewahlen ist die Zugänglichkeit, das bedeutet, jeder Wahlberechtigte kann von einem beliebigen Ort mit Internetzugang aus abstimmen und muss nicht physisch im Wahllokal (oder bei Briefwahlen in der Post) erscheinen. Trotz vieler weiterer positiver Effekte von Onlinewahlen hat der Ausschuss für Bildung, Forschung und Technikfolgenabschätzung am 06.04.2022 beschlossen, dass es vorerst kein E-Voting bei Bundes- oder Landtagswahlen geben wird. Grund dafür ist ein bestehender hoher Forschungsbedarf und das Abwarten der Sozialwahlen 2023, wo zum ersten Mal Ende-zu-Ende-Verifizierbarkeit eingesetzt wird [45]. Für die anstehenden Sozialwahlen wurde eine Richtlinie vom Bundesamt für Sicherheit in der Informationstechnik (BSI) veröffentlicht. Die technische Richtlinie ergänzt die Online-Wahl-Verordnung[6] um die IT-sicherheitstechnischen Anforderungen, die für die Durchführung der Onlinewahl im Rahmen des Modellprojektes erforderlich sind. Im folgenden wird der Aufbau einer Onlinewahl anhand der Technische Richtlinie TR-03162 des BSI erklärt [46].

In der Technischen Richtlinie werden grundlegende Festlegungen und Anforderungen zur Durchführung des Modellprojektes beschrieben. Danach wird auf die Vorbereitung und Durchführung von Onlinewahlen eingegangen. Zuletzt wird die Ermittlung des Wahlergebnisses und die Nachbereitung der Wahl definiert [46].

Die Durchführung einer Onlinewahl unterliegt strengen Richtlinien. Im Folgenden werden die wichtigsten Richtlinien erwähnt. Weiterhin wird eingegrenzt, welche Prozesse einer Onlinewahl in der Wahlsimulation in den Abschnitten 5.2 und 5.3 dargestellt werden sollen.

[5] Die Inferenzphase beschreibt einen Abschnitt, der auch als Wissenverarbeitung bezeichnet wird. In der Inferenzphase erhält ein trainiertes und validiertes Maschine-Learning-Modell Benutzeranfragen, die es dann mit der vorher antrainierten Wissensbasis beantwortet [41].

[6] „Verordnung über die technischen und organisatorischen Vorgaben für die Durchführung einer Onlinewahl im Rahmen des Modellprojekts nach § 194a Fünftes Buch Sozialgesetzbuch [46]"

„Zur angemessenen Absicherung der Online-Wahl wird die Methodik und Systematik des BSI IT-Grundschutzes angewandt [46].“ Das IT-Grundschutzkonzept baut auf einer Schutzbedarfbestimmung auf und sieht Strukturanalyse, Informationssicherheitsmanagement, IT-Sicherheitskonzept, Risikoanalyse und IT-Notfallmanagement vor [46].

Die Technische Richtlinie sieht folgende Anwendungen und Kommunikationswege als Bestandteile einer Onlinewahl vor:

- Wahlkennzeichen und Wählerverzeichnis Online
- Online Wahlsystem
  - Wahlplattform
  - Elektronische Wahlurne
  - Online Stimme
  - Elektronische Liste mit Wahlkennzeichen
  - Ermittlung des Online-Wahlergebnisses
  - Zeitserver
- Aufbewahrung und Vernichtung

In der durchgeführten praktischen Simulation werden nur die elektronische Wahlurne, die Online-Stimme und die Ermittlung des Online-Wahlergebnisses dargestellt. Die elektronische Wahlurne enthält die Onlinestimmen und darf semi-homomorph verschlüsselt werden. Online-Stimmen sind Datenstrukturen, welche in der elektronischen Wahlurne gespeichert werden und das Votum des Wählers abbilden. Bei der Ermittlung des Online-Wahlergebnisses werden die Online-Stimmen in der elektronischen Wahlurne ausgewertet. „Wird eine semi-homomorphe Verschlüsselung genutzt, ermöglicht diese die Auszählung der Stimmen, ohne dass einzelne Stimmen entschlüsselt werden müssen. Bei dieser Methode muss zusätzlich mit einem Zero-Knowledge-Proof[7] die Korrektheit aller Stimmen überprüft werden [46].“ Der Zero-Knowledge-Proof wurde in der im Praxisteil durchgeführten Simulation durch eine Validierung durchgeführt. Zusätzlich wurden sowohl elektronische Wahlurne als auch Online-Stimmen homomorph verschlüsselt. Auf weitere Bestandteile der Technischen Richtlinie wird in der folgenden praktischen Simulation einer Onlinewahl aus Komplexitätsgründen nicht eingegangen.

5.2 Aufbau der Onlinewahl-Umgebung

Ziel der Simulation ist es, die Eignung der homomorphen Verschlüsselung für Onlinewahlen in verschiedenen Szenarien experimentell zu untersuchen. Eine kleine Wahl mit ballot_N = 100 Wahlzetteln, $ballot\_size = 12$ Wahloptionen und n_votes Wahlstimmen soll im Folgenden simuliert und detailliert beschrieben werden. Dabei soll das Grundprinzip von Onlinewahlen mit homomorphen Verschlüsselungen dargestellt werden und Vor- und Nachteile dieses Wahlprinzips beleuchtet werden. Sowohl die Größe des Wahlzettels, die Anzahl der Wahlzettel, als auch die Anzahl der Wahlstimmen lassen sich problemlos auf andere Wahlumfänge (z.B. Bundestagswahl) abändern. Der komplette Programmcode ist im Anhang A dieser Arbeit zu finden. Im Folgenden wird der Ablauf einer Onlinewahl beispielhaft beschrieben.

[7] „Zero-Knowledge-Beweise, vereinzelt auch kenntnisfreie Beweise genannt, sind Protokolle, bei denen eine Partei eine andere davon überzeugt, dass sie ein Geheimnis (z. B. einen Schlüssel) kennt, ohne dieses selber preiszugeben [47].“

Im Vorfeld werden die für die simulierte Wahl relevanten Parameter festgelegt. So ist die Größe des Wahlzettels (Anzahl der Wahloptionen) auf zwölf festgelegt und die Anzahl der zu generierenden Wahlzettel wird auf 100 festgesetzt. Jeder Wähler (im Folgenden als Voter bezeichnet) darf eine Wahlstimme abgeben (Zeile 33-35 im Anhang A).

Zusätzlich dazu werden in Zeile 146-172 im Anhang die für die homomorphen Verschlüsselungen notwendigen Parameter wie z.B. Schema, PolyModulusDegree und Kontext definiert. Des Weiteren werden alle Schlüssel für Verschlüsselung, Entschlüsselung, Relinearisation und Rotation generiert.

Im ersten Schritt der Wahlsimulation werden die 100 Wahlstimmen zufällig generiert. Dabei ist zu beachten, dass sowohl gültige als auch ungültige Wahlstimmen generiert werden sollen, um die Wahl so realistisch wie möglich abzubilden. Eine gültige Wahlstimme ist ein Ganzzahlvektor der die Wahlentscheidung mit Eins und Null repräsentiert. Jeder simulierter Voter darf in dieser Onlinewahl genau n_votes Wahlstimmen abgeben. Die sonst in analogen Wahlen durch ein Kreuz dargestellte Wahlstimme wird bei der Onlinewahl mit einer Eins (1) dargestellt. Die nicht ausgewählten Wahlkandidaten ($ballot\_size - 1$) werden mit einer Null (0) symbolisiert.

Die Anzahl der ungültigen Stimmen lag bei der Bundestagswahl 2021 bei 492 495 Erststimmen, was einen prozentualen Anteil von rund einen 1% ergibt. [48]
Dieser Anteil an ungültigen Stimmen wird auch in der simulierten Onlinewahl generiert. Eine ungültige Stimme wird durch einen Vektor mit zwölf zufälligen, ganzzahligen Werten zwischen -100 und 100 dargestellt. Die Generierung von gültigen und ungültigen Wahlzetteln findet in Zeile 205-222 (im Anhang A) statt.

Anzumerken ist, dass der Vektor, auf den ein Wahlzettel generiert wurde, die festgelegte Plaintextgröße von 4096 Elementen besitzt. Der Wahlzettel belegt lediglich die ersten $ballot\_size$ Elemente. Die unbelegten Elemente werden mit Nullen gefüllt. Der Vektor wird als 2×2048 Matrix betrachtet.

Der Vektor wird nach der Generierung des Wahlzettels sofort encodet und verschlüsselt. Anschließend wird der verschlüsselte Wahlzettel-Vektor zur Überprüfung auf einen LIFO-Stapel[8] abgelegt. Nachdem alle generierten Wahlzettel abgelegt wurden startet der Wahlzettel-Validierungsprozess.

Die Validierung der Wahlzettel findet wie folgt statt. Als erstes wird der verschlüsselte Wahlzettel, welcher zuletzt dem LIFO-Stapel hinzugefügt wurde, vom Stapel genommen und kopiert. Der kopierte verschlüsselte Wahlzettel durchläuft nun die $check\_vote\{\}$-Funktion in Zeile 38-80 im Anhang A. Aufgabe dieser Funktion ist es zu überprüfen, ob der vorliegende verschlüsselte Wahlzettel ein gültiger oder ungültiger Wahlzettel ist. Die Überprüfung muss auf dem verschlüsselten Wahlzettel ausgeführt werden und der Wahlzettel darf zu keinem Zeitpunkt entschlüsselt werden können, um den Wahlgrundsatz der geheimen Wahl nicht zu verletzen.

