universitätbonn

Untersuchung und Implementierung ausgewählter homomorpher Hashing-Verfahren für die Integritätsprüfung homomorpher Chiffrate

Mohammad Harun Samim Faiz
2689231

Bachelorarbeit

Erstprüfer: Prof. Dr. Michael Meier
Zweitprüferin: Jun.-Prof. Dr.-Ing. Delphine Reinhardt
Betreuerin: Dipl.-Inform. Saffija Kasem-Madani
Institut für Informatik IV
Arbeitsgruppe für IT-Sicherheit
Rheinische-Friedrich-Wilhelms-Universität Bonn

Selbstständigkeitserklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Bonn, 12. Dezember 2016

Mohammad Harun Samim Faiz

2689231

Zusammenfassung

Die folgende wissenschaftliche Arbeit untersucht, wie mittels homomorpher Hashing-Verfahren eine Integritätsprüfung homomorpher Chiffrate realisiert werden kann. Dabei wird ein homomorphes Kryptosystem betrachtet, welches bei der Berechnung von Summen im Rahmen einer Datenanalyse eingesetzt werden soll. Ausgangspunkt ist das Paillier-Kryptosystem [20]. Es handelt sich um ein additiv-homomorphes Public-Key-Kryptosystem, welches ermöglicht, durch Multiplikation auf den Chiffreraum eine Addition im Klartextraum durchzuführen, ohne die Klartexte selbst zu erfahren. Für die Operationen wird der öffentliche Schlüssel des Paillier-Kryptosystems benötigt. Ziel ist hierbei, Konflikte zwischen Vertraulichkeit und Verfügbarkeit von sensiblen Daten zu lösen.

In einem praktischen Szenario erhält ein Analyst Chiffredaten und einen öffentlichen Schlüssel, um Analyseoperationen auf den Chiffraten durchzuführen. Da er für den weiteren Verlauf der Analyse seine Ergebnisse kennen muss, benötigt er eine Möglichkeit, die Ergebnisse zu entschlüsseln. Dafür werden die Analyseoperationen im Zusammenspiel mit einer Trusted Third Party (Verifzierer) untersucht, welche den privaten Schlüssel zum Entschlüsseln der Chiffrate besitzt. Dabei wurden Schwächen erkannt, die von der Verformbarkeit der Paillier-Chiffrate und des Probabilismus des Kryptosystems ausgehen. Der Verifizierer kann nicht nachvollziehen, welche Chiffrate er entschlüsselt und ob durch die Entschlüsselung sensible Daten aufgedeckt werden. Es wird ein Konzept vorgestellt, welches in Anlehnung an Zero-Knowledge-Beweise [14] Analyseoperationen mittels Paillier für den praktischen Anwendungsfall ermöglichen soll. Dafür wird ein homomorphes Hashing-Verfahren [16] basierend auf RSA [12] verwendet. Das Hashing-Verfahren ermöglicht der Trusted Third Party, die zu entschlüsselnden Chiffrate auf ihre Integrität zu prüfen, da nachvollzogen werden kann, wie dieses Chiffrat entstanden ist. Das gesamte Verfahren wurde zum Schluss implementiert und evaluiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In Zeiten des digitalen Wandels rückt der Begriff Privatsphäre immer mehr in den Vordergrund und spätestens nach den Enthüllungen durch Edward Snowden wurde Privatsphäre zu einem öffentlichen Diskussionsthema. In einem optimalen System werden nur Informationen preisgegeben, die der Halter dieser Informationen preisgeben möchte. Diese sollen auch nur demjenigen zugängig sein, den der Besitzer der Informationen zuvor autorisiert hat. Ein bekanntes Beispiel ist die Verschlüsselung von privaten Emails, die ermöglichen Informationen nur mit dem zuvor autorisierten Kommunikationspartner zu teilen. Gehört man nicht zum Kreis der autorisierten Personen, so sollten dieser Person die geteilten Informationen nur in einem nicht identifizierbaren und verschlüsselten Format vorliegen.

Problematisch wird diese Eigenschaft im Bereich der Analyse. Unternehmen erheben Daten, die zwecksmäßig analysiert werden sollen. Um die darunter befindlichen sensiblen Daten zu schützen, werden Verschlüsselungen angewendet. Dadurch wird sichergestellt, dass die Daten nicht zugänglich sind, jedoch wird die Möglichkeit genommen, diese zu analysieren. Für Unternehmen bedeutet dies, dass sie in einem Konflikt zwischen der Privatsphäre ihrer Nutzer und der Nutzbarkeit ihrer Daten stehen. Es stellt sich die Frage, wie es möglich gemacht werden kann, dass sensible Daten geschützt werden, aber dennoch nutzbar sind.

In der vorliegenden Arbeit wird ein Verschlüsselungssystem vorgestellt, welches die Möglichkeit bieten soll, den Konflikt zwischen der Vertraulichkeit und Verfügbarkeit von Daten aufzulösen. Die Arbeit untersucht, wie dieses System in der Praxis genutzt werden könnte und analysiert bestehende Schwachstellen. Es wird untersucht und implementiert, wie eine Integritätsprüfung homomorpher Chiffrate mittels homomorpher Hashing-Verfahren durchgeführt werden kann. Es ist eine Verknüpfung von verschiedenen wissenschaftlichen Arbeiten, die so bisher in keinem direkten Zusammenhang aufgegriffen wurden, um dem Konflikt der Privatsphäre und Nutzbarkeit (Privacy vs. Utility) nachzugehen. Grundlage dieser Arbeit ist das Paillier-Kryptosystem [20], homomorphe Hashing-Verfahren [16] [12], sowie Zero-Knowledge-Beweise (zero-knowledge proofs) [14].

1.1 Grundlagen

Sowohl im Verschlüsselungssystem als auch in der Implementierung zur Beseitigung der Schwachstellen des Systems wird von sogenannten mathematischen Homomorphie-Eigenschaften argumentiert. Im Folgenden wird diese Eigenschaft mit anderen grundlegenden Begriffen definiert und erläutert.

1.1.1 Verschlüsselungsverfahren/Kryptosystem

Definition 1.1 (Verschlüsselungsverfahren/Kryptosystem). "Ein Tupel $(\mathcal{M},\mathcal{C},\mathcal{K},\mathcal{E},\mathcal{D})$ wird Kryptosystem genannt, falls die folgenden Eigenschaften erfüllt sind:

1. $\mathcal{M} \subseteq \sum_1^*$ ist der **Klartextraum**, also die Menge aller möglichen **Klartexte** (messages).
2. $\mathcal{C} \subseteq \sum_2^*$ ist der **Chiffretextraum**, also die Menge aller möglichen **Chiffrate** (ciphers).
3. $\mathcal{K} \subseteq \sum_3^*$ heißt **Schlüsselraum**. Die Elemente dieser Menge heißen **Schlüssel** (keys).
4. $\mathcal{E} = \{E_k | k \in \mathcal{K}\}$ ist eine Familie von Funktionen $E_k : \mathcal{M} \to \mathcal{C}$. Die Elemente heißen **Verschlüsselungsfunktionen** (encryption).
5. $\mathcal{D} = \{D_k | k \in \mathcal{K}\}$ ist eine Familie von Funktionen $D_k : \mathcal{C} \to \mathcal{M}$. Die Elemente heißen **Entschlüsselungsfunktionen** (decryption).
6. Für jedes $e \in \mathcal{K}$ existiert ein $d \in \mathcal{K}$, sodass für alle $m \in \mathcal{M}$ die Gleichung $D_d(E_e(m)) = m$ gilt." [18]

Es werden zwei Arten von Kryptosystemen unterschieden: symmetrische und asymmetrische Verschlüsselungsverfahren. Beide erfüllen applikationsabhängige Zwecke. Da für diese Arbeit symmetrische Verschlüsselungsverfahren keine relevante Rolle spielen, werden nur asymmetrische Verschlüsselungsverfahren definiert.

Definition 1.2 (Asymmetrisches Verschlüsselungsverfahren/Public-Key-Kryptosystem). Ein asymmetrisches Verschlüsselungsverfahren baut auf drei effizienten Algorithmen auf:

- **Schlüsselgenerierung (1^n).** Der Algorithmus zum Erzeugen eines Schlüssels ist probabilistisch[1] und benötigt als Eingabe einen Sicherheitsparameter 1^n, wodurch als Ausgabe ein Schlüsselpaar generiert wird. Dieser besteht aus einem öffentlichen Schlüssel k und einem korrespondierenden privaten Schlüssel k^{-1}, der mathematisch mit k verknüpft ist.
- **Encrypt(k,m).** Die Verschlüsselungsfunktion kann sowohl deterministisch, als auch probabilistisch sein und erhält als Eingabe den öffentlichen Schlüssel k und einen Klartext m. Die Ausgabe wird dann wie folgt berechnet: $c = Encrypt(k,m)$
- **Decrypt(k^{-1},c).** Die Entschlüsselung ist deterministisch, die mittels des privaten Schlüssels k^{-1} das Chiffrat entschlüsseln kann, sodass der Klartext daraus resultiert: $m = Decrypt(k^{-1},c)$ [19].

Abbildung 1 zeigt welchen Zweck der öffentliche und private Schlüssel besitzt. Der öffentliche Schlüssel wird zum Verschlüsseln und der private Schlüssel zum Entschlüsseln verwendet. Daraus resultiert insbesondere, dass jedes Schlüsselpaar (k,k^{-1}) in Bezug auf ein beliebiges $m \in \mathcal{M}$ stets den inversen Effekt bei der Verwendung von *Encrypt* bzw. *Decrypt* besitzen muss:

$$Decrypt(k^{-1}, Encrypt(k,m)) = m$$

Werden stattdessen zwei nicht zueinander korrespondierende Schlüssel k und k^{-1} verwendet, dann darf bei der Entschlüsselung nicht der korrekte Klartext resultieren [19]. Es müsste theoretisch möglich sein, den privaten Schlüssel aus dem öffentlichen zu berechnen, jedoch werden die Schlüssellängen so gewählt, dass es praktisch unmöglich ist [1] [23]. Die Schlüsselpaargenerierung bauen meist auf Einwegfunktionen mit Falltür-Eigenschaften *(Trapdoor-Einwegfunktionen)* auf [19] [25].

[1] nicht-deterministisch

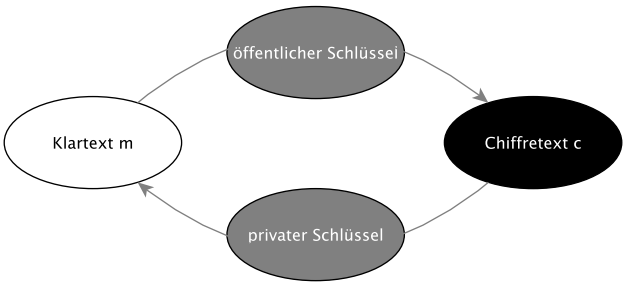

Abbildung 1: *Ver- und Entschlüsselung bei einem asymmetrischen Kryptosystem*

Definition 1.3 (Einwegfunktion). Eine Funktion $f : X \to Y$ ist eine Einwegfunktion, sofern $f(x)$ effizient zu berechnen ist, für alle $x \in X$, wohingegen $f^{-1}(y)$ genau nicht effizient zu berechnen ist [19].

Definition 1.4 (Falltür-Funktion). Eine Einwegfunktion $f : X \to Y$ ist eine Falltür-Funktion, wenn es mit Hilfe einer externen Information möglich ist, f zu invertieren, d.h. $f^{-1}(y)$ effizient zu berechnen [19].

Theorem 1. Existiert eine Falltür-Funktion, dann existiert gleichzeitig ein Public-Key-Kryptosystem, welches auf ihr aufbaut [25].

Falltür-Funktionen, wie im Beispiel des RSA-Kryptosystems[2], sind Grundlage der Public-Key-Verschlüsselungsverfahren. Im weiteren Verlauf dieser Arbeit wird eine solche Falltür-Funktion vorgestellt.

1.1.2 Homomorphe Verschlüsselungsverfahren

Definition 1.5 (homomorphe Verschlüsselungsverfahren). Sei ein Kryptosystem gegeben, welches aus dem Tupel $(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ besteht. Es gilt, dass $E_k : \mathcal{M} \to \mathcal{C}$ abbildet, wobei $k \in \mathcal{K}$ ein öffentlicher Schlüssel ist. Bilden nun $(\mathcal{M}, \circ)$ und $(\mathcal{C}, \square)$ jeweils eine Gruppe, dann ist das Verschlüsselungsverfahren homomorph falls gilt:

$$E_k(a) \square E_k(b) = E_k(a \circ b)$$

für alle $a, b \in \mathcal{M} \wedge k \in \mathcal{K}$ [26].

Definition 1.6 (Additiv-homomorphe Verschlüsselungsverfahren). Ein Kryptosystem ist additiv-homomorph, falls auf den Chiffraten operiert werden kann, sodass gilt:

$$E_k(a) \otimes E_k(b) = E_k(a + b)$$

für alle $a, b \in \mathcal{M} \wedge k \in \mathcal{K}$ [15].

[2] Dabei handelt es sich um ein asymmetrisches Kryptosystem, welches von Rivest, Shamir und Adleman im Jahr 1997 veröffentlich wurde. RSA ist die Abkürzung der Familiennamen.