[8] Der LIFO-Stapel wird auch als „Last in first out Stack“ bezeichnet. Stapeln nach dem LIFO-Prinzip bedeutet, „dass die Daten, die als letzte eingefügt wurden, als erste wieder vom Stack genommen werden[...] [49].“

Um diesen Zustand zu gewährleisten, wird sich der zyklischen Vektorrotation (siehe Definition 2.5) bedient und diese Operation auf den homomorphen Ciphertext-Vektor angewendet. Da die Überprüfung in zwei unterschiedlichen Schritten stattfindet, wird der Ciphertext zuerst in der oben erwähnten Matrix-Schreibweise betrachtet und die ersten 2048 Vektorelemente auf die letzten 2048 Elemente rotiert. Dadurch wird der Wahlzettel aus den ersten *ballot_size* Elementen der ersten Zeile der Matrix in die ersten *ballot_size* Elemente der zweiten Zeile der Matrix kopiert.

Die erste Zeile dient der Feststellung, dass der Wahlzettel ausschließlich nicht-negative Ganzzahlen $\{0, 1, 2, \dots\}$ beinhaltet. In der zweiten Zeile der Matrix wird die Überprüfung, dass genau *n_votes* Einsen auf dem Wahlzettel eingetragen wurde, durchgeführt.

Zuerst wird die zweite Zeile der Matrix quadriert und anschließend relinearisiert. Durch das Quadrieren wird gewährleistet, dass jeden Zahl in der zweiten Zeile nicht-negativ ist. Mit der Relinearisierung wird sichergestellt, dass der Stimmzettel anschließend weiterverarbeitet werden kann.

Nun wird die Matrix, wie in Definition 2.5 erläutert, zyklisch nach rechts rotiert:

$$\mathrm{Rot}(C, 2^i) \quad \text{für alle} \quad i \in \mathbb{N}_0, i < 11.$$

Nach jedem Rotationsschritt werden die Ausgangsmatrix und die rotierte Matrix addiert:

$$\mathrm{Add}(C, C')$$

Der Rotations- und Additionsprozess ist in Abbildung 5.2 vereinfacht an einem Beispiel dargestellt. Der Vektor C_1 stellt den verschlüsselten Wahlzettel dar. Als erstes wird der

$$\begin{aligned}
C_0: &\quad [a, b, c, d, 0, 0, 0, 0] \\
C_0' = \mathrm{Rot}(C_0, 2^0): &\quad [0, a, b, c, d, 0, 0, 0] \\
C_1 = C_0 + C_0': &\quad [a, a+b, b+c, c+d, d, 0, 0, 0] \\
C_1' = \mathrm{Rot}(C_1, 2^1): &\quad [0, 0, a, a+b, b+c, c+d, d, 0] \\
C_2 = C_1 + C_1': &\quad [a, a+b, a+b+c, a+b+c+d, b+c+d, c+d, d, 0] \\
C_2' = \mathrm{Rot}(C_2, 2^2): &\quad [b+c+d, c+d, d, 0, a, a+b, a+b+c, a+b+c+d]] \\
\text{Ergebnis}: &\quad [a+b+c+d, a+b+c+d, a+b+c+d, a+b+c+d, \\
&\quad\ a+b+c+d, a+b+c+d, a+b+c+d, a+b+c+d]
\end{aligned}$$

Abbildung 5.2: Veranschaulichung des durchgeführten Rotationsprozesses

Vektor kopiert und die Kopie um 2^0 rotiert (siehe C_1'). Anschließend werden C_1 und C_1' miteinander addiert und die Summe in Ciphertext C_2 abgebildet. Dieser Ciphertext wird wieder kopiert, die Kopie um 2^1 rotiert und in C_2' abgebildet. Anschließend wird die Summe von C_2 und C_2' in C_3 berechnet. Im letzten Zyklus wird C_3 kopiert und die Kopie anschließend in C_3' um 2^2 rotiert. Das finale Ergebnis des hier vorgestellten Rotationsprozesses entsteht durch die Addition von C_3 und C_3.

Nach erfolgreichem Durchlauf aller elf Rotationen, wird der Ciphertext entschlüsselt und dekodiert. Da jedes Element des Wahlzettels mit jedem anderen Element des Wahlzettels addiert wurde, lässt sich kein Rückschluss auf die Wahlentscheidung der wählenden Person ziehen und außerdem kann nun überprüft werden, ob der Wahlzettel gültig ist.

Um die Gültigkeit des Wahlzettels zu bestimmen, werden zwei Elemente (e_1 aus Zeile 1 und e_2 aus Zeile 2) aus der Matrix ausgewählt. Sind beide Elemente $e_1 = e_2$ gleich groß und erfüllen die Bedingung $e_1, e_2 \leq n\_votes$, so ist der Wahlzettel gültig, denn es bedeutet, dass die Summe aller Elemente des Wahlzettels und die Summe aller quadrierten Elemente n_votes ist. Aus dieser Erkenntnis lässt sich schlussfolgern, dass im ganzen Wahlzettel kleiner oder gleich n_votes Elemente den Wert 1 und $ballot\_size - n\_votes$ Elemente den Wert 0 besitzen. Auch die Abgabe von einem Teil der maximalen Anzahl an Wahlstimmen n_votes oder eines leeren Stimmzettels wird als gültiger Wahlzettel erkannt. Wenn $e_1 > n\_votes \vee e_2 > n\_votes$, dann ist der Wahlzettel ungültig.
Die Gültigkeit des Wahlzettels wird in der Variable *checkpointer* gespeichert und in die main()-Methode zurückgegeben.

In der main()-Methode wird nun eine Selektion durchgeführt. Die Wahlzettelvektoren mit $checkpointer = 1$ werden einmal auf den Wahlurnenvektor addiert und anschließend vom LIFO-Stapel entfernt. Für alle Wahlzettelvektoren, die durch $checkpointer = 0$ markiert wurden, wird eine Meldung, dass es sich bei dem vorliegenden Wahlzettel um einen ungültigen Wahlzettel handelt und dieser nicht mit in das Wahlergebnis einfließt, ausgegeben. Weiterhin wird die Anzahl aller ungültigen Wahlzettel in der Variable $n\_invalid\_votes$ gespeichert. Danach werden auch die ungültigen Wahlzettel vom LIFO-Stapel entfernt.

Nach Ende des Wahlzeitraums, wird dann die Wahlurne entschlüsselt und dekodiert, sodass die wahlsiegende Person ermittelt werden kann. Es folgt der Aufruf der *print_result*()-Methode.

In der *print_result*()-Methode in Zeile 83 bis 140 im Anhang A wird das Wahlergebnis bekannt gegeben. Als erstes wird der Wahlurnenvektor, welcher das Wahlergebnis repräsentiert, ausgegeben. Der Wahlurnenvektor bildet die Summe aller gültigen Stimmen ab und darf nicht verändert werden. Im folgenden wird der Wahlgewinner bestimmt (Zeile 97-126 im Code 5.2). Um Wahlergebnisse unterschiedlicher Größe evaluieren zu können, wird ein Vektor erstellt, in welchem die Kandidaten mit den meisten Wahlstimmen notiert werden. Zuerst wird der erste Kandidat ausgewählt und die zugehörige Stimmenanzahl wird als höchste Stimmenanzahl betitelt. Weiterhin wird die Kandidatennummer in den Vektor geschrieben. Nun wird die Stimmenanzahl der nächsten Kandidaten mit der höchsten Stimmenzahl verglichen. Ist die Stimmenanzahl kleiner als die höchste Stimmenzahl, wird zum nächsten Kandidaten gesprungen. Bei gleicher Größe wird die Kandidatennummer dem Vektor hinzugefügt. Falls die ausgewählte Stimmenanzahl größer ist als die höchste Stimmenanzahl, so wird diese zur neuen höchsten Stimmenanzahl, die Elemente des Vektors werden gelöscht und die Nummer des aktuellen Kandidaten wird eingetragen. Anschließend wird zur nächsten zu wählenden Person gesprungen.
Nach Durchlaufen aller *ballot_size* Kandidaten, wird der Wahlsieger durch die Ausgabe des Vektors verkündet.

```
1  {
2    std::cout << "Number " << i + 1 << " with " << vote_result[i]
```

```
            << "  to vote Political Party / Person *** " << std::endl;
}
// output measured Time
std::cout << "Needed Time = " << time / CLOCKS_PER_SEC << " Seconds"
          << std::endl;

// select the election winner(s)
int most_votes = vote_result[0];
vector<int> election_winners;
election_winners.push_back(0 + 1);

for (int i = 1; i < vote_result.size(); i++)
{
  if (vote_result[i] == most_votes)
  {
    election_winners.push_back(i + 1);
  }
  if (vote_result[i] > most_votes)
  {
    election_winners.clear();
    most_votes = vote_result[i];
    election_winners.push_back(i + 1);
  }
```

Als letztes werden die gültigen Stimmen gezählt und ein Wahlfeedback ausgegeben. Dieses Feedback beinhaltet die Anzahl aller Wahlzettel, die Anzahl der gültigen Wahlzettel und die Anzahl aller ungültigen Wahlzettel. Nach der Ausgabe des Feedbacks ist das Programm zur Simulation einer Wahl beendet.

5.3 Durchführung der Simulation von Onlinewahlen

Um die Kombination aus Operationen, die in dem Onlinewahlen-Programm implementiert sind, wurde die Simulation mit unterschiedlichen Parametern ausgeführt. Die dabei veränderten Parameter sind *ballot_size*, ballot_N und *n_votes*. Im folgenden Abschnitt wird auf die dabei erzielten Ergebnisse eingegangen.

5.3.1 Szenario 1 - Betriebsrat-Wahlen

In Szenario eins wird angenommen, dass der Betriebsrat der Agentur für Innovation in der Cybersicherheit (kurz Cyberagentur) gewählt werden soll. Für die Wahl zugelassen sind alle Mitarbeiter der Cyberagentur. Die genaue Mitarbeiterzahl der Cyberagentur ist unbekannt, aber in einem Artikel der Süddeutschen Zeitung wird von einer „finalen Beschäftigtenzahl von 100 Mitarbeitenden [50]“ berichtet. Aufgrund dieser Annahme wird ballot_N auf 100 gesetzt. Die Anzahl der zur Wahl stehenden Personen (*ballot_size*) wurde zufällig gewählt und auf 10 Personen festgesetzt. Jeder Voter darf genau eine Stimme ($n\_votes = 1$) abgeben. Die ersten fünf Stimmzettel sind in Abbildung 5.3 im Klartext dargestellt. Klarerweise werden alle Stimmzettel nicht ausgegeben, sondern sofort verschlüsselt, damit der Wahlgrundsatz der geheimen Wahl nicht verletzt wird. Zur Veranschaulichung werden in dieser Simulation die Wahlzettel im Klartext ausgegeben (Kommentar Zeile 224-231).

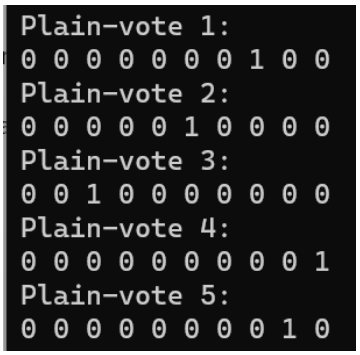

Abbildung 5.3: Szenario 1: Fünf Stimmzettel

Nach der Generierung der Wahlzettel, werden die Wahlzettel verschlüsselt und, wie oben beschrieben, auf ihre Gültigkeit überprüft. Dabei fällt auf, dass von den insgesamt 100 generierten Wahlzetteln ein Wahlzettel ungültig ist. Der Ungültige Wahlzettel wird in Abbildung 5.4 dargestellt.

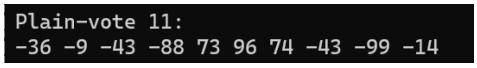

Abbildung 5.4: Szenario 1: Ungültiger Stimmzettel

Nach erfolgreicher Validierung folgt das Hinzufügen der Wahlzettel zur Wahlurne und die anschließende Auswertung der Wahlurne. Die aus der Auswertung resultierenden Wahlergebnisse werden in Form einer Ergebnistabelle ausgegeben. Weiterhin werden der Wahlsieger und Informationen zur Anzahl von gültigen und ungültigen Stimmen bekannt gegeben. Alle Wahlergebnisse werden in Abbildung 5.5 dargestellt.

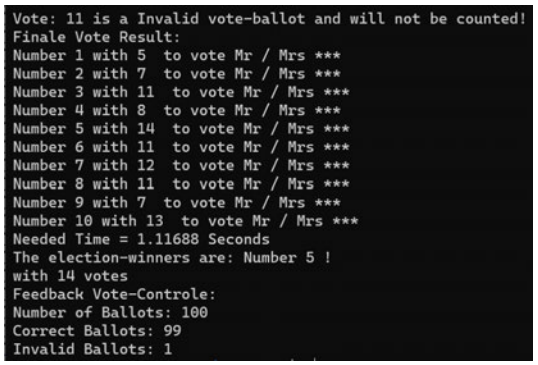

Abbildung 5.5: Szenario 1: Überblick Wahlergebnis

Wichtig für eine mögliche Anwendung in einer realen Wahl ist die Zeit, die der Ablauf dieser Onlinewahl in Anspruch nimmt. Die Zeiterfassung erfolgt für die Validierung des Wahlzettels, die Addition auf die Wahlurne sowie den Entschlüsselungsprozess der Wahlurne. Für die Zeiterfassung irrelevant ist die Wahlzettel-Generierung, da in einer realen Wahl die Wahlzettel in einem fest terminierten Zeitraum vom Voter persönlich erstellt und abgeschickt werden müssen. Da auch der Verschlüsselungsprozess der einzelnen Wahlzettel in die Phase der Wahlzettelerstellung fällt, wird dieser ebenfalls nicht zeitlich erfasst.
Wie bereits in Abbildung 5.5 zu sehen, benötigt die dargestellte Wahlsimulation rund 1,12

Sekunden. Der Mittelwert bei 10 Durchläufen beträgt 1,03 Sekunden. Wird die Wahlsimulation ohne Validierung und ohne ungültige Wahlzettel durchgeführt, so benötigt ein simulierter Wahldurchlauf im Schnitt 0,0029 Sekunden.

5.3.2 Szenario 2 - Fakultätsrat-Wahlen

Szenario 2 simuliert die Fakultätswahl der Computer und Biowissenschaften Fakultät der Hochschule Mittweida. Bei der letzten Wahl standen sieben Kandidaten zur Auswahl, weshalb $ballot\_size = 7$ gewählt wird. Die genaue Größe der Fakultät Computer- und Biowissenschaften ist nicht bekannt, weshalb $N\_BALLOTS$ geschätzt und auf 1000 gesetzt wird. Jeder Voter darf, wie auch in Szenario 1, genau eine Stimme ($n\_votes = 1$) abgeben. Die generierten gültigen und ungültigen Wahlzettel unterscheiden sich im Vergleich zu den in Szenario 1 beschriebenen Wahlzetteln (siehe Abbildung 5.3 und 5.4) nur in der Anzahl der Kandidaten.

Die Wahlergebnisse werden in Abbildung 5.6 präsentiert. Auffällig in dieser Simulation ist, dass zwei Kandidaten die Wahl gewinnen.

```
Vote: 960 is a Invalid vote-ballot and will not be counted!
Vote: 797 is a Invalid vote-ballot and will not be counted!
Vote: 790 is a Invalid vote-ballot and will not be counted!
Vote: 668 is a Invalid vote-ballot and will not be counted!
Vote: 632 is a Invalid vote-ballot and will not be counted!
Vote: 435 is a Invalid vote-ballot and will not be counted!
Vote: 215 is a Invalid vote-ballot and will not be counted!
Vote: 115 is a Invalid vote-ballot and will not be counted!
Finale Vote Result:
Number 1 with 145  to vote Mr / Mrs ***
Number 2 with 127  to vote Mr / Mrs ***
Number 3 with 147  to vote Mr / Mrs ***
Number 4 with 146  to vote Mr / Mrs ***
Number 5 with 147  to vote Mr / Mrs ***
Number 6 with 145  to vote Mr / Mrs ***
Number 7 with 135  to vote Mr / Mrs ***
Needed Time = 11.0392 Seconds
The election-winners are: Number 3, 5 !
with 147 votes
Feedback Vote-Controle:
Number of Ballots: 1000
Correct Ballots: 992
Invalid Ballots: 8
```

Abbildung 5.6: Szenario 2: Überblick Wahlergebnis

Die Zeitmessung erfolgt genauso wie in Szenario 1. Für 1000 Wahlzettel werden durchschnittlich 10,3 Sekunden für die Durchführung der Wahl benötigt. Ohne Validierung und ungültige Wahlstimmen benötigt die Wahlsimulation durchschnittlich rund 0,0234 Sekunden. Nach den ersten zwei Szenarien ist ein lineares Wachstum der Simulationsdauer im Verhältnis zur Wahlzettelanzahl zu betrachten.

5.3.3 Szenario 3 - Oberbürgermeisterwahl Mittweida

Die Oberbürgermeisterwahl in der Hochschulstadt Mittweida stellt das dritte Szenario dar. Mittweida ist eine kleine Kreisstadt mit rund 15 000 Einwohnern. Zur Oberbürgermeisterwahl 2022 gab es 11 743 wahlberechtigte Personen, deshalb wird $N\_BALLOTS$ gemäß der Größenordnung und auf den Wert 10 000 gesetzt. Da nur ein Wahlkandidat gewinnen kann, ist $n\_votes = 1$. In [51] wurden sechs Kandidaten, die zur Wahl aufgestellt wurden, erwähnt, weshalb $ballot\_size = 6$ festgelegt wurde. In Abbildung 5.7 werden die simulierten Wahlergebnisse vorgestellt. Die Ausgabe der Nummern ungültiger Wahlzettel wurde aufgrund des Umfangs nicht mit in die Abbildung gedruckt. Die Zeitmessung von Szenario 3 verdeutlicht, dass ein linearer Anstieg der Simulationsdauer im Vergleich zu Szenario 1 und Szenario 2 vorliegt, was die lineare Komplexität des implementierten Algorithmus in Abhängigkeit von der Anzahl der Wahlzettel zeigt. Der Mittelwert von 10 Simulationen mit Verifikation und ungültigen Wahlzetteln beträgt rund 103 Sekunden, der Mittelwert von 10 Simulationen ohne Verifikation und ungültigen Wahlzetteln beträgt rund 0,3 Sekunden.

```
Finale Vote Result:
Number 1 with 1617  to vote Mr / Mrs ***
Number 2 with 1680  to vote Mr / Mrs ***
Number 3 with 1689  to vote Mr / Mrs ***
Number 4 with 1678  to vote Mr / Mrs ***
Number 5 with 1670  to vote Mr / Mrs ***
Number 6 with 1578  to vote Mr / Mrs ***
Needed Time = 110.972 Seconds
The election-winners are: Number 3 !
with 1689 votes
Feedback Vote-Controle:
Number of Ballots: 10000
Correct Ballots: 9912
Invalid Ballots: 88
```