Verschlüsselungssysteme, die über diese Eigenschaft hinaus ermöglichen, dass beliebige Operationen auf Klartexte über die Chiffrate durchgeführt werden können, werden auch vollhomomorphe Verschlüsselungssysteme genannt. Darauf wird in dieser Arbeit nicht weiter eingegangen. Für ein tieferes Verständnis wird [13] empfohlen.

1.1.3 Hashfunktion

Eine Hashfunktion zielt darauf ab, eine Eingabemenge beliebiger Länge auf eine Ausgabemenge mit fester Länge abzubilden. Die Eingabe wird dadurch gehasht bzw. auf die Ausgabemenge (Hashwerte) gestreut [19].

Definition 1.7 (Hashfunktion). Sei $\sum_{in}$ eine beliebige Eingabemenge und $\sum_{out}$ eine beliebige Ausgabemenge. Dann ist jede Funktion $h : \sum_{out}^{*} \rightarrow \sum_{out}^{n}$, welche effizient berechnet werden kann, eine Hashfunktion. Sie erzeugt Hashwerte der Länge n [19].

Ein Verwendungszweck von Hashfunktionen ist die Prüfung der Integrität einer Datei. Dafür wird der Hashwert der Datei berechnet, der als Vergleichswert für alle weiteren Hashwerte der Datei benutzt werden kann. Wird diese Datei über einen beliebigen Kanal übertragen, kann der Empfänger nach Anwendung der Hashfunktion auf die Datei überprüfen, ob das Ergebnis der Ausgabe dem Vergleichswert gleicht. In diesem Fall wird der Begriff Prüfsumme verwendet [2].

1.1.4 Kryptografische Hashfunktion

Besonders bei Passwörtern ist es naheliegend sie in einer nicht entschlüsselbaren Form ablegen zu können. Die Verschlüsselung soll möglich sein, umgekehrt soll die Entschlüsselung praktisch unmöglich sein. Eine solche Art von Funktion wird Einweg-Funktion genannt. Dadurch können Passwörter verschlüsselt abgelegt werden, um sie später für einen Vergleich zu benutzen. Der Originalwert des Passworts wird bei der Eingabe mittels der selben Einweg-Funktion gehasht, sodass anschließend beide Hashwerte miteinander verglichen werden können [23] [11] [4]. In diesem Beispiel ist zu erkennen, dass bei Hashfunktionen zwei Eigenschaften zu beachten sind. Die Umkehrung der Hashfunktion darf praktisch nicht möglich sein, da sonst sensible Daten wie Passwörter berechnet werden können. Hingegen müssen die Funktionen kollisionsresistent sein, d.h. dass für zwei verschiedene Eingaben nicht dieselbe Ausgabe errechnet werden kann. Andernfalls wäre es möglich, Eingaben zu finden, welche mit dem Hashwert des Passworts übereinstimmen, ohne das Passwort zu kennen.

Definition 1.8 (Kryptografische Hashfunktion). Eine Hashfunktion $h : \sum_{out}^{*} \rightarrow \sum_{out}^{n}$ ist kryptographisch, wenn sie eine Einwegfunktion und kollisionsresistent ist [2] [17].

Es wird zwischen starken und schwachen kollisionsresistenten Hashfunktionen unterschieden. Eine Hashfunktion ist schwach kollisionsresistent, wenn für einen Klartext $m \in \mathcal{M}$ kein Klartext $m' \neq m$ mit $h(m) = h(m')$ effizient berechnet werden kann. Sie ist stark kollisionsresistent, wenn keine zwei Klartexte $m, m' \in \mathcal{M}$ mit $m' \neq m$ und $h(m) = h(m')$ effizient berechnet werden können [23].

1.1.5 Homomorphe Hashing-Verfahren

Ähnlich wie bei homomorphen Verschlüsselungsverfahren lässt sich die Homomorphie-Eigenschaft auf Hashing-Verfahren übertragen.

Definition 1.9 (Homomorphe Hashing-Verfahren). Ein Hashing-Verfahren h, welches homomorph ist, erfüllt die Bedingung:

$$h(a) \square h(b) = h(a \circ b)$$

für alle $a, b \in \mathcal{M}$.

Definition 1.10 (Additiv-homomorphe Hashing-Verfahren). Ein Hashing-Verfahren ist additiv-homomorph, falls auf den Hashwerten operiert werden kann, sodass gilt:

$$h(a) \otimes h(b) = h(a + b)$$

für alle $a, b \in \mathcal{M}$.

1.1.6 Zero-Knowledge-Beweise

Zero-Knowledge-Proofs (ZKP) oder Zero-Knowledge-Beweise wurden das erste Mal im Jahr 1985 von Goldwasser et al. [14] vorgestellt. Innerhalb ihrer wissenschaftlichen Arbeit beschreiben sie ein Protokoll, welches einer Partei A (Prover) ermöglichen soll, einer weiteren Partei B (Verifier) davon zu überzeugen, dass sie in Besitz eines Geheimnisses ist, ohne Informationen über das Geheimnis preisgeben zu müssen.

Nach einer formalen Definition folgt ein praktisches Beispiel zum besseren Verständnis von Zero-Knowledge-Beweisen, um die Grundlagen des Protokolls zu veranschaulichen Im weiteren Verlauf der Arbeit wird das Ziel eines Zero-Knowledge-Beweises auf die Integritätsprüfung homomorpher Chiffrate portiert.

Definition 1.11 (Zero-Knowledge-Beweis). Seien die Elemente des Tupels (P, V)[3] Turingmaschinen, wobei die Turingmaschine V polynomiell beschränkt ist. P und V können miteinander interagieren, außerdem ist ihr Input identisch. Beschreibe L eine formale Sprache. Dann ist das Tupel (P, V) eine Zero-Knowledge-Beweis-System, falls P und V folgende Eigenschaften besitzen:

1. **Vollständigkeit:** Für jegliche Eingabe $x \in L$ in (P, V) gilt:
   Pr[4]$[s \leftarrow (P, V)(x), V(x, s) = 1] \geq 1 - \frac{1}{n^k}$.
2. **Zuverlässigkeit:** Für jegliche Eingabe $x \notin L$ und einem beliebigen P' gilt:
   $Pr[s \leftarrow (P', V)(x), V(x, s) = 1] < \frac{1}{n^k}$.
3. **Zero-Knowledge:** Für jedes $x \in L$ und jeden Verifier V' existiert ein Simulator S, sodass zwei Verteilungen $S_{V'}(x)$ und $View_{V'}(x)$ rechnerisch ununterscheidbar sind.

Die ersten beiden Eigenschaften sind auch in anderen interaktiven Beweis-Protokollen zu finden, während die dritte Eigenschaft das Protokoll Zero-Knowledge macht. Die Vollständigkeit besagt, dass solange die Eingabe Teil der festgelegten Sprache ist, d.h. der Prover sagt die Wahrheit, der Verifier dies nahezu jedes Mal akzeptiert.

[3] P repräsentiert den Prover und V den Verifier.
[4] Um die Wahrscheinlichkeit P von der Turingmaschine P zu unterscheiden, wurde stattdessen Pr für Verteilungen verwendet.

Umgekehrt soll nach der Zuverlässigkeitseigenschaft kein betrügerischer Prover den Verifier, mit Ausnahme einer kleinen Wahrscheinlichkeit, überzeugen können. Um Zero-Knowledge zu gewährleisten, muss gelten, dass es einen Simulator S gibt, welcher für jeglichen Input die Konversation zwischen P und V' nacherstellen kann. Daraus folgt, dass ein betrügerischer Verifier mittels dieser Konversation nichts berechnen kann, was er nicht bereits vorher hätte berechnen können. Der Verifier hat nach der Konversation kein neues Wissen erhalten. Für ein weiteres Verständnis von Zero-Knowledge-Beweisen wird auf [14] verwiesen.

Zur Veranschaulichung der Grundidee des Konzepts wird nun ein bekanntes Beispiel vorgestellt, welches von Quisquater et al. in ihrer Arbeit *How to Explain Zero-Knowledge Protocols to Your Children* [22] aus dem Jahr 1989 veröffentlicht wurde.

Veranschaulichung eines ZKPs. Angenommen es gibt einen Prover P und einen Verifier V, sodass die Eigenschaften aus Definition 1.11 erfüllt sind. Diese werden ab sofort mit Peggy und Viktor referenziert. Peggy muss Viktor beweisen, dass sie eine Schlüsselkombination für eine Tür innerhalb eines ringförmigen Komplexes besitzt. Jedoch soll Viktor diese Schlüsselkombination niemals erfahren. Die Tür ist aus beiden Gangrichtungen A und B zu erreichen und lässt sich von beiden Seiten öffnen (s. Abbildung 2). Zunächst stehen beide außerhalb des Komplexes. Peggy geht alleine in den Gang hinein und wählt einen der beiden Pfade aus.

Peggy entscheidet sich wie in Abbildung 3 zu sehen für den Pfad A. Viktor steht außerhalb des Eingangs und weiß nicht, für welchen Pfad sich Peggy entschieden hat. Nun tritt Viktor in den ringförmigen Komplex ein und fordert Peggy auf, nach einer zufälligen Auswahl entweder aus Richtung A oder B zu ihm zu kommen.

Nach Abbildung 4 fordert Viktor Peggy auf, von Pfad B aus zu ihm zu gelangen. Da Peggy den Schlüssel besitzt, öffnet sie die Tür und kann über B zu Viktor gelangen. Peggy ist somit der Aufforderung Viktors nachgekommen, ohne dass Viktor den Schlüssel erfahren hat.

Es wird der Fall betrachtet, bei dem Peggy die Schlüsselkombination nicht kennt. Somit besitzt sie stets eine Wahrscheinlichkeit von 50%, sich bereits vor der Aufforderung Viktors auf die Seite des geforderten Pfads hinzustellen. Daher ist es erforderlich, dass Peggy mehreren Aufforderungen nachkommt und Viktor beweist, dass sie jedes Mal aus dem richtigen Pfad zu Viktor zurückkehren kann. Erscheint Peggy nach n-Versuchen immer aus dem richtigen Pfad, dann kann Viktor mit einer Wahrscheinlichkeit von $1 - 2^{-n}$ davon ausgehen, dass Peggy die Schlüsselkombination kennt. Unabhängig von n gilt, dass Viktor dieses Geheimnis niemals erfährt.

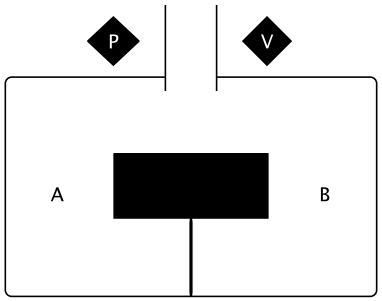

Abbildung 2: *Ringförmiger Gang mit einer Tür, welcher durch einen geeigneten Schlüssel von beiden Seiten geöffnet werden kann.*

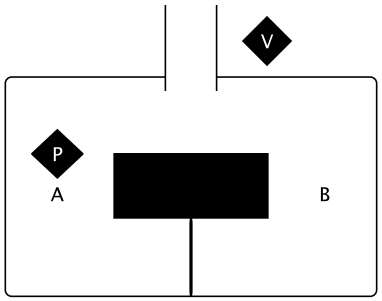

Abbildung 3: *Peggy entscheidet sich für den Pfad A und wartet bis Viktor ihr einen Pfad zuruft.*

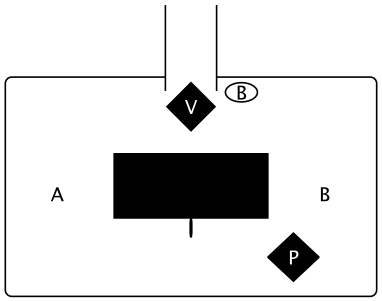

Abbildung 4: *Viktor wählt Pfad B. Peggy hat den Schlüssel und kann die Tür öffnen, um über B zu Viktor zu gelangen.*

2 Paillier-Kryptosystem

Im Jahre 1999 wurde von Pascal Paillier ein Public-Key-Kryptosystem vorgestellt [20], dass auch unter dem Namen Paillier-Kryptosystem bekannt ist. Neben der Public-Key-Eigenschaft, erfüllt dieses Kryptosystem Homomorphie-Eigenschaften, die zur Lösung des Konflikts Privacy vs. Utility verwendet werden können. In diesem Kapitel wird das Kryptosystem vorgestellt, der Kernaufbau des Ver- und Entschlüsselns erläutert und es wird gezeigt, wie das Paillier-Kryptosystem für Analysezwecke genutzt werden kann. Anschließend wird das Verfahren auf die praktische Anwendung hin untersucht und mögliche Schwachstellen werden aufgedeckt. Im Folgenden werden die für diese Arbeit relevanten Kernaussagen der Arbeit von Paillier [20] wiedergegeben. Auf anderen Arbeiten basierende Aussagen sind entsprechend referenziert.

2.1 Decisional-Composite-Residuosity-Annahme (DCRA)

Wie auch das RSA-Kryptosystem, welches auf die Eulersche Phi-Funktion aufbaut, basiert das Paillier-Kryptosystem auf eine Falltür-Funktion. Paillier stellt diese Funktion in [20] vor, welche einer mathematischen Annahme im Kontext des Modulus unterliegt. Es handelt sich dabei um die *Decisional-Composite-Residuosity-Annahme* (DCRA). Diese Annahme wird nun zusammenfassend erklärt:

Sei $n = pq$ die Zusammensetzung zwei großer Primzahlen. Dann besagt die Decisional-Composite-Residuosity-Annahme, dass für eine Ganzzahl $z \in \mathbb{Z}^*_{n^2}$, schwer zu berechnen ist, ob z ein n-ter Rest modulo n^2 ist.

Definition 2.1 (Deciding-Composite-Residuosity). Eine Zahl z ist der n-te Rest modulo n^2, falls eine weitere Zahl $y \in \mathbb{Z}^*_{n^2}$ existiert, sodass gilt:

$$z = y^n \; mod \; n^2. \tag{2.1}$$

Paillier stützt sein Public-Key-Kryptosystem auf diese Annahme, da nicht effizient geprüft werden kann, ob ein solches y existiert. Diese Annahme nutzt Paillier, um eine Einwegfunktion mit Falltür-Eigenschaft aufzustellen. Hieran baut er die *Computational-Composite-Residuosity-Annahme* (CCRA) auf. Die Annahme besagt, dass das von ihm aufgestellte Composite-Residuosity-Problem durch keinen nicht-deterministischen Algorithmus in polynomieller Zeit gelöst werden kann[1]. Für ein tieferes mathematisches Verständnis und den Beweis zur Korrektheit der beiden Annahmen wird auf die wissenschaftliche Arbeit Pailliers [20] verwiesen. Der Fokus der jetzigen Arbeit liegt auf der praktischen Anwendung des Kryptosystems.

[1] Sofern P ≠ NP gilt.

2.2 Schema des Paillier-Kryptosystems

Paillier gibt in derselben Arbeit [20] weitere Modifikationen des Kryptosystems an, jedoch wird zum grundlegenden Verständnis des Public-Key-Kryptosystems die vereinfachte Form des Erzeugens eines Schlüsselpaars, das Ver- und das Entschlüsseln von Klartexten dargestellt.

2.2.1 Schlüsselgenerierung

Es wird davon ausgegangen, dass $n = pq$ ist, wobei p und q große Primzahlen sind.

Privater Schlüssel. Der private Schlüssel ist $\lambda = kgV(p-1, q-1)$.

Öffentlicher Schlüssel. Der öffentliche Schlüssel setzt sich aus n und einer weiteren zufällig gewählten Zahl g zusammen, wobei für g gelten muss:

$$ggT(L(g^{\lambda} \bmod n^2), n) = 1,$$
$$L(u) = \frac{u-1}{n}$$

Dann ist (n, g) die öffentliche Komponente des Public-Key-Schlüsselpaars und (λ) die private Komponente. Dementsprechend sind die Primzahlen p und q ebenfalls privat.

2.2.2 Verschlüsselung

Sei $m < n$ ein Klartext, r zufällig gewählt, sodass $r < n$ und die Verschlüsselungsfunktion $E_{n,g}(m)$ mit m als Eingabe und (n, g) als konstanter Parameter. Dann ergibt sich das Chiffrat c aus dem Klartext wie folgt:

$$c = E_{n,g}(m) = g^m \cdot r^n \bmod n^2 \tag{2.2}$$

Probabilismus von Paillier. Wird ein Klartext zweimal verschlüsselt, dann sollten unter der Bedingung, dass r eine Zufallszahl ist, zwei verschiedene Zufallszahlen r verwendet werden. Dies hat zur Folge, dass die beiden resultierenden Chiffrate verschieden sind. Dadurch erhält das Kryptosystem seine probabilistische Eigenschaft. Anders als bei Hashfunktionen, welche meist deterministisch sind, kann ein Klartext mehrfach verschlüsselt werden, ohne dass die Chiffrate des Klartexts identisch sind. Die Infrastruktur des Paillier-Kryptosystems bietet außerdem die Möglichkeit, ein Chiffrat erneut zu chiffrieren bzw. zu randomisieren. Diese Operation wird in den kommenden Kapiteln vorgestellt und diskutiert.

2.2.3 Entschlüsselung

Sei $c < n^2$ ein Chiffretext, $D_{\lambda}(c)$ die Entschlüsselungsfunktion mit c als Eingabe und λ als Falltür-Parameter. Dann wird das Chiffrat c wie folgt entschlüsselt:

$$m = \frac{L(c^{\lambda} \bmod n^2)}{L(g^{\lambda} \bmod n^2)} \bmod n \tag{2.3}$$

2.2.4 Pailliers Falltür-Funktion

Die Formel (2.2) ist eine Einwegfunktion, da sie nicht effizient genug invertiert werden kann. Die Invertierung liegt genau der Problematik der DCRA zu Grunde, aus der die Eigenschaft einer Einwegfunktion folgt.

Theorem 2. Die Verschlüsselung mittels Paillier ist genau dann und nur dann eine Einwegfunktion, wenn die *Computational-Composite-Residuosity-Annahme* hält.

Das Invertieren der Verschlüsselungsfunktion entspricht genau dem *Composite-Residuosity-Class-Problem*, welches Paillier in seiner Arbeit definiert hat. Ist die Verschlüsselungsfunktion eine Einwegfunktion, dann besitzt sie zusätzlich eine Falltür. Denn mit dem zusätzlichen Wissen von λ bzw. p und q ist die Verschlüsselung über die Entschlüsselungsfunktion (2.3) umkehrbar.

2.3 Homomorphie-Eigenschaft des Paillier-Kryptosystems

Das Paillier-Kryptosystem ist additiv-homomorph. Unter Betrachtung von (2.2) wird der Aufbau eines Chiffrats deutlich. Gegeben seien zwei Chiffrate:

$$c_1 = E_{n,g}(m_1) = g^{m_1} \cdot r^n \ mod \ n^2$$
$$c_2 = E_{n,g}(m_2) = g^{m_2} \cdot r^n \ mod \ n^2$$

Es ist möglich, auf den Chiffraten zu operieren, sodass die Klartexte der korrespondierenden Chiffrate sich aufaddieren und einen neuen Chiffretext ergeben:

$$\begin{gathered} c_1 \cdot c_2 = (g^{m_1} \cdot r^n \ mod \ n^2) \cdot (g^{m_2} \cdot r^n \ mod \ n^2) \Leftrightarrow \\ (g^{m_1} g^{m_2} \cdot r^{n^2}) \ mod \ n^2 = (g^{m_1+m_2} \cdot r^{n^2}) \ mod \ n^2 \Leftrightarrow \\ g^{m_{1+2}} \cdot r'^n \ mod \ n^2 = c_{1+2} \end{gathered} \tag{2.4}$$

Die Entschlüsselung von c_{1+2} aus (2.4) mit Hilfe des korrespondierenden λ und (2.3) ergibt:

$$\begin{gathered} m_{1+2} = \frac{L(c_{1+2}^{\lambda} \ mod \ n^2)}{L(g^{\lambda} \ mod \ n^2)} \ mod \ n, \\ m_{1+2} = m_1 + m_2 \end{gathered} \tag{2.5}$$

Beim Multiplizieren zweier Chiffrate addieren sich die Potenzen der Teilfaktoren auf. Dadurch entsteht eine neue Zufallszahl r und die Nachrichten m_1 und m_2 stehen als Summe in der Potenz von g, wie in der letzten Gleichung von (2.4) zu erkennen ist. Dies repräsentiert ein neues Chiffrat, welches $m_1 + m_2$ verschlüsselt. Durch Entschlüsseln des neuen Chiffrats wird die Summe der beiden Klartexte m_1 und m_2 aufgedeckt (s. 2.5). Verallgemeinert lässt sich sagen, dass für alle $a, b \in \mathcal{M} \wedge k \in \mathcal{K}$ gilt:

$$E_k(a) \cdot E_k(b) = E_k(a + b) \tag{2.6}$$

Paillier bietet drei weitere Möglichkeiten an, auf den Chiffraten zu operieren. In kommenden Kapiteln wird betrachtet, wie diese Operationen zum Aufdecken von Klartexten im praktischen Gebrauch des Kryptosystems führen können.

- Addition eines Klartexts mit einem Chiffrat:

$$g^m r^n \cdot g^{\Delta m} = g^{m+\Delta m} r^n \ mod \ n^2 \tag{2.7}$$

- Erneute Randomisierung des Chiffrats (Addition einer 0):

$$g^m r^n \cdot \Delta r^n = g^m r'^n \ mod \ n^2 \tag{2.8}$$

- Multiplikation eines Klartexts mit einem Chiffrat:

$$(g^m r^n)^w = g^{wm} r^{wn} = g^{wm} r'^n \ mod \ n^2 \tag{2.9}$$

2.4 Paillier im Kontext Privacy vs. Utility

Sicherheit des Paillier-Kryptosystems. Paillier gibt an, dass das Kryptosystem semantisch sicher ist. Er stellt ein Theorem auf, dass das Kryptosystem genau dann und nur dann semantisch sicher ist, wenn die (DCRA) hält. Da das Problem hinter der Annahme nicht effizient lösbar ist, hält das Kryptosystem *Chosen-Plaintext-Angriffen* stand (IND-CPA). Es ist möglich, die Chiffrate des Paillier-Kryptosystems zu verformen, ohne die Klartexte zu verändern. Deshalb ist das Kryptosystem gegenüber adaptiven *Chosen-Ciphertext-Angriffen* (IND-CCA2) nicht sicher. Paillier und Pointchevral [21] stellen das Paillier-Kryptosystem in modifizierter Form vor, sodass dieses zumindest im Random-Oracle-Modell[2] gegenüber IND-CCA2 als sicher gilt [26].

Nutzbarkeit der Daten nach der Verschlüsselung. Bei nicht homomorphen Kryptosystemen ist nach der Verschlüsselung keine Modifikation der Klartexte auf Ebene der Chiffrate möglich. Wurde ein ausreichend großer Schlüssel im Zusammenhang mit einem als sicher anzunehmenden Verfahren verwendet, dann besteht die einzige Möglichkeit, auf den Klartexten zu operieren, darin, die Chiffrate vorher zu entschlüsseln. Dies widerspricht der Anforderung, die Klartexte initial verschlüsseln zu wollen. Anders ist es bei einem Kryptosystem mit Homomorphie-Eigenschaften. Werden die Klartexte innerhalb eines homomorphen Kryptosystems verschlüsselt, so wie am Beispiel von Paillier erkennbar, ist es möglich, Klartexte über Operationen auf den Chiffraten zu modifizieren. Die modulare Multiplikation der Chiffrate entspricht der Addition im Klartextraum (2.6).

Bei geschicktem Einsetzen von applikationsabhängigen Schwellenwerten (thresholds) ist es sogar möglich, diese Operation zu invertieren und somit die Klartexte zu subtrahieren [10].

Die Homomorphie-Eigenschaft des Paillier-Kryptosystems ermöglicht bei Anwendung auf sensiblen Daten, sowohl die Privatsphäre der Datenbesitzer zu wahren, als auch Nutzbarkeit der verschlüsselten Daten zu realisieren. Mit Hilfe des öffentlichen Schlüssels können zwei oder mehrere Klartexte über ihre Chiffrate miteinander addiert oder subtrahiert werden, ohne dass die Ergebnisse dieser Operationen als Klartexte bekannt werden. Dadurch lassen sich grundlegende Analyseoperationen durchführen, wie die Berechnung von Durchschnittswerten oder Vergleichsoperationen. Problematisch wird es, wenn mit dem Ergebnis in Klartextform weitergerechnet werden muss. Dies ist ohne eine Entschlüsselung mittels des privaten Schlüssels nicht möglich. Das Paillier-Kryptosystem wird nun in der praktischen Anwendung betrachtet.

[2] Ein zufälliges Orakel, welches eine ideale kryptographische Hashfunktion modelliert.

2.5 Praktische Anwendung verschiedener Analyseoperationen mittels Paillier

In diesem Teilkapitel sollen anhand eines gegebenen Datensatzes verschiedene Analyseoperationen nachgegangen werden, um aufzuzeigen, auf welche Grenzen ein sogenannter *Analyzer* beim Analysieren der Daten stößt. Dafür habe ein beliebiges Unternehmen Daten erhoben und mittels Paillier verschlüsselt. Das Unternehmen möchte, nachdem sie die benötigten Informationen an den Analyzer weitergeleitet hat, keine weiteren Kapazitäten in den Analyseprozess investieren.

Öffentlicher Schlüssel. Der Analyzer erhält einen öffentlichen Schlüssel (n, g), der Klartexte mittels Paillier verschlüsseln kann.

Datensatz. Er erhält einen Datensatz, der genau zehn Elemente $(c_0 \dots c_9)$ enthält, welche aus der Verschlüsselung mittels des öffentlichen Schlüssels (n, g) des Paillier-Kryptosystems entstanden sind. Dabei entsprechen die Klartexte hinter den Chiffraten jeweils den eigenen Indizes. D.h. $(m_0 = 0; m_1 = 1 \dots)$ und auf diesen sollen nun Operationen durchgeführt werden.