Abbildung 5.7: Szenario 3: Überblick Wahlergebnis

5.3.4 Szenario 4 - Kreistagswahl

In Szenario 4 wird eine Kreistagswahl in Nordsachsen simuliert. Zur letzten Kreistagswahl 2019 beteiligten sich von den rund 160 000 Wahlberechtigten ca. 100 000 Wähler an der Wahl in Nordsachsen. Deshalb wird ballot_N = 100 000 gewählt. In sächsischen Kreistageswahlen darf jeder Wähler 3 Stimmen abgeben ($n\_votes = 3$). Die Stimmen dürfen entweder auf 3 unterschiedliche Bewerber, oder in Summe auf einen Bewerber verteilt werden. Die Wahlzettelgeneration kann aber nur die Werte 0 und 1 generieren, weshalb es in der Simulation nicht möglich ist, dass ein Kandidat mehr als eine Stimme von einem Wähler bekommen kann. Insgesamt bewarben sich 2019 in der Kreistagswahl in Nordsachsen 1132 Kandidaten (908 Männer und 224 Frauen), was $ballot\_size = 1132$ ergibt [52]. Der Durchlauf des originalen Simulationsprogramms war aufgrund von begrenzter Hardware nicht möglich. Deshalb wurde das Simulationsprogramm umgeschrieben und der abgeänderte Code ist unter Anhang B dargestellt. Die Abänderung des Programmcodes beinhaltet die Entfernung des Zwischenspeicherns im LIFO-Stapel. Der Validierungsprozess wird in der überarbeiteten Main-Methode unmittelbar nach der Generierung des Wahlzettels durchgeführt. Das simulierte Wahlergebnis wird in Abbildung 5.8 dargestellt. Die durchschnittliche Simulationsdauer (bei Simulationsdurchläufen) beträgt 1140 Sekunden. Die gemessene Simulationsdauer weicht von dem linearen Anstieg der Simulationsdauer der drei vorherigen Szenarien ab. Demnach beträgt die zu erwartende Simulationsdauer rund 1030 Sekunden. Ursache für die Differenz der beiden Simulationszeiten, kann die Abänderung des Simulationsprogramms sein.

```
Number 1129 with 288  to vote Political Party / Person ***
Number 1130 with 251  to vote Political Party / Person ***
Number 1131 with 255  to vote Political Party / Person ***
Number 1132 with 250  to vote Political Party / Person ***
Needed Time = 1141.77 Seconds
The election-winners are: Number 35 !
with 316 votes
Feedback Vote-Controle:
Number of Ballots: 100000
Correct Ballots: 98990
Invalid Ballots: 1010
```

Abbildung 5.8: Szenario 4: Überblick Wahlergebnis

5.3.5 Szenario 5 - Landtagswahl

Szenario 5 steht repräsentativ für eine Onlinewahl im Umfang der sächsischen Landtagswahl. Bei der Landtagswahl 2019 wurden 2 188 486 Erst- und Zweitstimmen abgegeben. Um dieses Szenario besser abbilden zu können, wird die Stimmenanzahl auf 2 Millionen Stimmen gerundet. Demzufolge wird ballot_N = 2 000 000 und $n\_votes = 2$ gesetzt. Insgesamt wurden 22 Parteien gewählt, weshalb $ballot\_size = 22$ gewählt wird [53]. Normalerweise müsste noch die Anzahl der zur Wahl stehenden Kandidaten berücksichtigt werden, aber die ist in jedem Landkreis des Bundeslandes Sachsen unterschiedlich und wird deshalb nicht berücksichtigt.

Die Erst- und Zweitstimme werden in einem Wahlzettel repräsentiert. Dadurch lassen sich beide Stimmen in der aktuellen Version des proof-of-concept Programms nicht getrennt voneinander auswerten. In dieser Simulation hat jeder Voter die Möglichkeit, zwei Stimmen auf zwei unterschiedliche Parteien, oder kandidierende Personen zu verteilen. In einer realen Wahl muss eine strikte Unterteilung zwischen Partei und Kandidaten sichergestellt werden. In dieser Simulation wurde keine Unterscheidung zwischen Person und Partei unternommen. Der

Durchlauf des originalen Simulationsprogramm war auch in Szenario 5 nicht möglich. Auch in diesem Szenario wurde das abgeänderte Simulationsprogramm aus Anhang B verwendet. Im Falle einer realen Wahl wäre das Zwischenspeichern der verschlüsselten Online-Stimme aber auf jeden Fall von Vorteil, da die Stimmen erst alle gesammelt und dann in einem Prozess validiert werden können. Um dies in einer realen Wahl zu gewährleisten, muss leistungsstärkere Hardware zum Einsatz kommen. Weiterhin empfiehlt sich das Nutzen von Parallelisierung (z.B. Aufteilung der Wahl in Wahlkreise und gleichzeitige Validierung und Auszählung der Stimmen). Die abgeänderte Simulation dauert 22 447,5 Sekunden, das entspricht rund 6 Stunden. Die Wahlergebnisse werden in Abbildung 5.9 präsentiert.

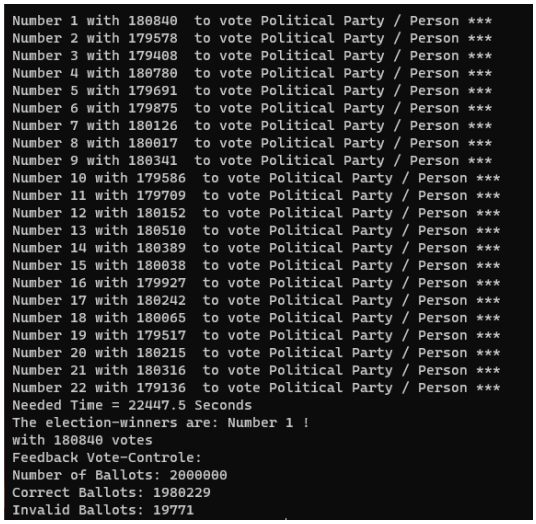

Abbildung 5.9: Szenario 5: Überblick Wahlergebnis

5.3.6 Szenario 6 - Deutsche Bundestagswahl

Szenario 6 ist repräsentativ für eine Onlinewahl im Umfang der Bundestagswahl. Zur letzten Bundestagswahl waren 61 181 072 Wahlberechtigte zur Wahl zugelassen. ballot_N wird gerundet und bekommt den Wert 62 000 000 zugeschrieben. Jeder Voter hat $n\_votes = 2$ Stimmen. In der Statistik der Bundestagswahl werden 47 Parteien näher erwähnt. Es besteht aber wie in Szenario 5 auch bei der Bundestagswahl das Problem, dass die Kandidatenanzahl je nach Bundesland und Landkreis variiert. Deshalb wird in diesem Beispiel von $ballot\_size = 47$ Parteien ausgegangen [54].

Aufgrund der sich linear verhaltenden Simulationsdauer und der großen Anzahl an Wahlstimmen, wurde darauf verzichtet Szenario 5 praktisch durchzuführen. Statt dessen wird die Laufzeit zuerst berechnet. Wird von einem linearen Wachstum der Simulationsdauer ausgegangen, wie es in Simulation 1 bis Simulation 3 zu sehen ist, so ist mit einer Laufzeit von mindestens 638 600 Sekunden, also rund 1774 Stunden auf einem System mit einem Prozessor zu rechnen. Weitere Optimierungen oder eine Parallelisierung würden die lange Simulationsdauer verkürzen. Grundsätzlich empfiehlt es sich daher, den Wahlvorgang auf die Bundesländer oder noch kleinteiliger aufzuteilen, sodass die Arbeitslast auf mehrere Server aufgeteilt wird. Auf Bundestagsebene könnte dies bedeuten, dass z.B. jedes Bundesland

eigene Wahlurnen beaufsichtigt. Nach der Auszählung der Stimmen könnten die Wahlergebnisse der einzelnen Bundesländer dann addiert werden um dadurch die Evaluations- und Validierungsphase der Wahl zu verkürzen.

5.3.7 Fazit der Simulation

Die durchgeführte Simulation veranschaulicht einen Teilprozess einer Onlinewahl. Sowohl Wahlurne als auch Online-Stimmen wurden homomorph verschlüsselt. Es ist zu sehen, dass durch die homomorphe Verschlüsselung die Validierung und Auswertung der Online-Stimmen ohne vorherige Entschlüsselung ermöglicht wird. Dadurch wird die Trennung der Identitätsdaten von dem abgegebenen Wählervotum ermöglicht. Negativ anzumerken ist die niedrige Leistungsfähigkeit der Simulation. Die Simulation ist auf verhältnismäßig niedrige Anzahl an Wahlstimmen begrenzt. Um die Leistungsfähigkeit zu steigern, sollten generierte Wahlzettel nicht im Arbeitsspeicher (RAM) sondern auf einem Datenträger gespeichert werden. Außerdem ist die Verwendung leistungsfähigerer Hardware empfehlenswert.