Für Analysezwecke möchte ein Unternehmen folgendes wissen:

1. Was ist die Gesamtsumme aller Werte?
2. Welcher Klartext ist größer: der zu c_1 oder c_5 gehörige?

Um die Gesamtsumme zu erhalten, müssen alle Chiffrate miteinander multipliziert werden. Die zweite Frage lässt sich lösen, indem die Klartexte hinter den Chiffraten subtrahiert werden. Je nachdem, ob das Ergebnis positiv, negativ oder neutral ist, kann entschieden werden, wie sich die beiden Chiffrate voneinander unterscheiden. Nachdem der Analyzer die nötigen Operationen auf den Chiffraten durchgeführt hat, muss er die Ergebnisse zunächst verschlüsselt an das Unternehmen senden, da er keinen Zugriff auf den privaten Schlüssel hat, um eine direkte Antwort auf die Fragen geben zu können. Außerdem kann es sein, dass der Analyzer mehrere Ergebnisse in entschlüsselter Form benötigt, um eine endgültige Antwort auf Fragestellungen geben zu können. Dies bedeutet, dass während der Analyse mit dem Analyzer interagiert werden muss.

Idealerweise ist eine Interaktion zwischen dem Unternehmen und dem Analyzer bis zur Beantwortung der Fragen nicht gegeben. Den Analyzer selbst mit dem privaten Schlüssel auszustatten ist ausgeschlossen. Wäre der Analyzer im Besitz des privaten Schlüssels, könnte er nicht nur das Ergebnis seiner Operation entschlüsseln, sondern auch die Chiffrate $(c_0 \dots c_9)$. Somit hätte der Analyzer Zugang zu sensiblen Daten. Daher ist in Erwägung zu ziehen, einer weiteren Entität in diesem Szenario den privaten Schlüssel anzuvertrauen, die die Entschlüsselungsoperation übernimmt. Solche externen Entitäten oder Parteien werden auch Trusted Third Party genannt. In den folgenden Kapiteln wird dieser Gedanke aufgegriffen und weiter diskutiert, sodass ein Verfahren vorgestellt wird, welches zulässt Analyseoperation praxisbezogen durchzuführen.

3 Analyseoperationen mittels Paillier-Chiffraten und einer Trusted Third Party

Angenommen eine Partei A (Data Holder) hat Daten erhoben und diese mit dem Paillier-Kryptosystem verschlüsselt. Für die Analyse wurde eine externe Partei B (Analyzer) beauftragt. Partei B ist es nicht gestattet die Daten im Klartext einzusehen, daher erhält sie neben dem öffentlichen Schlüssel lediglich die Chiffrate der Klartexte. Partei A hat kein Vertrauen in Partei B und geht nicht davon aus, dass sich Partei B stets im legalen Rahmen bewegt. Zudem entscheidet Partei A, dass es in ihrem Sinn ist, keine weiteren zeitlichen Ressourcen in die Analyse zu investieren. Da die Partei B jemanden benötigt, der die Analyseergebnisse entschlüsselt, beauftragt Partei A eine weitere Partei C (Decryptor). Partei C besitzt den privaten Schlüssel und genießt ein bestimmtes Niveau an Vertrauen, um Aufgaben zu übernehmen, die den vorher festgelegten Vertrauensrahmen nicht überschreiten. Es handelt sich um eine Trusted Third Party.

Definition 3.1 (Trusted Third Party). Bei einer Trusted Third Party (TTP) handelt es sich um eine Entität innerhalb eines Szenarios, welche das Vertrauen aller anderen Beteiligten besitzt. Diese Partei kann dabei verschiedenen Aufgaben wie der Authentifikation, Schlüsselverteilung oder Zertifizierungen nachgehen [7].

In diesem Szenario wird angenommen, dass die TTP und der Analyzer ausschließlich protokollkonform interagieren. Der TTP wird seitens der Partei A in dem Maße vertraut, dass sie davon ausgeht, dass die TTP den privaten Schlüssel ausschließlich zum Entschlüsseln von Analyseergebnissen verwendet. Es wird ein Modell betrachtet, um einen beispielhaften Ablauf einer Addition über Chiffrate und anschließender Entschlüsselung vorzustellen. Im Folgenden werden die Beteiligten des vereinfachten Szenarios beschrieben.

3.1 Vereinfachtes Szenario

Data Holder (DH). Der Data Holder sammelt eine Menge $\mathcal{M}$ von Klartextelementen. Es wird davon ausgegangen, dass alle gesammelten Daten sensibel sind und nicht unbefugt eingesehen werden dürfen. Weiterhin besitzt der DH ein Schlüsselpaar $k_{Paillier} \in \mathcal{K}$ bestehend aus dem privaten Schlüssel (λ) und dem öffentlichen Schlüssel (n, g) des Paillier-Kryptosystems. Mit dem öffentlichen Schlüssel chiffriert er den sensiblen Datensatz und erhält die korrespondierende Menge der Chiffrate $\mathcal{C}_{k_{Paillier}}$.

Analyzer. Der Analyzer besitzt die Aufgabe, die sensiblen Daten zu analysieren. Dafür erhält er vom DH Chiffrate, die aus dem Verschlüsselungsverfahren des Paillier-Kryptosystems entstanden sind. Um homomorphe Operationen auf den Chiffraten durchzuführen, besitzt er zusätzlich den öffentlichen Schlüssel (n, g).

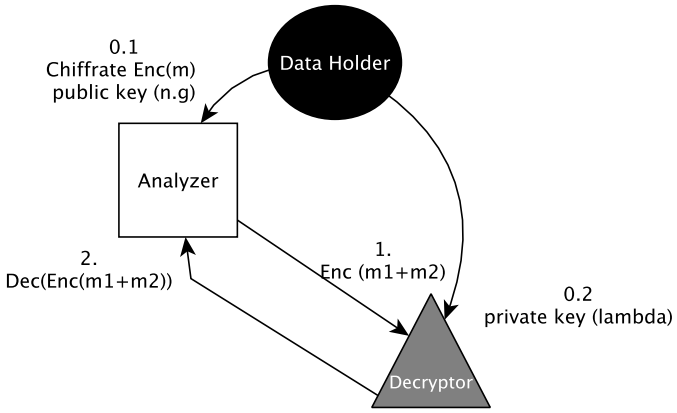

Abbildung 5: *Relationen und Wissensübersicht der involvierten Parteien im vereinfachten Szenario.*

Decryptor. Der Decryptor entschlüsselt die Ergebnisse, die er vom Analyzer erhält, mit dem privaten Schlüssel (λ) und sendet die Klartexte wieder zurück. Dabei handelt es sich beim Decryptor um eine Trusted Third Party, die das Vertrauen genießt, diesen Schlüssel nicht weiterzugeben und lediglich für die vorher beschriebene Aufgabe zu nutzen. Wichtig ist anzumerken, dass für das vereinfachte Szenario diese Vertrauensbasis als ausreichend betrachtet wird. Im erweiterten Szenario wird versucht das Wissen der TTP gezielt einzuschränken, um die Möglichkeit außerhalb der Abmachung zu agieren, minimal zu halten.

In Abbildung 5 wird das Verhältnis der Parteien zueinander und ihr Wissenstand aufgeführt. Das Verhältnis des Data Holders mit den beiden anderen Parteien ist einseitig, welches von ihm ausgeht. Sobald die jeweils benötigten Daten an den Analyzer und Decryptor weitergeleitet wurden, ist der DH in diesem Szenario nicht mehr aktiv. Nur noch Analyzer und Decryptor interagieren miteinander. Nun wird ein beispielhafter Ablauf zwischen Analyzer und Decryptor betrachtet:

- Seien $c_1, c_2 \in \mathcal{C}_{k_{Paillier}}$, wobei $c_1 = g^{m_1} \cdot r^n \; mod \; n^2$ und $c_2 = g^{m_2} \cdot r^n \; mod \; n^2$, die Chiffrate der Klartexte $m_1, m_2 \in \mathcal{M}$.
- Der Analyzer kann nun wie in (2.6) die Chiffrate c_1 und c_2 miteinander multiplizieren und die Klartexte addieren. Dadurch erhält er ein neues Chiffrat c_{1+2}, dessen Klartext $m_1 + m_2$ gleicht.
- Der Analyzer sendet das Chiffrat c_{1+2} an den Decryptor. Dieser kann unter Verwendung des privaten Schlüssels und der Formel (2.3) das Chiffrat entschlüsseln und das Ergebnis an den Analyzer senden. Der Ablauf ist abgeschlossen.

3.2 Sicherheitsanalyse des vereinfachten Szenarios

Das vereinfachte Modell in Abbildung 5 soll dem generellen Verständnis dienen. In diesem Kapitel wird nun das vereinfachte Szenario analysiert und grundlegende Probleme für eine praxisnahe Anwendung dargestellt.

Angenommen dem Decryptor kann vollständig vertraut werden und er hält sich an die mit ihr getätigten Vereinbarungen. Es ist also auszuschließen, dass er unbefugte Daten weiterleitet oder versucht selber an Informationen zu kommen, die er nicht einsehen darf. Es werden mögliche Lücken erörtert, die der Analyzer aus dem vereinfachten Szenario ausnutzen kann, um illegal an sensible Daten zu gelangen. Der Analyzer und Decryptor besitzen folgende Informationen:

Analyzer: Besitzt die Chiffrate $C_{k_{Paillier}}$ und den öffentlichen Schlüssel (n, g) zum Operieren.

Decryptor: Besitzt den privaten Schlüssel λ zum Entschlüsseln.

In einem solchen Szenario ist es trivial zu bemerken, dass dem Analyzer keine Hindernisse bevorstehen, um dem Decryptor Chiffrate aus $C_{k_{Paillier}}$ zu senden, welche dann entschlüsselt werden sollen. Der Decryptor entschlüsselt die Chiffrate, deckt damit sensible Daten auf und sendet sie an den Analyzer. Der Decryptor nimmt nicht zur Kenntnis, dass es sich um sensible Informationen handelt, der Analyzer hingegen hat es geschafft, illegal zu handeln und sich unbefugte Informationen anzueignen. Dem Decryptor fehlt eine Möglichkeit zu verifizieren, dass das zu entschlüsselnde Chiffrat aus einer Operation über die Chiffrate aus $C_{k_{Paillier}}$ entstanden ist.

Zu diesem deutlichen Problem könnte eine eindeutige Lösung folgen. Würde der Decryptor zusätzlich mit den Chiffraten aus $C_{k_{Paillier}}$ ausgestattet werden, dann scheint die Gefahr vermeintlich beseitigt worden zu sein. Sendet ihm der Analyzer nun ein Chiffrat aus $C_{k_{Paillier}}$, dann könnte er dies erkennen. Im vereinfachten Szenario wird angenommen, dass der Decryptor den privaten Schlüssel nicht nutzen würde, um die Chiffrate aus $C_{k_{Paillier}}$ zu entschlüsseln. Dennoch besitzt der Analyzer eine Möglichkeit sich Zugang zu sensiblen Daten zu verschaffen.

Dafür werden die in (2.7), (2.8) und (2.9) beschriebenen Eigenschaften des Paillier-Kryptosystems, welche dem Analyzer ermöglichen die Chiffrate zu verformen, herangezogen. Unter Anwendung eines der Operationen wären die Chiffrate für den Decryptor nicht wiederzuerkennen. Die wohl einfachste Variante wäre die Addition mit einer 0 auf ein beliebiges Chiffrat $c \in C_k$. Dies führt zur Erzeugung eines von c verschiedenen Chiffrats c', wobei beide Chiffrate denselben Klartext m verschlüsseln. Über die Eigenschaften (2.8) und (2.9) ist das Erzeugen eines neuen Chiffrats ebenfalls möglich. Dabei ändert sich der Klartextwert entsprechend der Operation. Der Analyzer kann diese Operation revidieren, indem er nach der Entschlüsselung den jeweiligen Wert subtrahiert oder dividiert. Somit sind dem Analyzer mehrere Möglichkeiten gegeben, dass er den Decryptor nicht nachvollziehen lässt, ob die von ihm zu entschlüsselnden Chiffrate aus den Chiffraten der Menge $C_{k_{Paillier}}$ stammen. Das Weiterleiten der Chiffrate $C_{k_{Paillier}}$ an den Decryptor löst das Problem nicht.

Die Herausforderung in einem solchen System besteht darin, den Decryptor effizient nachvollziehen zu lassen, dass die Chiffrate, die er entschlüsseln soll, aus einer legalen Rechnung[1] resultieren. Es muss ein Rahmen geschaffen werden, der dem Analyzer die Möglichkeit gibt zu beweisen, dass die verschlüsselten Summen aus den Chiffraten der Menge $C_{k_{Paillier}}$ entstanden sind. Außerdem muss es möglich sein, dass der Decryptor eindeutig und effizient verifizieren kann, dass die Behauptungen des Analyzers korrekt sind. Da sich in diesem Rahmen der Fokus des Decryptors auf das Verifizieren verschiebt, wird dieser ab sofort Verifier genannt.

[1] Damit sind diejenigen Rechnungen gemeint, deren Ergebnisse nach der Entschlüsselung nicht ausreichend genug darüber informieren, um auf sensible Daten zu schließen.

4 Homomorphe Hashing-Verfahren

Krohn et al. [16] beschreiben als einer der ersten ein homomorphes Hashing-Verfahren. Das Verfahren wird verwendet, um in Echtzeit *Erasure Codes*[1] beim Downloaden von Daten zu verifizieren. Die Hashfunktion, welche kollisionsresistent ist, basiert auf dem diskreten Logarithmus. Das in [16] vorgestellte homomorphe Hashing-Verfahren findet in dieser Arbeit keine Verwendung, da die Hashfunktion die Klartextlänge festlegt und diese nicht flexibel ist. Für ein vollständiges Verständnis wird auf die Arbeit [16] verwiesen.

4.1 Additiv-homomorphe Hashfunktion

Gazzoni Filho und Barreto [12] stellen ein homomorphes Hashing-Verfahren vor, welches das RSA-Kryptosystem als Grundlage verwendet. Dabei wird der private Schlüssel nach der Schlüsselgenerierung dauerhaft gelöscht. Der öffentliche Schlüssel, der zuvor für die Verschlüsselung von Klartexten vorgesehen war, wird nun als Schlüssel zur Generierung eines Hashwertes verwendet.

4.1.1 Definition

Definition 4.1 (Additiv-homomorphe Hashfunktion). Sei $n = pq$, wobei p und q zwei Primzahlen sind, sodass n ein RSA-Modulus ist. Außerdem sei $\phi(n) = (p-1)(q-1)$ die Ordnung von $(\mathbb{Z}/n\mathbb{Z})^*$. Dann besteht der Schlüssel der Hashfunktion aus n und einem zufällig ausgwählten Integerwert b. Um Daten d beliebiger Länge zu hashen, muss folgendes berechnet werden [12]:

$$H(d) = b^d \ mod \ n \tag{4.1}$$

Die Homomorphie-Eigenschaft wird unter der Betrachtung von (4.1) offensichtlich. Werden zwei Hashwerte $H(d)$ und $H(d')$ miteinander multipliziert, dann resultiet daraus:

$$H(d)H(d') = b^d b^{d'} \ mod \ n = b^{d+d'} \ mod \ n = H(d+d') \tag{4.2}$$

Anhand von (4.2) ist erkennbar, dass es sich um eine additiv-homomorphe Hashfunktion handelt. Durch Multiplikation im Hashraum wird die Addition im Klartextraum erreicht. Für den weiteren Verlauf der Arbeit wird der Schlüssel der Hashfunktion bestehend aus dem RSA-Modulus n und b als k_H bezeichnet.

[1] Vorwärtskorrektur Codes des binären Auslöschungskanals

4.1.2 Sicherheit des Verfahrens

Um eine Kollision in diesem Hashing-Verfahren herbeizuführen, müssen zwei Klartexte d und d' mit $H(d) = H(d')$ gefunden werden. Dann gilt, dass $b^d \equiv b^{d'} \bmod n$, woraus folgen würde: $b^{d-d'} \equiv 1 \bmod n$. Nach dem Satz von Euler müsste $d - d'$ ein Vielfaches von $\phi(n)$ sein. Damit beschränkt sich das Herbeiführen einer Kollision auf das Suchen von kongruenten Zahlen bezogen auf den Modulus $\phi(n)$. Somit wäre das Herbeiführen einer Kollision trivial, wenn $\phi(n)$ oder eine Faktorisierung von n bekannt ist. Die Sicherheit des Verfahrens beruht auf das Faktorisierungsproblem, welches mathematisch hart ist [12].

4.2 Homomorphe Hashing-Verfahren basierend auf homomorphen Kryptosystemen

Homomorphe Hashfunktionen können auf homomorphe Kryptosysteme aufbauen. Dafür wird der öffentliche Schlüssel eines homomorphen Kryptosystems zum Generieren von Hashwerten verwendet, der private Schlüssel wird verworfen. Die Chiffrate des homomorphen Kryptosystems sind die Hashwerte der homomorphen Hashfunktion. Dadurch werden alle Eigenschaften des Kryptosystems auf die Hashfunktion übertragen. Eine homomorphe Hashfunktion basierend auf Paillier wäre probabilistisch und additiv-homomorph. Zusätzlich könnten alle Operationen auf den Chiffraten des Paillier-Kryptosystems auch auf die Hashwerte durchgeführt werden.

Homomorphe Hashing-Verfahren, die auf homomorphen Kryptosystemen aufbauen, sind auch an deren Zeitkomplexität gebunden. Im weiteren Verlauf dieser Arbeit folgt eine Evaluation der Implementierung der Hashfunktion aus 4.1 und einer Hashfunktion, die auf Paillier basiert. Diese wird zeigen, dass aus Komplexitätsgründen das Verfahren aus 4.1 zu bevorzugen ist.

5 Paillier in einem verteilten Mehrparteienszenario

In diesem Kapitel wird ein Konzept zum Lösen der im vereinfachten Szenario aus Kapitel 3 beschriebenen Problematik, effiziente Integritätsprüfungen auf Paillier-Chiffrate durchzuführen, vorgestellt. Ziel ist ein praxisnahes Verfahren zu realisieren, welches die Analyse von Daten unter Berücksichtigung von Privatsphäre schützenden Vorgaben ermöglicht.

5.1 Parteien

Für das Konzept werden die folgenden drei Parteien betrachtet:

Data Holder. Der Data Holder erhebt Daten und verschlüsselt diese mittels des Paillier-Kryptosystems. Die dadurch entstandenen Chiffrate sollen für Analysezwecke zur Verfügung gestellt werden. Der Data Holder möchte unbedingt verhindern, dass eine der miteingebundenen Parteien die Klartexte der Daten einsieht. Außerdem erstellt er basierend auf einer homomorphen Hashfunktion, einen weiteren Datensatz, welcher die Hashwerte der Klartexte enthält. Teile der Daten gehen an den Analyzer und den Verifier, um Analayseoperationen zu ermöglichen. Wichtig ist zu erwähnen, dass der Data Holder, sobald er die Daten ausgeteilt hat, nicht mehr aktiv ist. Erst wenn der Vorgang für neue Klartexte wiederholt werden soll, wird der Data Holder wieder aktiv.

Analyzer Die Aufgabe des Analyzers ist Operationen durchzuführen, welche den Analysezwecken nachkommen. Außerdem muss er dem Verifier beweisen können, dass die verschlüsselten Ergebnisse, die er vom Verifier entschlüsselt haben möchte, aus dem vom Data Holder bereitgestellten Datensatz stammen. Es wird angenommen, dass der Analyzer an den erhobenen Klartexten des Data Holders interessiert ist.

Verifier Der Verifier besitzt die Aufgabe die verschlüsselten Ergebnisse für den Analyzer zu entschlüsseln. Auf Grund der Verformbarkeit der Paillier-Chiffrate ist dies nicht trivial. Er muss feststellen können, dass der zu entschlüsselnde Datensatz keine Rückschlüsse auf die Klartexte lässt oder dass die Klartexte selbst entschlüsselt werden. Der Verifier fungiert als Trusted Third Party, der soweit vertraut wird, dass sie niemandem unbefugt ihren Wissenstand mitteilt oder mit dem Analyzer durch Entschlüsselung von Klartexten kooperiert. Seine Möglichkeiten Informationen während des Vorgangs zu gewinnen, sollen minimal gehalten werden.

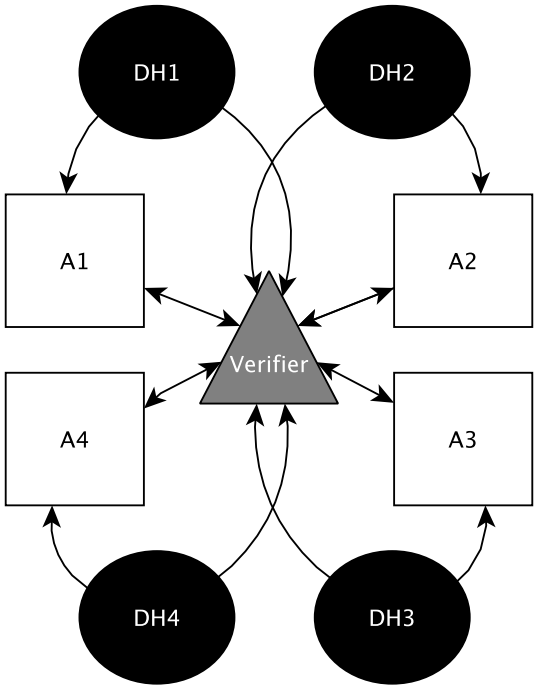

Abbildung 6: *Beispiel eines Mehrparteienszenarios im Kontext von Analyseoperationen auf Paillier-Chiffrate.*

In Abbildung 6 wird eine mögliche Beziehung von mehreren Data Holdern und Analyzern mit nur einem einzigen Verifier dargestellt. Die Relation, die von einem Data Holder ausgeht, impliziert das Verteilen der Daten nach Abschließen der Setup-Schritte. Die Beziehung zwischen Analyzer und Verifier impliziert den gesamten Verifizierungsprozess zum Entschlüsseln von Rechenergebnissen. Wichtig ist anzumerken, dass zwei Data Holder durchaus den Dienst desselben Analyzers in Anspruch nehmen können. Es soll möglich sein, dass ein Analyzer die Daten von zwei oder mehreren Data Holdern aggregiert und analysiert.

5.2 Ziel des Konzepts

Bevor das Konzept und die Verfahren darin näher erläutert werden, wird das Ziel des Konzepts beschrieben. Es soll ermöglicht werden, dass der Analyzer dem Verifier beweist, dass seine Berechnungen legal sind, um die entschlüsselten Ergebnisse der Berechnungen vom Verifier zu erhalten. Zudem muss der Verifier eine effiziente Möglichkeit besitzen, diese Berechnungen zu verifizieren. Nach Abschluss des Verfahrens sollen beide beteiligten Parteien nicht mehr Wissen über die Klartexte besitzen wie vor Anwendung des Verfahrens, d.h. das Wissen ist nicht ausreichend genug, um Rückschlüsse auf sensible Daten ziehen zu können. Dabei ist es wichtig, dass dies sowohl für den Analyzer als auch für den Verifier gilt. Es wird vorausgesetzt, dass Vertrauen zum Verifier besteht, sodass dieser nicht mit dem Analyzer zum Entschlüsseln von verschlüsselten Klartexten kooperiert. Andernfalls ist der Zugang zu den sensiblen Daten trivial.

5.3 Vorraussetzungen/Setup

Im Folgenden werden die Setup-Schritte des Data Holders beschrieben. Diese werden noch vor Beginn des Protokolls durchgeführt. Nach Abschluss des Setups ist der Data-Holder nicht mehr Teil des Protokolls.

0.1 Der Data Holder erhebt Daten und besitzt eine Menge $\mathcal{M}$.

0.2.1 Der Data Holder generiert ein Schlüsselpaar $k_{Paillier} \in \mathcal{K}$ des Paillier Kryptosystems und erhält den öffentlichen Schlüssel (n, g) und den privaten Schlüssel λ.

0.2.2 Der Data Holder generiert ein Schlüsselpaar des RSA-Kryptosystems (verwirft den privaten Schlüssel) und erhält zusammen mit einem zufälligen Integerwert b: $k_H \in \mathcal{K}$.

0.3 Der Data Holder erzeugt einen verschlüsselten Datensatz aus den Klartexten der Menge $\mathcal{M}$ mittels des öffentlichen Schlüssels von Paillier und erhält die Chiffratmenge $\mathcal{C}_{k_{Paillier}}$.

0.4 Der Data Holder erzeugt einen gehashten Datensatz aus den Klartexten $\mathcal{M}$ mittels k_H (s. Kapitel 4.1) und erhält die Hashmenge $\mathcal{H}_{k_H}$.

0.5 Der Data Holder sendet $\mathcal{C}_k$ und (n, g) an den Analyzer innerhalb eines sicheren Kanals.

0.6 Der Data Holder sendet $\mathcal{H}_{k_H}$, λ und k_H an den Verifier innerhalb eines sicheren Kanals.

Nach dem Setup haben die drei Parteien folgenden Wissenstand:

Data Holder.
1. **Klartexte $\mathcal{M}$:** Diese liegen dem Data Holder zu Beginn vor.
2. **Paillier-Schlüsselpaar:** Erhält er zu Beginn des Setups des Paillier-Kryptosystems.
3. **Schlüssel der homomorphen Hashfunktion:** Ist der öffentliche Schlüssel eines RSA-Schlüsselpaars und ein zufälliger Integerwert b.
4. **Verschlüsselter Datensatz $\mathcal{C}_{k_{Paillier}}$:** Die Klartexte werden über den öffentlichen Schlüssel des Paillier-Kryptosystems verschlüsselt.
5. **Hashwerte der Klartexte $\mathcal{H}_{k_H}$:** Nach Anwendung der homomorphen Hashfunktion auf die Klartexte erhält der Data Holder einen neuen Datensatz.

Analyzer.
1. **Chiffrate der Menge $\mathcal{C}_{k_{Paillier}}$:** Dem Analyzer werden die zu analysierenden Datensätze als Chiffrate zur Verfügung gestellt.
2. **Öffentlicher Schlüssel (n, g) des Paillier-Kryptosystems:** Der Verifier benötigt den öffentlichen Schlüssel, um die Operationen auf dem verschlüsselten Datensatz durchzuführen.

Verifier.
1. **Privater Schlüssel λ des Paillier-Kryptosystems:** Der Verifier erhält den privaten Schlüssel. Dieser wird benötigt um die Chiffrate zu entschlüsseln.
2. **Schlüssel der homomorphen Hashfunktion:** Mit Hilfe des öffentlichen Schlüssels k_H kann der Verifier Klartexte in den Raum der Hashwerte abbilden, welche dann additiv-homomorph sind.
3. **Hashwerte der Klartexte $\mathcal{H}_{k_H}$:** Um die Rechnungen des Analyzers auf ihre Integrität überprüfen zu können, werden dem Verifier Hashwerte der Klartexte zur Verfügung gestellt. Auf diesen können additiv-homomorphe Operationen durchgeführt werden.