Insgesamt wird durch die Simulation veranschaulicht, dass die Teilprozesse der Online-Stimmenaufbewahrung in der elektronischen Wahlurne und die verschlüsselte Wahlergebnisermittlung sicher mit homomorphen Verschlüsselungen umzusetzen sind. Lediglich die Leistungsfähigkeit ist ausbaubar.

6 Zusammenfassung und Ausblick

Die Aufgabe der Arbeit, einen Überblick über die weitläufige Thematik der homomorphen Verschlüsselungen zu erstellen, wurde im Grundlagenteil gegeben. Neben der vollständig homomorphen Verschlüsselung findet vor allem die Leveled Homomorphic Encryption sehr viele Anwendungsmöglichkeiten. Auf eine Auswahl an Anwendungsgebieten wird in Abschnitt 5.1 eingegangen. Das Ziel dabei ist es, die Vorteile der homomorphen Verschlüsselungen, im Vergleich zu herkömmlichen Verschlüsselungen, an Anwendungsbeispielen zu verdeutlichen. Es ist erkennbar, dass homomorphe Verschlüsselungen eine höhere Sicherheit bei cloudbasierten Anwendungen bieten. Dadurch ist es Unternehmen z.B. möglich, Aufgaben zu outsourcen und dabei Datenschutz und Sicherheit der ausgelagerten Daten zu gewährleisten. Die Arbeit soll aber auch Nachteile homomorphe Verschlüsselungen darstellen. Es ist erkennbar, dass homomorphe Verschlüsselungen eine höhere Komplexität besitzen und dadurch die Implementierung der Verschlüsselungen erschwert wird. Weiterhin führt eine große Anzahl an Operationen, die auf die Verschlüsselung ausgeführt werden (wie z.B. unter Abschnitt 5.3.6) zu langen Rechenzeiten.

Homomorphe Verschlüsselungen werden durch eine Vielzahl an Bibliotheken zur Verfügung gestellt. Neben Zama, Helib, HEAAN, Palisade und vielen weiteren Bibliotheken stellen die Microsoft SEAL Bibliothek und die OpenFHE Bibliothek eine große Anzahl an Schemata zur homomorphen Verschlüsselungen bereit. Im Benchmarking-Teil (Abschnitt 4) wurden die Microsoft SEAL und die OpenFHE Bibliothek auf ihre Eigenschaften und Leistungsfähigkeit miteinander verglichen. Die Microsoft SEAL Bibliothek ist der klare Sieger dieses Vergleiches, denn obwohl sie weniger Methoden und weniger Notationen bereitstellt, gewinnt sie jeden Benchmarking Test gegen die OpenFHE Bibliothek. Die Benchmarking-Tests wurden im BFV- und CKKS-Schema durchgeführt und geben Auskunft über Dauer von Addition und Multiplikation auf ganzen und reellen Zahlen.

Mit dem Sieger des Benchmarkings wurde in Abschnitt 5 ein Usecase für Onlinewahlen implementiert. Ziel dabei war es, ein spezielles Anwendungsgebiet zu benennen und einen Überblick über die Anwendbarkeit von homomorphen Verschlüsselungen zu geben. Weiterhin wird Bezug auf eine aktuelle Richtlinie des BSI genommen, um darzustellen, wo genau bei Onlinewahlen homomorphe Verschlüsselungen zum Einsatz kommen können. Homomorphe Verschlüsselungen eignen sich insbesondere für die Verschlüsselung der digitalen Wahlurne. Aufgrund der speziellen Verschlüsselung ist es möglich, verschlüsselte Wahlzettel auf die verschlüsselte Wahlurne zu addieren. Dadurch kann der komplette Stimmenauswertungsprozess in einer verschlüsselten Umgebung erfolgen und der Wahlgrundsatz der geheimen Wahl ist geschützt. Auch der Validierungsprozess ist sehr gut mithilfe von Rotationen homomorpher Verschlüsselungen durchführbar (Abschnitt 5.2).

Zusammenfassend ist zu sagen, dass homomorphe Verschlüsselungen wie z.B. Microsoft SEAL eine sichere Nutzung von cloudbasierten Anwendungen versprechen und dieses Versprechen auch einhalten können. Sie sind damit ein bedeutender Teil der modernen Kryptographie, und eine zukünftig wachsende Anwendung dieses Verfahrens ist nicht auszuschließen. Weiterhin wird es in der Zukunft möglich sein, einzelne Prozesse und Methoden zu verbessern, sodass

mit leistungsfähigeren und schnelleren Homomorphen Verschlüsselungen zu rechnen ist. Auch neue Ansätze, wie eine Noise-freie, vollständig homomorphe Verschlüsselung (Quelle [55]), geben einen Ausblick auf eine sicherere Zukunft.