5.4 Das Verfahren

Es folgt das Verfahren, welches das vorher beschriebene Ziel realisiert. Dafür müssen legale Operationen auf den Chiffraten der Menge $\mathcal{C}_{k_{Paillier}}$ seitens des Analyzers definiert werden.

Legale Operationen. Sollen an Hand des Paillier-Kryptosystems Analyseoperationen auf den Chiffraten der Menge $\mathcal{C}_{k_{Paillier}}$ durchgeführt werden, dann sind nur Operationen im Sinne der Gleichung (2.6) legal. Das bedeutet, dass diejenigen Operationen legal sind, die nur Chiffrate aus der Menge $\mathcal{C}_{k_{Paillier}}$ nutzen. Dementsprechend sind jegliche Operationen, die Klartexte verwenden, sodass nach der Entschlüsselung des Ergebnisses die Klartexte der verwendeten Chiffrate aus der Menge $\mathcal{C}_{k_{Paillier}}$ ohne Hindernisse aufgedeckt werden können, illegal. Dies beinhaltet das Addieren bzw. Multiplizieren von Klartexten zu bzw. mit einem Chiffrat. Außerdem ist jegliche Form von neuer Randomisierung, d.h. die Addition einer verschlüsselten Null, ebenfalls illegal. Damit sind die Gleichungen (2.7), (2.8) und (2.9) als illegal zu betrachten.

Nachdem das Setup vollständig durchgeführt wurde, findet die Kommunikation zwischen Analyzer und Verifier statt, der Data Holder ist nicht mehr aktiv. Dabei kann das Verfahren in vier Schritte eingeteilt werden.

1. Decryption Request (Analyzer)
2. Verification Process (Verifier)
3. Decryption (Verifier)
   - Decryption True
   - Decryption False
4. Check Decryption (Analyzer)

Für einen beispielhaften Ablauf wird jeder dieser Schritte beschrieben. Seien $c_1, c_2, c_3 \in \mathcal{C}_{k_{Paillier}}$ dem Analyzer vorliegende Chiffrate. Angenommen c_{1+2} sei das Ergebnis einer Additions-Operation auf c_1 und c_2. Außerdem sei c_3' das Ergebnis einer Addition von c_3 mit einer verschlüsselten Null.

Decryption Request. Der Analyzer fordert den Verifier auf, dass er ihm die Ergebnisse c_{1+2} und c_3' mit Hilfe des privaten Schlüssels entschlüsseln soll. Dafür sendet er dem Verifier die Chiffrate über einen sicheren Kanal zu. Zusätzlich wird der Verifier darüber informiert, dass c_{1+2} aus den Chiffraten mit *ID*[1] 1 und 2 entstammt. Da c_3' ein illegales Chiffrat ist, gibt der Analyzer fälschlicher Weise die *IDs* 2 und 3 an. Der Verifier erhält somit zwei Datenstrukturen, mit jeweils einem zu entschlüsselnden Chiffrat und *IDs*, die angeben sollen, aus welchen Chiffraten der Menge $\mathcal{C}_{k_{Paillier}}$ es entstanden ist.

Verification Process. Nachdem der Verifier die Daten des Analyzers erhalten hat, beginnt er den Verifizierungsprozess. Dafür entschlüsselt er c_{1+2} und c_3'. Im Anschluss hasht er die Klartexte mit Hilfe von k_H. So erhält er die Hashwerte h_{1+2} und h_3'. Mit Hilfe der IDs findet er die zugehörigen Hashwerte h_1, h_2 und h_3 und führt die entsprechenden Operationen auf ihnen nach (4.2) durch. Daraufhin prüft der Verifier, dass gilt:

- $h_1 h_2 == h_{1+2}$?
- $h_2 h_3 == h_3'$?

[1] Dem Analyzer ist in irgendeiner Weise die Möglichkeit gegeben, auf Chiffrate der Menge $\mathcal{C}_{k_{Paillier}}$ zu referenzieren, sodass der Verifier diese Referenz nachvollziehen kann.

Offensichtlich ist c_{1+2} aus den Chiffraten c_1 und c_2 der Menge $\mathcal{C}_{k_{Paillier}}$ entstanden. Das Resultat des Vergleichs ist das boolesche *True*. Der Vergleich für h'_3 hat das boolesche *False* als Folge, da c_3 nur erneut randomisiert wurde.

Decryption. Abhängig vom Resultat des Vergleichs, entscheidet der Verifier, ob er die entschlüsselten Chiffrate $D_k(c_{1+2})$ und $D_k(c'_3)$ dem Analyzer zuschickt. In diesem Fall, erhält der Analyzer $D_k(c_{1+2})$, da er nachweisen konnte, dass dieser Wert aus einer legalen Operation stammt. Für $D_k(c'_3)$ ist dies nicht der Fall.[2]

Check Decryption. Der Analyzer erhält seine Datenstrukturen zurück und prüft, ob die Chiffrate für ihn entschlüsselt worden sind. Ist das Feld, indem vorher das zu entschlüsselnde Chiffrat stand, frei, dann wurde die Entschlüsselung verweigert. Ist ein Wert enthalten, wie im Fall $D_k(c_{1+2})$, dann wurde das Chiffrat erfolgreich entschlüsselt.

5.5 Wissensstand des Verifiers und Analyzers nach Ablauf des Verfahrens

Wurden die vier Schritte des Verfahrens durchlaufen, dann ändert sich der Wissensstand der beiden Parteien wie folgt:

Wissensstand des Verifiers. Unabhängig davon, ob der Analyzer legale oder illegale Operationen durchführt, muss der Verifier stets die Chiffrate entschlüsseln. Im Fall einer nicht legalen Entschlüsselungsanfrage des Analyzers, deckt der Verifier dennoch die Klartexte der Chiffrate aus der Menge $\mathcal{C}_{k_{Paillier}}$ auf. Somit kann er gegebenenfalls sensible Daten einsehen. Dem Verifier wird soweit vertraut, dass er das Wissen nicht mit dem Analyzer teilt, jedoch wäre es nicht möglich, den Verifier selbst zu kontrollieren, ob er von diesem unbefugten Wissen Gebrauch macht. Dem Verifier wird zwar vertraut, aber gleichzeitig wird versucht die Informationsgewinnung während des Ablaufs des Konzepts minimal zu halten. Aus diesem Grund ist es nötig, dass der Data Holder die sensiblen Klartexte vor der Verschlüsselung randomisiert.

Randomisierung der Klartexte. Um vorzubeugen, dass der Verifier sensible Daten einsehen kann, sollte der Data Holder zu Beginn einen randomisierten Datensatz erstellen. Zudem speichert er einen weiteren Datensatz, der angibt, wie die Klartexte randomisiert worden sind, sodass die Randomisierung im Nachhinein revidiert werden kann. Nun setzt er die Setup-Schritte fort, sendet dem Analyzer aber zusätzlich den Datensatz, der angibt, wie die Klartexte der Chiffrate randomisiert worden sind. Erhält er die Klartexte vom Verifier, kann er mit Hilfe des zusätzlichen Datensatzes die Randomisierung aufheben und den originalen Klartextwert erhalten. Gleichzeitig wird dadurch verhindert, dass der Verifier sensible Daten einsieht. Entschlüsselt er nämlich ein Chiffrat, unabhängig davon, ob es aus legalen oder illegalen Operationen stammt, dann ist nach der Entschlüsselung nicht festzustellen, ob es sich um den originalen Klartext handelt.

[2] Abhängig von der jeweils verfolgten Sicherheitspolitik, wäre es durchaus möglich, dass ausgehend von dem illegalen Versuch des Analyzers keine Chiffrate mehr entschlüsselt werden und der Analyzer bis auf Weiteres ausgeschlossen wird.

Das erweiterte Setup sieht nun wie folgt aus:

0.1 Der Data Holder sammelt Daten und besitzt eine Menge $\mathcal{M}$.
0.2.1 Der Data Holder generiert ein Schlüsselpaar $k_{Paillier} \in \mathcal{K}$ des Paillier Kryptosystems und erhält den öffentlichen Schlüssel (n, g) und den privaten Schlüssel λ.
0.2.2 Der Data Holder generiert ein Schlüsselpaar des RSA-Kryptosystems (verwirft den privaten Schlüssel) und erhält zusammen mit einem zufälligen Integerwert b: $k_H \in \mathcal{K}$.
0.3 Der Data Holder randomisiert die Elemente aus $\mathcal{M}$ und erhält einen neuen Datensatz $\mathcal{M}'$. Zusätzlich speichert er die **Randomisierungsliste** $\mathcal{R}$ mit der Zuordnung zu den Klartexten.
0.4 Der Data Holder erzeugt einen verschlüsselten Datensatz aus den randomisierten Klartexten der Menge $\mathcal{M}'$ mittels des öffentlichen Schlüssels von Paillier und erhält die Chiffratmenge $\mathcal{C}'_{k_{Paillier}}$.
0.5 Der Data Holder erzeugt einen gehashten Datensatz aus den randomisierten Klartexten $\mathcal{M}'$ mittels k_H wie in (4.1) und erhält die Hashmenge $\mathcal{H}'_{k_H}$.
0.5 Der Data Holder sendet $\mathcal{C}'_{k_{Paillier}}$, $\mathcal{R}$ und (n, g) an den Analyzer innerhalb eines sicheren Kanals.
0.6 Der Data Holder sendet $\mathcal{H}'_{k_H}$, λ und k_H an den Verifier innerhalb eines sicheren Kanals.

Wissensstand des Analyzers. Für den Analyzer werde die Fälle *Decryption True* und *Decryption False* separat betrachtet.

Decryption False. Wird dem Analyzer kein Wert zurückgesendet, dann bleibt sein Wissensstand unverändert.

Decryption True. Wird dem Analyzer die entschlüsselte Lösung zugesendet, dann hat sich sein Wissensstand offensichtlich verändert. Jedoch hat er kein direktes Wissen über die sensiblen Klartexte gewonnen. Dennoch wird mit jeder erfolgreichen Entschlüsselung eine Relation zwischen den Chiffraten und dem Ergebnis bekannt. Wird das vorangegangene Beispiel herangezogen, so weiß der Analyzer, dass die Addition der Klartexte m_1 und m_2 $D_k(c_{1+2})$ ergeben. Problematisch wird dies, wenn gezielt versucht wird, ein lineares Gleichungssystem (LGS) zum Lösen der Unbekannten aufzustellen. Dafür muss der Analyzer noch zwei weitere legale Operationen durchführen, undzwar die Operationen auf c_1 und c_3 bzw. c_2 und c_3. Der Verifier wird erkennen, dass es sich um legale Operationen handelt und sendet dem Analyzer dann $D_k(c_{1+3})$ und $D_k(c_{2+3})$ zurück. Der Analyzer muss folgendes lineares Gleichungssystem aufstellen, um die Klartexte hinter c_1, c_2 und c_3 zu erhalten:

1. $m_1 + m_2 = D_k(c_{1+2})$
2. $m_1 + m_3 = D_k(c_{1+3})$
3. $m_2 + m_3 = D_k(c_{2+3})$

Der Analyzer besitzt drei Unbekannte m_1,m_2 und m_3 und drei Gleichungen, sodass das lineare Gleichungssystem lösbar ist.

Die Problematik des Aufstellens eines linearen Gleichungssystems wird im kommenden Kapitel noch einmal aufgegriffen. Hier wird lediglich darauf hingewiesen, dass der Verifier das Aufstellen eines LGS verhindern muss. Gelingt dies dem Verifier, dann besitzt er trotzdem die Möglichkeit selbst ein LGS aufzustellen, um mögliche Chiffrate der Menge $\mathcal{C}'_k$ zu erhalten. Diese Problematik wird mit Hilfe der Randomisierung behoben. Angenommen er stellt ein solches LGS auf, welches mindestens ein Chiffrat löst, dann ist die Lösung stets randomisiert, da der Verifier nicht im Besitz der Randomisieriungsliste $\mathcal{R}$ ist.

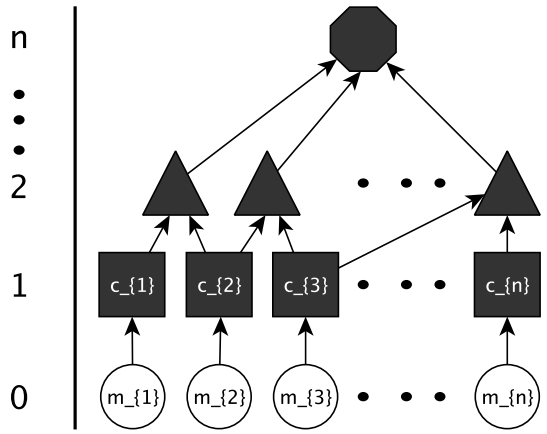

Abbildung 7: *Level-Übersicht in Abhängigkeit der enthaltenen Chiffrate.*

5.6 Level-Funktion

Um die Errechnung der Klartexte aus den entschlüsselten Summen zu erschweren, wird das Konzept erweitert. Die Level-Funktion wird eingeführt, welche angibt, aus wie vielen Chiffraten eine verschlüsselte Summe besteht.