Anhang A: Programmcode Online-Wahl

```
/*
USECASE:
Online election with homomorphic encryption.
In this example, ballot papers are generated randomly.
The voter has to decide between "ballot_size" candidates.
The ballot_size could be every Integer between 0 and PolyModulusDegree/2.
A correct ballot consists of PolyModulusDegree/2-1 times Value "0" and one Value
"1". The value "1" represents the vote decision. Invalid ballots consists values
between -100 and 100. In this example 99% of generated ballot papers are correct
and only 1% is invalid.


VALIDITY:
To check if a ballot is correct [only 0,1, and sum == 1].

ALGORITHM:
To check if a ballot is correct every value of ballot
will be added and also squared and added. The two results obtained have to be the
value "1" to be correct. The whole check procedure takes place in function
check_vote. The funktion print_result outputs the vote_result, number of invalid
and correct ballots and the measured time the whole process needed (ballot
generation and print_result not included).
*/
#include "examples.h"
#include <algorithm>
#include <ctime>
#include <random>
#include <stack>

using namespace std;
using namespace seal;
// globale variables
int ballot_size = 12; // number of candidates
int n_ballots = 100;    // number of generated ballots
int n_votes =1; // number of electoral votes

/// Function to check the ballot
int check_vote(Ciphertext encrypted_ballot_stacked, seal::Evaluator &evaluator,
               seal::Decryptor &decryptor, seal::BatchEncoder &batch_encoder,
               seal::GaloisKeys &galois_keys, seal::RelinKeys &relin_keys)
{
  // define needed Variables
  Plaintext p_check;
  vector<uint64_t> v_check;
  int checkpointer;
  auto c_votes = encrypted_ballot_stacked;
  auto c_squares = c_votes; // deep copy
  // rotate ballot to the second row of the Ciphertext Matrix
  evaluator.rotate_columns_inplace(c_squares, galois_keys);
  // square the (duplicated) ballot in the second row component-wise
  evaluator.square(c_squares, c_squares);
  // relinearize the product
  evaluator.relinearize_inplace(c_squares, relin_keys);
  // add the (squared) second row of the Ciphertext to the first row in-place
```

```
55    evaluator.add(c_votes, c_squares, c_votes);
56
57    // rotate the Ciphertext left and add to the previous version to check the
58    // correctness of the ballot by accumulating (encrypted) sums for 0 resp. 1.
59    int shift = 1;
60    for (int i = 0; i < 11; i++)
61    {
62      auto copy = c_votes;                                          // deep copy
63      evaluator.rotate_rows_inplace(c_votes, shift, galois_keys); // shift == pow
        (2,i)
64      evaluator.add(c_votes, copy, c_votes);
65      shift *= 2;
66    }
67    // decrypt and decode the result
68    decryptor.decrypt(c_votes, p_check);
69    batch_encoder.decode(p_check, v_check);
70
71    // set the checkpointer -> two points (each of each row) are needed
72    if (v_check[0] && v_check[4095] <= n_votes) {
73      checkpointer = 1;
74    }
75    else
76    {
77      checkpointer = 0;
78    }
79    return checkpointer;
80  }
81
82  /// Function to output the results of the election
83  void print_result(vector<uint64_t> vote_result, double time,
84                    int n_invalid_votes)
85  {
86    // list of votes for each candidate
87    std::cout << "Finale Vote Result: " << std::endl;
88    for (int i = 0; i < ballot_size; i++)
89    {
90      std::cout << "Number " << i + 1 << " with " << vote_result[i]
91                << "  to vote Political Party / Person *** " << std::endl;
92    }
93    // output measured Time
94    std::cout << "Needed Time = " << time / CLOCKS_PER_SEC << " Seconds"
95              << std::endl;
96
97    // select the election winner(s)
98    int most_votes = vote_result[0];
99    vector<int> election_winners;
100   election_winners.push_back(0 + 1);
101
102   for (int i = 1; i < vote_result.size(); i++)
103   {
104     if (vote_result[i] == most_votes)
105     {
106       election_winners.push_back(i + 1);
107     }
108     if (vote_result[i] > most_votes)
109     {
110       election_winners.clear();
```

```
111       most_votes = vote_result[i];
112       election_winners.push_back(i + 1);
113     }
114   }
115   // output election winner(s)
116   std::cout << "The election-winners are: Number ";
117   for (size_t i = 0; i < election_winners.size(); i++)
118   {
119     std::cout << election_winners[i];
120     if (i != election_winners.size() - 1)
121     {
122       std::cout << ", ";
123     }
124   }
125   std::cout << " !" << std::endl;
126   std::cout << "with " << most_votes << " votes" << std::endl;
127
128   // counted_votes
129   int counted_votes = 0;
130   for (int i = 0; i < vote_result.size(); i++)
131   {
132     counted_votes += vote_result[i];
133   }
134   // output correct and invalid votes
135   std::cout << "Feedback Vote-Controle: " << std::endl;
136   std::cout << "Number of Ballots: " << (counted_votes + n_invalid_votes*n_votes)
      /n_votes
137             << std::endl;
138   std::cout << "Correct Ballots: " << counted_votes/n_votes << std::endl;
139   std::cout << "Invalid Ballots: " << n_invalid_votes << std::endl;
140 }
141 int main()
142 {
143
144   // Parameter
145   // set scheme type
146   EncryptionParameters parms(scheme_type::bfv);
147   // set PolyModulusDegree
148   size_t poly_modulus_degree = 4096;
149   parms.set_poly_modulus_degree(poly_modulus_degree);
150   // set Coefficient Modulus
151   parms.set_coeff_modulus(CoeffModulus::BFVDefault(poly_modulus_degree));
152   // set Plaintext Modulus
153   parms.set_plain_modulus(PlainModulus::Batching(
154       poly_modulus_degree,
155       20));
156   // set Context
157   SEALContext context(parms);
158   // output parameters
159   print_parameters(context);
160
161   // Key-Generation
162   KeyGenerator keygen(context);
163   SecretKey secret_key = keygen.secret_key();
164   PublicKey public_key;
165   keygen.create_public_key(public_key);
166   RelinKeys relin_keys;
```

```
167  keygen.create_relin_keys(relin_keys);
168  Encryptor encryptor(context, public_key);
169  Evaluator evaluator(context);
170  Decryptor decryptor(context, secret_key);
171  GaloisKeys galois_keys;
172  keygen.create_galois_keys(galois_keys);
173
174  // Encoding
175  BatchEncoder batch_encoder(context);
176  size_t slot_count = batch_encoder.slot_count();
177  size_t row_size = slot_count / 2;
178  // create new vector for counting votes
179  vector<uint64_t> vote_counter(slot_count, 0ULL);
180  // create new vector for election result
181  vector<uint64_t> vote_result;
182  // create counter for invalid votes
183  int n_invalid_votes;
184  // output vector -> all values have to be 0
185  cout << "Startmatrix:" << endl;
186  print_matrix(vote_counter, row_size);
187
188  // create Plaintext and Ciphertext
189  Plaintext plain_vote_counter;
190  Plaintext plain_ballot;
191  Ciphertext encrypted_vote_counter;
192  Ciphertext encrypted_ballot;
193
194  // encode Vector to Batched-Matrix
195  batch_encoder.encode(vote_counter, plain_vote_counter);
196  // encrypt Batched Matrix to Cyphertext
197  encryptor.encrypt(plain_vote_counter, encrypted_vote_counter);
198
199  // create variable for time-measurement
200  double time = 0.0, tstart;
201  // create stack for encrypted ballots
202  std::stack<Ciphertext> stacked_ballots;
203
204  //create random ballot paper
205  for (int vote_id=0; vote_id < n_ballots; vote_id++) {
206    vector<uint64_t> ballot(ballot_size);
207    std::random_device rd;
208    std::mt19937 z(rd());
209    std::uniform_real_distribution<double> dist(0.0, 1.0);
210    if (dist(z) <= 0.99) {
211      std::fill(ballot.begin(), ballot.begin() + n_votes, 1);
212      std::fill(ballot.begin() + n_votes, ballot.end(), 0);
213      std::shuffle(ballot.begin(), ballot.end(), z);
214    }
215    else
216    {
217      std::uniform_real_distribution<double> d(-100, 100);
218      for (int i = 0; i < ballot_size; ++i)
219      {
220        ballot[i] = d(rd);
221      }
222    }
223
```

```
    ///* //Print Plain Votes for Testcases
    //In case of a real election, the ballots have to be encrypted immediately
    cout << "Plain-vote " << vote_id +1 << ":" << endl;
    for (int num : ballot) {
      std::cout << num << " ";
    }
    std::cout << std::endl;
    //*/
    // ballot encoding
    batch_encoder.encode(ballot, plain_ballot);
    // ballot encryption
    encryptor.encrypt(plain_ballot, encrypted_ballot);
    // put ballot on a LIFO-stack
    stacked_ballots.push(encrypted_ballot);
  }
  // start time-measurement
  tstart = clock();
  for (int vote_id; vote_id < n_ballots; vote_id++) {
    // take encrypted ballot from LIFO-stack
    Ciphertext encrypted_ballot_stacked = stacked_ballots.top();
    // use check_vote function to check if the encrypted ballot is invalid
    int checkpointer =
        check_vote(encrypted_ballot_stacked, evaluator, decryptor,
                   batch_encoder, galois_keys, relin_keys);
    // selection correct ballots
    if (checkpointer == 1)
    {
      evaluator.add(encrypted_vote_counter, encrypted_ballot_stacked,
                    encrypted_vote_counter);
    }
    // selection invalid ballots
    if (checkpointer == 0) {
      std::cout << "Vote: " << n_ballots - vote_id
                << " is a Invalid vote-ballot and will not be counted!"
                << std::endl;
      n_invalid_votes++;
    }
    // Remove the used ballot on top of the LIFO-stack
    stacked_ballots.pop();
  }
  // Decrypt election results
  decryptor.decrypt(encrypted_vote_counter, plain_vote_counter);
  // Decode election results
  batch_encoder.decode(plain_vote_counter, vote_result);
  // stop time measurement
  time = clock() - tstart;
  // use print_result function to output results
  print_result(vote_result, time, n_invalid_votes);

  return 0;
}

/*
Benchmarking Results:
*With Invalid Votes and Verification (ballot generation and print_result not
included):
->100 Votes: ~1,03 Seconds
```

```
->1000 Votes: ~10,3 Seconds
->10000 Votes: ~103 Seconds
->100000 Votes: ~1140 Seconds (Simulation-Program B)
->1000000 Votes:      Seconds (Simulation-Program B)

*Without Invalid Votes and Verification (ballot generation and print_result not
included):
->100 Votes: ~0,0029 Seconds
->1000 Votes: ~0,0234 Seconds
->10000 Votes: ~0,3 Seconds
->100000 Votes: ~2,8 Seconds
->1000000 Votes:


100 000
*/
```

Anhang B: Programmcode Ergänzung

```
}
int main()
{

  // Parameter
  // set scheme type
  EncryptionParameters parms(scheme_type::bfv);
  // set PolyModulusDegree
  size_t poly_modulus_degree = 4096;
  parms.set_poly_modulus_degree(poly_modulus_degree);
  // set Coefficient Modulus
  parms.set_coeff_modulus(CoeffModulus::BFVDefault(poly_modulus_degree));
  // set Plaintext Modulus
  parms.set_plain_modulus(PlainModulus::Batching(
      poly_modulus_degree,
      20));
  // set Context
  SEALContext context(parms);
  // output parameters
  print_parameters(context);

  // Key-Generation
  KeyGenerator keygen(context);
  SecretKey secret_key = keygen.secret_key();
  PublicKey public_key;
  keygen.create_public_key(public_key);
  RelinKeys relin_keys;
  keygen.create_relin_keys(relin_keys);
  Encryptor encryptor(context, public_key);
  Evaluator evaluator(context);
  Decryptor decryptor(context, secret_key);
  GaloisKeys galois_keys;
  keygen.create_galois_keys(galois_keys);

  // Encoding
  BatchEncoder batch_encoder(context);
  size_t slot_count = batch_encoder.slot_count();
  size_t row_size = slot_count / 2;
  // create new vector for counting votes
  vector<uint64_t> vote_counter(slot_count, 0ULL);
  // create new vector for election result
  vector<uint64_t> vote_result;
  // create counter for invalid votes
  int n_invalid_votes;
  // output vector -> all values have to be 0
  cout << "Startmatrix:" << endl;
  print_matrix(vote_counter, row_size);

  // create Plaintext and Ciphertext
  Plaintext plain_vote_counter;
  Plaintext plain_ballot;
  Ciphertext encrypted_vote_counter;
  Ciphertext encrypted_ballot;