Definition 5.1 (Level-Funktion) Die Level-Funktion $\mathcal{L}(c)$ gibt an, aus wie vielen verschiedenen Chiffraten der Menge $\mathcal{C}$ ein Chiffrat c besteht. Das Level aller Chiffrate aus $\mathcal{C}$ ist stets 1. Das Level aller Klartexte aus $\mathcal{M}$ ist stets 0.

In Abbdildung 7 wird eine Übersicht der Level dargestellt. Die Klatexte besitzen das Level 0, da keine Chiffrate enthalten sind. Die Chiffrate der Menge $\mathcal{C}$ befinden sich auf Level 1. Über die Dreiecke werden alle Chiffrate repräsentiert, die aus zwei Chiffraten mit Level 1 entstanden sind. Das Chiffrat, welches als Achteck repräsentiert wird, befindet sich auf Level n, da es aus allen n-Chiffraten der Menge $\mathcal{C}$ entstanden ist.

Wird eine verschlüsselte Summe aus Chiffraten gebildet, die nicht in Level 1 sondern höher liegen, dann wird betrachtet, wie viele verschiedene Chiffrate aus Level 1 verwendet wurden:

- $c' = c_1 c_2 c_3$, mit $\mathcal{L}(c') = 3$
- $c'' = c_1 c_2$, mit $\mathcal{L}(c'') = 2$
- $c''' = c'c''$, mit $\mathcal{L}(c''') =?$
- $c''' = c_1 c_2 c_3 c_1 c_2 = c_1^2 c_2^2 c_3$, mit $\mathcal{L}(c''') = 3$

Dem Verifier kann vorgeschrieben werden, dass er nur Chiffrate entschlüsseln darf, deren Level nicht kleiner als ein festgelegtes k sind. Zusätzlich wird dadurch erschwert, dass lineare Gleichungssysteme gebildet werden, die dazu führen, Klartexte zu erhalten. Je mehr Chiffrate aus Level 1 miteinbezogen werden müssen, um so mehr Gleichungen werden benötigt, um ein lösbares Gleichungssystem aufzustellen.

6 Implementierung des verteilten Mehrparteienszenarios

Im vorliegenden Kapitel wird die Implementierung des in Kapitel 5.4 vorgestellten Konzepts beschrieben. Als Programmiersprache wurde Python[1] gewählt. Für das Paillier-Kryptosystem wurde eine bereits bestehende Bibliothek [6] verwendet. Zusätzlich wurde zur Realisierung des homomorphen Hashing-Verfahrens auf Basis von RSA eine Python RSA-Bibliothek [24] verwendet.

6.1 Implementierte Bibliotheken

Die Implementierung unterteilt sich in vier Bibliotheken:

1. Paillier-Kryptosystem[2] (*paillier.py*)
2. Homomorphes-Hashing Verfahren (*hh.py*)
3. Tools *tools.py*
4. TLS Connector *connector.py*

Mit diesen Bibliotheken ist das verteilte Mehrparteienszenario realisierbar. Nach der Beschreibung der einzelnen Bibliotheken erfolgt eine Evaluierung der Implementierung.

Paillier-Kryptosystem. In der Bibliothek *paillier.py* des Moduls *phe* [6] sind alle nötigen Methoden für das Paillier-Kryptosystem enthalten. Neben der Schlüsselgenerierung und der Ver- und Entschlüsselung, findet sich die Operation der Addition der Klartexte über die Chiffrate wieder. Auch negative und Gleitkommazahlen können verschlüsselt werden. Zu den für das Verfahren relevanten Methoden wurde zusätzlich eine Evaluierung durchgeführt, welche die praktische Eignung der Bibliotheken demonstrieren soll.

Homomorphes Hashing-Verfahren. Für die Realisierung des Konzepts wurde ein additiv-homomorphes Hashing-Verfahren wie aus Kapitel 4.1 implementiert. Dieses befindet sich in *hh.py (homomorphic hash)*. Die Schlüsselgenerierung wird über Pythons RSA-Bibliothek realisiert, wobei nur der öffentliche Schlüssel gespeichert wird. Der öffentliche Schlüssel wird dann als Modulus verwendet, um die Klartexte mittels eines zufällig ausgewählten Integerwerts zu hashen. Zusätzlich enthält die Bibliothek die Operation der Addition der Klartexte über die Hashwerte. Außerdem lassen sich negative Integerwerte als Hash darstellen. Auf Grund des Aufbaus des Hash-Verfahrens, ist das Hashen einer 0 nicht möglich. Jedoch tritt der Fall, dass eine 0 gehasht werden muss, selten ein.[3]

[1] Python 3
[2] Diese Bibliothek ist im Package so nicht enthalten, viel mehr baut die Bibliothek *Tools* auf sie auf.
[3] Es wird nur dann eine 0 im Datensatz zu finden sein, wenn im Radomisierungsprozess auf eine negative Zahle das äquivalente positive Gegenstück hinzuaddiert wurde.

Tools. Für Data Holder, Analyzer und Verifier werden alle nötigen Methoden bereitgestellt, welche für einen korrekten Ablauf notwendig sind. Klartexte in einer vorgegebenen Tabellenform können eingelesen, randomisiert, verschlüsselt und gehasht werden. Auf verschlüsselte oder gehashte Daten können Operationen durchgeführt werden, mit der Option, dass nur Teilbereiche der Daten miteinbezogen werden sollen. Für die zu entschlüsselnden Chiffrate steht eine Klasse zur Verfügung, die einen Datentypen erstellt, welche alle nötigen Informationen für den Verifizierungsprozess enthält. Über diesen Datentyp kann der Verifier mit Hilfe der Methode *verify*, die Chiffrate verifizieren und bei Erfolg entschlüsselt an den Analyzer zurücksenden. Als Parameter kann ein Mindestlevel vorgegeben werden, welches für jedes Chiffrat eingehalten werden muss. Da die Implementierung einer effizienten Methode, welche prüft, ob ein lineares Gleichungssystem aufgestellt werden kann, den Rahmen dieser Arbeit überschreiten würde, ist eine entsprechende Methode nicht enthalten.

TLS Connector. Für den Datenaustausch zwischen allen beteiligten Parteien ist es wichtig, dass diese über einen sicheren Kanal versendet werden. Die Bibliothek *connector.py* ermöglicht den Parteien, innerhalb eines TLS Handshakes eine Verbindung aufzunehmen und Daten auszutauschen. Dabei können verschiedene Verbindungsmodi ausgewählt werden. Es wird zwischen der Setup-Verbindung vom Data Holder und Analyzer/Verifier und dem Verifizierungs-Prozess vom Analyzer und Verifier unterschieden.

6.2 Verifizierung einer verschlüsselten Summe

Um mehrere Chiffrate miteinander zu multiplizieren (Addition der Klartexte), kann der Analyzer die jeweiligen Teilsummen aus einem Datensatz einlesen und in einer Liste speichern. Mit dieser Liste und dem öffentlichen Schlüssel des Paillier-Kryptosystems kann er ein Objekt erstellen, welches die verschlüsselte Summe und die *IDs* aller Teilsummen selbst enthält. Nachdem das Objekt an den Verifier gesendet wurde, kann er mit Hilfe der *IDs* der Teilsummen erkennen, welche Hashwerte für den Vergleich benötigt werden. Er muss alle Hashwerte miteinander multiplizieren (Addition der Klartexte) und die entschlüsselte Chiffresumme hashen. Daraufhin werden beide Ergebnisse auf Gleichheit geprüft. Bei Korrektheit wird die entschlüsselte Summe an den Analyzer zurückgesendet, bei Ungleichheit wird ein leerer Wert zurückgegeben.

6.3 Verschlüsseln und Hashen von negativen Zahlen

Sowohl die Bibliothek des Paillier-Kryptosystems als auch die selbstimplementierte Bibliothek des homomorphen Hashing-Verfahrens erlaubt es, negative Zahlen zu verschlüsseln bzw. zu hashen. Dafür wird der große Modulus ausgenutzt, um die entsprechend äquivalente positive Zahl darzustellen. Es folgt eine Beschreibung, wie negative Zahlen in der Eigenimplementierung gehasht werden:

Hashen negativer Zahlen. Sei n der verwendete RSA-Modulus mit einer Mindestlänge von 2048 Bits. Dann geht der gesamte Hashraum $\mathcal{H}_{k_H}$ von 0 bis $n-1$. n ist groß genug, sodass die Werte $(n-1, n-2, n-3\dots)$ als negative Zahlen $(-1, -2, -3\dots)$ interpretiert werden können, da ihre Verwendung unwahrscheinlich ist. Deshalb wird für das Hashen von negativen Zahlen folgendes festgelegt:

- Für alle $x \in \mathbb{Z}^-$ wird n hinzuaddiert: $x + n = n - x \bmod n$.

Damit werden alle negativen Zahlen x mit $\mathcal{H}_{k_H}(x) = b^{n-x} \bmod n$ im Hashraum dargestellt.

Für den Verifizierungsprozess ist zu beachten, dass zu jeder negativen Zahl der Wert n hinzuaddiert wurde. Sei $a \in \mathbb{Z}^-$, welches aus drei Teilsummen $x_1, x_2, x_3 \in \mathbb{Z}^-$ entstanden ist. Angenommen der Verifier erhält ein Chiffrat $E_{Paillier}(a)$, welches a verschlüsselt. Dieses Chiffrat soll für den Analyzer entschlüsselt werden. Um die Integrität des Chiffrats zu prüfen, muss er das Chiffrat wie im *Verification Process* beschrieben entschlüsseln, hashen und mit dem dazugehörigen Hashwert vergleichen. Dafür werden folgende Berechnungen durchgeführt:

1. $D_{Paillier}(E_{Paillier}(a)) = a$ (Entschlüsseln)
2. $H_{k_H}(a) = b^{n-a} \bmod n$ (Hashen)
3. $H_{k_H}(x_1)H_{k_H}(x_2)H_{k_H}(x_3) = b^{n-x_1} * b^{n-x_2} * b^{n-x_3} = b^{3n-a} \pmod n$ (Addition)
4. $b^{n-a} == b^{3n-a} \pmod n$? (Vergleich)

Der Vergleich wird niemals erfolgreich sein, da der Verifier beim Hashen des Klartexts a nur ein n hinzuaddiert hat. Da a aus drei negativen Teilsummen entstanden ist, besitzt der Hashwert, mit dem verglichen wird, $3n$ in der Potenz. Also muss der Verifier zu den entschlüsselten Summen, das korrekte Vielfache von n hinzuaddieren. Dafür muss er nicht nur wissen, was die Teilsummen der zu entschlüsselnden Summen sind, sondern auch wie viele negative Zahlen sich unter ihnen befinden. Dadurch können in der Implementierung negative Zahlen gehasht werden und es ist möglich, additiv-homomorphe Operationen auf ihnen durchzuführen.

7 Evaluation

7.1 Aufbau

Die Implementierung des Mehrparteienszenarios wurde evaluiert, um insbesondere die Laufzeiten der jeweiligen Methoden bewerten zu können. Dabei werden zwei Arten von Methoden unterschieden, die Methoden des Setup-Prozesses und die Methoden des Verifizierungs-Prozesses. In beiden Prozessen wurden die Kernmethoden, wie das Verschlüsseln und Hashen von Klartexten, auf denen die Bibliotheken aufbauen, evaluiert. Die zugrundeliegende Hardware, welche bei der Evaluation verwendet wurde, kann aus Tabelle 1 entnommen werden. Für die Evaluation wurde die Python-Bibliothek *timeit* genutzt. Mit ihr ist es möglich, die Zeit ab dem Start bis zum Ende der Ausführung einer Methode festzuhalten. Es wurden Messreihen durchgeführt, um den *Best Case* und *Average Case* für die eigenen Hardwarespezifikationen festzulegen. Gerundet wurde auf die sechste Nachkommastelle. Für alle Operationen, die einen Schlüssel benötigen, wurde eine Schlüssellänge von 2048 Bits verwendet.