```

```
55  // encode Vector to Batched-Matrix
56  batch_encoder.encode(vote_counter, plain_vote_counter);
57  // encrypt Batched Matrix to Cyphertext
58  encryptor.encrypt(plain_vote_counter, encrypted_vote_counter);
59
60  // create variable for time-measurement
61  double time = 0.0, tstart;
62  // create stack for encrypted ballots
63  std::stack<Ciphertext> stacked_ballots;
64
65  //create random ballot paper
66  tstart = clock();
67  for (int vote_id=0; vote_id < n_ballots; vote_id++) {
68    vector<uint64_t> ballot(ballot_size);
69    std::random_device rd;
70    std::mt19937 z(rd());
71    std::uniform_real_distribution<double> dist(0.0, 1.0);
72    if (dist(z) <= 0.99) {
73      std::fill(ballot.begin(), ballot.begin() + n_votes, 1);
74      std::fill(ballot.begin() + n_votes, ballot.end(), 0);
75      std::shuffle(ballot.begin(), ballot.end(), z);
76    }
77    else
78    {
79      std::uniform_real_distribution<double> d(-100, 100);
80      for (int i = 0; i < ballot_size; ++i)
81      {
82        ballot[i] = d(rd);
83      }
84    }
85
86    // ballot encoding
87    batch_encoder.encode(ballot, plain_ballot);
88    // ballot encryption
89    encryptor.encrypt(plain_ballot, encrypted_ballot);
90
91  // start time-measurement
92
93  Plaintext p_check;
94  vector<uint64_t> v_check;
95  int checkpointer;
96  auto c_votes = encrypted_ballot;
97  auto c_squares = c_votes; // deep copy
98  // rotate ballot to the second row of the Ciphertext Matrix
99  evaluator.rotate_columns_inplace(c_squares, galois_keys);
100 // square the (duplicated) ballot in the second row component-wise
101 evaluator.square(c_squares, c_squares);
102 // relinearize the product
103 evaluator.relinearize_inplace(c_squares, relin_keys);
104 // add the (squared) second row of the Ciphertext to the first row in-place
105 evaluator.add(c_votes, c_squares, c_votes);
106
107 // rotate the Ciphertext left and add to the previous version to check the
108 // correctness of the ballot by accumulating (encrypted) sums for 0 resp. 1.
109 int shift = 1;
110 for (int i = 0; i < 11; i++)
111 {
```

```
112     auto copy = c_votes;                                          // deep copy
113     evaluator.rotate_rows_inplace(c_votes, shift, galois_keys); // shift == pow
        (2,i)
114     evaluator.add(c_votes, copy, c_votes);
115     shift *= 2;
116   }
117   decryptor.decrypt(c_votes, p_check);
118   batch_encoder.decode(p_check, v_check);
119   // set the checkpointer -> two points (each of each row) are needed
120   if (v_check[0] && v_check[4095] <= n_votes) {
121     checkpointer = 1;
122   }
123   else
124   {
125     checkpointer = 0;
126   }
127     // selection correct ballots
128     if (checkpointer == 1)
129     {
130       evaluator.add(encrypted_vote_counter, encrypted_ballot,
131                     encrypted_vote_counter);
132     }
133     // selection invalid ballots
134     if (checkpointer == 0) {
135       std::cout << "Vote: " << n_ballots - vote_id
136                 << " is a Invalid vote-ballot and will not be counted!"
137                 << std::endl;
138       n_invalid_votes++;
139     }
140   }
141   // Decrypt election results
142   decryptor.decrypt(encrypted_vote_counter, plain_vote_counter);
143   // Decode election results
144   batch_encoder.decode(plain_vote_counter, vote_result);
145   // stop time measurement
146   time = clock() - tstart;
147   // use print_result function to output results
148   print_result(vote_result, time, n_invalid_votes);
149
150   return 0;
151 }
```

Literaturverzeichnis

[1] V. Campos, „Homomorphe Verschlüsselungen“, Bachelorarbeit, 17. Aug. 2014, S. 66. Adresse: https://violacampos.github.io/assets/thesis.pdf.

[2] www.cryptool.org und CrypTool-Projekt, *Skytale-Verschlüsselung (Permutationsverschlüsselung)*, Juni 2008. Adresse: https://commons.wikimedia.org/wiki/File:Skytale3d_de.png (besucht am 09. 07. 2023).

[3] BSI. „Verschlüsselt kommunizieren im Internet“, Bundesamt für Sicherheit in der Informationstechnik. (), Adresse: https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Informationen-und-Empfehlungen/Onlinekommunikation/Verschluesselt-kommunizieren/verschluesselt-kommunizieren.html?nn=131792 (besucht am 30. 05. 2023).

[4] N. A. Zbick, „Homomorphe Verschlüsselung“, Technische Universität Dortmund, Dortmund, Seminararbeit, 28. Juni 2012, S. 27. Adresse: https://rgse.uni-koblenz.de/web/pages/teaching/ss12/mbse-sem/pub/Ausarbeitung_Zbick.pdf.

[5] BSI. „Cloud Computing Grundlagen“, Bundesamt für Sicherheit in der Informationstechnik. (), Adresse: https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Empfehlungen-nach-Angriffszielen/Cloud-Computing/Grundlagen/grundlagen.html?nn=128880 (besucht am 31. 05. 2023).

[6] P. A. Höher, „Kryptologie“, in *Grundlagen der digitalen Informationsübertragung: Von der Theorie zu Mobilfunkanwendungen*, P. A. Höher, Hrsg., Wiesbaden: Springer Fachmedien, 2013, S. 143–154, ISBN: 978-3-8348-2214-7. DOI: 10.1007/978-3-8348-2214-7_7. Adresse: https://doi.org/10.1007/978-3-8348-2214-7_7 (besucht am 10. 07. 2023).

[7] G. Teschl und S. Teschl, *Mathematik für Informatiker: Band 1: Diskrete Mathematik und Lineare Algebra*. Springer-Verlag, 22. Juni 2013, 524 S., Google-Books-ID: ejkiBAAAQBAJ, ISBN: 978-3-642-37972-7.

[8] C. Neuhaus, *Cloud security mechanisms* (Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam 87), A. Polze, Hrsg. Potsdam: Univ.-Verl, 2014, 78 S., ISBN: 978-3-86956-281-0.

[9] C. Baxa. „Algebra_1_9.pdf“, Universität Wien - Vorlesung Algebra 1. (), Adresse: https://www.mat.univie.ac.at/~baxa/Algebra_1_9.pdf (besucht am 10. 07. 2023).

[10] R. L. Rivest, L. Adleman und M. L. Dertouzos. „On data banks and privacy homomorphisms“. (1978), Adresse: https://luca-giuzzi.unibs.it/corsi/Support/papers-cryptography/RAD78.pdf (besucht am 31. 05. 2023).

[11] A. Acar, H. Aksu, A. S. Uluagac und M. Conti, „A survey on homomorphic encryption schemes: Theory and implementation“, *ACM Computing Surveys*, Jg. 51, Nr. 4, S. 1–35, 31. Juli 2019, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3214303. Adresse: https://dl.acm.org/doi/10.1145/3214303 (besucht am 04. 07. 2023).

[12] M. S. M. Brenner, „Rechnen mit verschlüsselten Programmen und Daten“, Diss., Gottfried Wilhelm Leibniz Universität Hannover, Neustadt a. Rbge., 7. Dez. 2012, 211 S.

[13] J. Fan und F. Vercauteren, *Somewhat Practical Fully Homomorphic Encryption*. Adresse: https://eprint.iacr.org/undefined/undefined (besucht am 04. 07. 2023).

[14] X. Yi, R. Paulet und E. Bertino, *Homomorphic Encryption and Applications* (SpringerBriefs in Computer Science). Cham: Springer International Publishing, 2014, ISBN: 978-3-319-12228-1 978-3-319-12229-8. DOI: 10.1007/978-3-319-12229-8. Adresse: https://link.springer.com/10.1007/978-3-319-12229-8 (besucht am 31. 05. 2023).

[15] Z. Brakerski, C. Gentry und V. Vaikuntanathan, „(Leveled) Fully Homomorphic Encryption without Bootstrapping“, *ACM Transactions on Computation Theory*, Jg. 6, Nr. 3, 13:1–13:36, 1. Juli 2014, ISSN: 1942-3454. DOI: 10.1145/2633600. Adresse: https://doi.org/10.1145/2633600 (besucht am 04. 07. 2023).

[16] C. Gentry, „A fully homomorphic encryption scheme“, Diss., Stanford University, 2009.

[17] M. R. Albrecht, R. Player und S. Scott, „On the concrete hardness of learning with errors“, *Journal of Mathematical Cryptology*, Jg. 9, Nr. 3, S. 169–203, 1. Okt. 2015, Publisher: De Gruyter, ISSN: 1862-2984. DOI: 10.1515/jmc-2015-0016. Adresse: https://www.degruyter.com/document/doi/10.1515/jmc-2015-0016/html (besucht am 06. 07. 2023).

[18] O. Regev, „The learning with errors problem“, Adresse: https://cims.nyu.edu/~regev/papers/lwesurvey.pdf (besucht am 20. 08. 2023).

[19] G. Đorđević, „Performance comparison of homomorphic encryption scheme implementations“, S. 6, Adresse: https://www.etran.rs/2021/zbornik/Papers/104_RTI_2.5.pdf (besucht am 20. 08. 2023).

[20] *Microsoft SEAL*, original-date: 2018-11-09T00:33:14Z. Adresse: https://github.com/microsoft/SEAL (besucht am 31. 05. 2023).

[21] huelse, *Huelse/SEAL-Python*, original-date: 2019-07-15T09:16:52Z. Adresse: https://github.com/Huelse/SEAL-Python (besucht am 31. 05. 2023).

[22] A. Viand, P. Jattke und A. Hithnawi, „SoK: Fully Homomorphic Encryption Compilers“, in *2021 IEEE Symposium on Security and Privacy (SP)*, Mai 2021, S. 1092–1108. DOI: 10.1109/SP40001.2021.00068. arXiv: 2101.07078[cs]. Adresse: http://arxiv.org/abs/2101.07078 (besucht am 31. 05. 2023).

[23] „OpenFHE.org – OpenFHE – open-source fully homomorphic encryption library“. (), Adresse: https://www.openfhe.org/ (besucht am 31. 05. 2023).

[24] L. Jiang und L. Ju, *FHEBench: Benchmarking Fully Homomorphic Encryption Schemes*, 1. März 2022. arXiv: 2203.00728[cs]. Adresse: http://arxiv.org/abs/2203.00728 (besucht am 31. 05. 2023).

[25] „OpenFHE documentation“. (), Adresse: https://openfhe-development.readthedocs.io/en/latest/ (besucht am 17. 07. 2023).

[26] Z. Brakerski, „Fully homomorphic encryption without modulus switching from classical GapSVP“, in *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini und R. Canetti, Hrsg., Ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2012, S. 868–886, ISBN: 978-3-642-32009-5. DOI: 10.1007/978-3-642-32009-5_50.

[27] H. Chen, K. Laine und R. Player, „Simple encrypted arithmetic library - SEAL v2.1“, in *Financial Cryptography and Data Security*, M. Brenner, K. Rohloff, J. Bonneau u. a., Hrsg., Ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2017, S. 3–18, ISBN: 978-3-319-70278-0. DOI: 10.1007/978-3-319-70278-0_1.

[28] „Introduction to the BFV encryption scheme“. (), Adresse: https://www.inferati.com/blog/fhe-schemes-bfv (besucht am 31. 05. 2023).

[29] J. H. Cheon, A. Kim, M. Kim und Y. Song, *Homomorphic Encryption for Arithmetic of Approximate Numbers*, 2017. Adresse: https://eprint.iacr.org/2016/421.pdf (besucht am 06. 08. 2023).

[30] „Introduction to the CKKS encryption scheme“. (), Adresse: https://www.inferati.com/blog/fhe-schemes-ckks (besucht am 06. 08. 2023).

[31] Microsoft Research, *Introduction to CKKS (Approximate Homomorphic Encryption)*, 7. Jan. 2020. Adresse: https://www.youtube.com/watch?v=iQlgeL64vfo (besucht am 07. 08. 2023).

[32] K. Laine, „Simple encrypted arithmetic library 2.3.“, Nov. 2017. Adresse: https://www.microsoft.com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1.pdf.

[33] Y. Polyakov, K. Rohloff, G. W. Ryan und D. Cousins, „PALISADE lattice cryptography library user manual (v1.11.9)“, S. 51, 2. Dez. 2022. Adresse: https://gitlab.com/palisade/palisade-release/blob/master/doc/palisade_manual.pdf.

[34] A. A. Badawi, J. Bates, F. Bergamaschi u. a., *OpenFHE: Open-Source Fully Homomorphic Encryption Library*, 9. Sep. 2022. Adresse: https://eprint.iacr.org/undefined/undefined (besucht am 08. 08. 2023).

[35] K. M. M. Aung, E. Lim, J. J. Sim, B. H. M. Tan, H. Wang und S. L. Yeo, „Field instruction multiple data“, in *Advances in Cryptology – EUROCRYPT 2022*, O. Dunkelman und S. Dziembowski, Hrsg., Ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2022, S. 611–641, ISBN: 978-3-031-06944-4. DOI: 10.1007/978-3-031-06944-4_21.

[36] M. Chase, H. Chen, J. Ding u. a., „Security of homomorphic encryption“, Jan. 2018. Adresse: https://www.microsoft.com/en-us/research/wp-content/uploads/2018/01/security_homomorphic_encryption_white_paper.pdf.

[37] D. Hrestak und S. Picek, „Homomorphic encryption in the cloud“, in *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Mai 2014, S. 1400–1404. DOI: 10.1109/MIPRO.2014.6859786.

[38] „Homomorphic Encryption Makes Slow But Steady Progress“. (), Adresse: https://pureai.com/articles/2020/07/13/~/media/ECG/pureai/Images/2020/07/homomorphic1.asxh (besucht am 21. 07. 2023).

[39] M. Kim und K. Lauter, „Private genome analysis through homomorphic encryption“, *BMC Medical Informatics and Decision Making*, Jg. 15, Nr. 5, S3, 21. Dez. 2015, ISSN: 1472-6947. DOI: 10.1186/1472-6947-15-S5-S3. Adresse: https://doi.org/10.1186/1472-6947-15-S5-S3 (besucht am 21. 07. 2023).

[40] F. Armknecht, C. Boyd, C. Carr u. a., *A Guide to Fully Homomorphic Encryption*. Adresse: https://eprint.iacr.org/undefined/undefined (besucht am 22. 07. 2023).

[41] W. Ertel, *Grundkurs Künstliche Intelligenz*, 5. Aufl. Springer Vieweg Wiesbaden, 8. Okt. 2021, 423 S., ISBN: 978-3-658-32074-4. Adresse: https://doi.org/10.1007/978-3-658-32075-1.

[42] S. Meftah, B. H. M. Tan, C. F. Mun, K. M. M. Aung, B. Veeravalli und V. Chandrasekhar, „DOReN: Toward Efficient Deep Convolutional Neural Networks with Fully Homomorphic Encryption“, *IEEE Transactions on Information Forensics and Security*, Jg. 16, S. 3740–3752, 2021, Conference Name: IEEE Transactions on Information Forensics and Security, ISSN: 1556-6021. DOI: 10.1109/TIFS.2021.3090959. Adresse: https://ieeexplore.ieee.org/abstract/document/9460962.

[43] „Die fünf Wahlgrundsätze des Grundgesetzes“, Die Bundesregierung informiert | Startseite. (2. Juli 2021), Adresse: https://www.bundesregierung.de/breg-de/themen/bundestagswahl-2021/wahlgrundsaetze-1938242 (besucht am 23. 07. 2023).

[44] „Online-Wahlen“, Bundesamt für Sicherheit in der Informationstechnik. (), Adresse: https://www.bsi.bund.de/DE/Themen/Oeffentliche-Verwaltung/Moderner-Staat/Online-Wahlen/online-wahlen.html?nn=130244 (besucht am 23. 07. 2023).

[45] „Deutscher Bundestag - E-Voting vorerst keine Option bei Bundestags- oder Landtagswahlen“, Deutscher Bundestag. (2022), Adresse: https://www.bundestag.de/dokumente/textarchiv/2022/kw14-pa-fachgespraech-bildung-882928 (besucht am 23. 07. 2023).

[46] „BSI TR-03162 IT-sicherheitstechnische Anforderungen zur Durchführung einer Online-Wahl im Rahmen des Modellprojekts nach § 194a Fünftes Buch Sozialgesetzbuch (Online-Wahl)“, Bundesamt für Sicherheit in der Informationstechnik. (), Adresse: https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03162/tr-03162.html?nn=460776 (besucht am 11. 07. 2023).

[47] C. Paar und J. Pelzl, *Kryptografie verständlich* (eXamen.press). Berlin, Heidelberg: Springer, 2016, ISBN: 978-3-662-49296-3 978-3-662-49297-0. DOI: 10.1007/978-3-662-49297-0. Adresse: http://link.springer.com/10.1007/978-3-662-49297-0 (besucht am 18. 08. 2023).

[48] „Ungültige Stimmabgabe - Die Bundeswahlleiterin“. (), Adresse: https://www.bundeswahlleiterin.de/service/glossar/u/ungueltige-stimmabgabe.html (besucht am 11. 07. 2023).

[49] J. Wolf und R. Krooß, *C von A bis Z*, 3. Aufl. Rheinwerk Computing, 1190 S., Abschnitt 21.3 Stacks nach dem LIFO-(Last-in-First-out-)Prinzip, ISBN: 978-3-8362-1411-7. Adresse: https://openbook.rheinwerk-verlag.de/c_von_a_bis_z/021_c_dyn_datenstrukturen_003.htm (besucht am 16. 07. 2023).

[50] S. Zeitung. „Cyberagentur hinkt Anspruch hinterher: Zu wenige Mitarbeiter“, Süddeutsche.de. (31. Dez. 2021), Adresse: https://www.sueddeutsche.de/service/internet-halle-saale-cyberagentur-hinkt-anspruch-hinterher-zu-wenige-mitarbeiter-dpa.urn-newsml-dpa-com-20090101-211231-99-550322 (besucht am 17. 07. 2023).

[51] S. L. des Freistaates. „Wahlergebnisse 2022 - Wahlen - sachsen.de“. (), Adresse: http://www.wahlen.sachsen.debuergermeisterwahl-wahlergebnisse-2022.php (besucht am 18. 07. 2023).

[52] R. K. und Öffentlichkeitsarbeit. „Kreistagswahlen - 26. Mai 2019 - Wahlen - sachsen.de“. (), Adresse: http://www.wahlen.sachsen.dekreistagswahlen-2019.html (besucht am 18. 08. 2023).

[53] „Sachsen - Die Bundeswahlleiterin“. (), Adresse: https://www.bundeswahlleiterin.de/service/landtagswahlen/land-14.html (besucht am 18. 07. 2023).

[54] „Ergebnisse Deutschland - Die Bundeswahlleiterin“. (), Adresse: https://www.bundeswahlleiterin.de/bundestagswahlen/2021/ergebnisse/bund-99.html (besucht am 19. 07. 2023).

[55] Z. Zheng, F. Liu und K. Tian, *An Unbounded Fully Homomorphic Encryption Scheme Based on Ideal Lattices and Chinese Remainder Theorem*, 27. Jan. 2023. arXiv: 2301.12060[cs,math]. Adresse: http://arxiv.org/abs/2301.12060 (besucht am 10. 08. 2023).

Eidesstattliche Erklärung

Hiermit versichere ich – Hans Daus – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

Halle (Saale), 21. August 2023 | Hans Daus

Ort, Datum | Hans Daus