Tabelle 1: *Verwendete Hardware bei der Durchführung der Evaluation.*

	Hardwarespezifikation
Betriebssystem	Unix (OS X 10.11.6)
Arbeitsspeicher	4 GB 1600 MHz DDR3
Prozessor	1,3 GHz Intel Core i5

7.2 Messwerte

Tabelle 2: *Laufzeiten der Setup-Methoden.*

Bibliothek	Methoden	Beschreibung	Laufzeit (Best Case)	Laufzeit (Average Case)
phe:paillier.py	generate_paillier_keypair	Generiert ein Schlüsselpaar des Paillier-Kryptosystems. Die verwendetete Schlüssellänge beträgt 2048 Bits.	229.079347 ms	361.023834 ms
phe:pailler.py	encrypt	Verschlüsselt einen Klartext.	113.791352 ms	120.233564 ms
hhic:hh.py	generateKey	Generiert ein RSA-Schlüsselpaar, wobei der private Schlüssel verworfen wird. Die Schlüssellänge beträgt 2048 Bits.	1252.528338 ms	3193.421082 ms
hhic:hh.py	hash	Hasht einen Klartext.	0.379661 ms	0,410004 ms
hhic:hh.py	randomizeData	Randomisiert einen Klartext.	0.037797 ms	0.040869 ms

Tabelle 3: *Laufzeiten der Methoden des Verifizierungsprozesses.*

Bibliothek	Methoden	Beschreibung	Laufzeit (Best Case)	Laufzeit (Average Case)
phe:paillier.py	decrypt	Entschlüsselt ein Chiffrat mittels des privaten Schlüssels.	57.827835 ms	58.292018 ms
phe:pailler.py	_add_encrypted	Addiert zwei verschlüsselte Klartexte.	0.131927 ms	0.140188 ms
hhic:hh.py	addHomomorph	Addiert zwei gehashte Klartexte.	0.044837 ms	0.053258 ms

7.3 Evaluation der Setup-Methoden

Das Setup wird vom Data Holder durchgeführt. Alle nötigen Methoden, die das Setup ermöglichen, wurden evaluiert und auf ihre Korrektheit überprüft. In Tabelle 2 sind die Laufzeiten der einzelnen Methoden ablesbar. Die Schlüsselgenerierung des RSA-Kryptosystems mit 1,2 bis 3,1 Sekunden und des Paillier-Kryptosystems mit 0,2 bis 0,4 Sekunden sind unter der Betrachtung, dass sie einmalig durchgeführt werden, zumutbar. Es ist wichtig anzumerken, dass beim Setup keine Interaktion mit anderen Parteien stattfindet. Der Data Holder bereitet zunächst die Daten auf und leitet mit der Verteilung der Daten die Interaktion zwischen Verifier und Analyzer ein. Dementsprechend ist die Laufzeit der Verschlüsselung im Paillier-Kryptosystem in diesem Kontext zu betrachten. Je mehr Dateneinträge ein zu verschlüsselnder Datensatz besitzt, desto größer wird die zur Verschlüsselung erforderliche Laufzeit. Bei einer durchschnittlichen Laufzeit von 0,12 Sekunden würde der Algorithmus bei 1.000 Klartexten etwa 2 Minuten benötigen. Deutlicher wird es für mehr Klartexte. Auch dies sei erlaubt, da der Setup-Prozess vor dem Verifizierungs-Prozess stattfindet.

Über einen Vergleich der Komplexität des Hashens und des Entschlüsselns eines Klartexts wird deutlich, wieso das Hashing-Verfahren aus Kapitel 4.1 zu bevorzugen ist. Die Hashfunktion des Paillier-Hashing-Verfahrens entspricht der Entschlüsselung im Paillier-Kryptosystem. Die Laufzeit des Entschlüsselns ist 300-mal größer als die der implementierten Hashfunktion.

7.4 Evaluation der Verifizierungs-Methoden

Bei der interaktiven Kommunikation zwischen Analyzer und Verifier ist es wichtig, dass die jeweiligen Schritte des Verfahrens effizient abgearbeitet werden. Die drei Kernmethoden sind die Entschlüsselung mittels des privaten Schlüssels des Paillier-Kryptosystems, das Hashen der Klartexte, sowie die Addition von verschlüsselten bzw. gehashten Klartexten. Die Addition zweier verschlüsselter Klartexte wird hauptsächlich vom Analyzer durchgeführt. Diese befindet sich deutlich unterhalb einer Millisekunde (s. Tabelle 3). Die Addition gehashter Klartexte wird noch schneller durchgeführt. Somit werden die zwei Kernmethoden, die innerhalb kürzester Zeit immer wieder angewendet werden müssen, effizient abgearbeitet. Die Methode zur Entschlüsselung von Chiffraten, welche vom Verifier genutzt wird, hat mit 0,058 Sekunden eine deutlich längere Laufzeit. Dieser Wert liegt dennoch unter einer Zehntelsekunde. Zudem wird die Methode für jede verschlüsselte Summe einmal angewendet. Alle weiteren Methoden des Verifizierungsprozesses bauen auf den vorher beschriebenen Methoden auf.

8 Related Work

Voll-homomorphe Kryptosysteme [13] sind für den praktischen Gebrauch nicht effizient genug. Derweil wurden Kryptosysteme vorgestellt, welche nur teilweise homomorph sind. Das vorgestellte Paillier-Kryptosystem [20] ist beispielsweise additiv-homomorph. Hingegen hat ElGamal [8] ein Kryptosystem vorgestellt, welches multiplikativ-homomorph[1] ist.

Das Paillier-Kryptosystem wird bereits in anderen Bereichen angewendet. Eines der bekannteren Fälle ist das E-Voting. Baudron et al. [3] und Damgard und Jurik [5] stellen auf das Paillier-Kryptosystem aufbauende E-Voting-Protokolle vor. Wähler geben ihre Stimme verschlüsselt ab, woraufhin das Wahlergebnis über die Summe aller Stimmen berechnet wird. Da das Paillier-Kryptosystem probabilistisch ist, sollten die Chiffrate zwei gleicher Stimmabgaben niemals identisch sein. Die verschlüsselte Summe kann entschlüsselt und das Ergebnis bekanntgegeben werden, ohne die Stimmabgaben aufzudecken.

Krohn et al. [16] stellen homomorphe Hashing-Verfahren vor, welche zur Verifikation von *Erasure Codes* beim Downloaden von Daten in Echtzeit dienen sollen. Das Hashing-Verfahren basiert auf dem diskreten Logarithmus und ist kollisionsresistent. Die verwendete Hashfunktion legt die Länge der Klartexte fest. Hingegen stellen Gazzoni Filho und Barreto [12] ein Hashing-Verfahren vor, bei dem die Klartextlänge flexibel sein kann. Das Hashing-Verfahren ist additiv-homomorph und die Sicherheit des Verfahrens baut auf das Faktorisierungsproblem auf. Die homomorphe Hashfunktion wird im Rahmen mehrerer Protokolle vorgestellt, welche das Betrügen beim Datenaustausch verhindern sollen.

Ein bekanntes Beispiel für interaktive Zero-Knowledge-Beweise [14] ist das Feige-Fiat-Shamir-Protokoll [9]. Feige et al. präsentieren ein Verfahren, welches einem Geheimnisträger ermöglicht, einen Verifier zu überzeugen, dass er in Besitz eines Schlüssels ist, ohne Informationen über den Schlüssel preisgeben zu müssen. Die Sicherheit des Verfahrens basiert auf der Schwierigkeit, Quadratwurzeln in einem Restklassenring $\mathbb{Z}_n$ zu berechnen. Die Verfahren von Baudron et al. [3] und Damgard und Jurik [5] sind auch Zero-Knowledge-Protokolle. Bei einem Wahlgang soll es nicht möglich sein, Informationen über die Stimmabgaben zu gewinnen. Für die Realisierung der Zero-Knowledge-Eigenschaft werden mehrere Verschlüsselungsschritte durchgeführt, die das Verfahren verlangsamen. Allerdings benötigt das vorgestellte Mehrparteienszenario dieser Arbeit ein Protokoll, welches eine effiziente Kommunikation zwischen Analyzer und Verifier ermöglicht.

[1] Eine Operation auf den Chiffraten ermöglicht die Multiplikation der Klartexte.

9 Zusammenfassung

Mittels homomorpher Kryptosysteme ist es möglich, den Konflikt der Vertraulichkeit und Verfügbarkeit von Daten zu lösen. Klartexte lassen sich verschlüsseln und ermöglichen, auf den Klartexten über die Chiffrate zu operieren, ohne dass sie aufgedeckt werden. In dieser Arbeit wurde das Kryptosystem von Paillier vorgestellt, welches homomorph und probabilistisch ist. Die Anwendung eines solchen Verschlüsselungsschemas stellt für den praktischen Gebrauch eine Herausforderung dar. Die Verantwortung für Analyseoperationen und Entschlüsselungen müssen an verschiedene Parteien verteilt werden. Außerdem besteht die Problematik, dass auf Grund der Verformbarkeit der Chiffrate erschwert wird, die Integrität der zu entschlüsselnden Chiffrate zu überprüfen.

In dieser Arbeit wurde ein Mehrparteienszenario vorgestellt, das zeigt, wie das Paillier-Kryptosystem im Kontext der Datenanalyse eingesetzt werden kann. Dabei besteht die minimalste Variante des Szenarios aus drei Parteien. Diese sind der Data Holder, welcher Daten erhebt und aufbereitet, der Analyzer, welcher Analyseoperationen mit den Chiffraten durchführt und der Verifier, welcher die Integrität von Chiffraten überprüft und diese gegebenenfalls entschlüsselt. Dabei wurde in Anlehnung an einen Zero-Knowledge-Beweis ein Verfahren vorgestellt, welches dem Analyzer ermöglichen soll, zu beweisen, dass die zu entschlüsselnden Chiffrate aus legalen Operationen entstanden sind. Gleichzeitig wird das erworbene Wissen, welches im Laufe des Beweises entsteht, minimal gehalten. Für die Integritätsprüfung wurden homomorphe Hashfunktionen vorgestellt. Mit Hilfe von homomorphen Hashwerten ist der Verifier in der Lage, die Rechenschritte des Analyzers zu rekonstruieren, ohne Chiffrate oder Klartexte kennen zu müssen.

Das gesamte Verfahren wurde implementiert und evaluiert. Die Ergebnisse der Evaluation deuten darauf hin, dass die Implementierung für den praktischen Gebrauch geeignet ist. Die Methoden, welche vom Data Holder verwendet werden, sind vergleichsweise langsam. Jedoch werden diese im Setup angewendet, wo längere Laufzeiten eher in Kauf genommen werden können. Alle weiteren Methoden, die in der interaktiven Kommunikation zwischen Analyzer und Verifier verwendet werden, waren effizient.

9.1 Ausblick

Die folgenden Bereiche sind noch offen und können für zukünftige Arbeiten aufgegriffen werden:

Das Verfahren ist nach Definition nicht Zero-Knowledge. Die Informationsgewinnung wird weitestgehend minimiert und nicht abgeschaffen. Dafür muss eine Sprache im Zusammenhang mit dem Paillier-Kryptosystem definiert werden, welche dem Analyzer ermöglicht, den Verifier von seiner Korrektheit zu überzeugen, sodass nach Ablauf des Protokolls kein neues Wissen gewonnen werden kann. Dies kann insbesondere bedeuten, dass homomorphe Hashing-Verfahren nicht zu einer solchen Lösung beitragen können und andere Verfahren betrachtet werden müssen.

Offen ist auch, wie der Verifier effizient prüfen kann, ob der Analyzer in der Lage ist, ein lineares Gleichungssystem aufzustellen und zu lösen. Die Festlegung, dass das Entschlüsseln eines Chiffrats erst ab einem bestimmten Level durchgeführt werden darf, erschwert das Bilden eines linearen Gleichungssystems. Dennoch muss dem Analyzer diese Möglichkeit vollständig genommen werden.

Das Verfahren kann auf andere homomorphe Kryptosysteme adaptiert oder mit diesen erweitert werden, sodass neben der Addition von Klartexten z.B. auch die Multiplikation ermöglicht wird.

9.2 Fazit

Einleitend wurde die Frage gestellt, wie es ermöglicht werden kann, dass sensible Daten geschützt werden, aber dennoch nutzbar sind. Homomorphe Verschlüsselungsverfahren sind ein Mittel dies zu erreichen. Um sie auf die Praxis bezogen nutzbar zu machen, muss ein Rahmen geschaffen werden, welcher ermöglicht die Integrität von homomorphen Chiffraten zu prüfen. Das in dieser Arbeit vorgestellte Verfahren definiert einen solchen Rahmen und leistet damit einen Beitrag, den Konflikt der Vertraulichkeit und Verfügbarkeit von Daten zu lösen.

10 Literaturverzeichnis

[1] R. Algesheimer. *Elliptische Kurven als alternatives Public Key-Verfahren im Homebanking-Standard HBCI*. diplom. de, 2000.

[2] J.-P. Aumasson, W. Meier, R. C.-W. Phan, and L. Henzen. *The Hash Function BLAKE*. Springer, 2014.

[3] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard. Practical multi-candidate election system. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 274–283. ACM, 2001.

[4] A. Beutelspacher. *Kryptologie*, volume 7. Springer, 1996.

[5] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *International Workshop on Public Key Cryptography*, pages 119–136. Springer, 2001.

[6] Data61 and CSIRO. Partially homomorphic encryption library for python, 2016. `https://pypi.python.org/pypi/phe`, zuletzt aufgerufen am: 10.12.2016.

[7] L. Dong and K. Chen. *Cryptographic protocol: security analysis based on trusted freshness*. Springer Science & Business Media, 2012.

[8] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 10–18. Springer, 1984.

[9] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of cryptology*, 1(2):77–94, 1988.

[10] M. Franz, B. Deiseroth, K. Hamacher, S. Jha, S. Katzenbeisser, and H. Schröder. Secure computations on non-integer values. In *2010 IEEE International Workshop on Information Forensics and Security*, pages 1–6. IEEE, 2010.

[11] K. Freiermuth, J. Hromkovic, L. Keller, and B. Steffen. *Einführung in die Kryptologie: Lehrbuch für Unterricht und Selbststudium*. Springer-Verlag, 2014.

[12] D. L. Gazzoni Filho and P. S. L. M. Barreto. Demonstrating data possession and uncheatable data transfer. *IACR Cryptology ePrint Archive*, 2006:150, 2006.

[13] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[14] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 291–304. ACM, 1985.

[15] S.-H. Heng and K. Kurosawa. *Provable Security: 4th International Conference, ProvSec 2010, Malaysia, October 13-15, 2010, Proceedings*, volume 6402. Springer SBM, 2010.

[16] M. N. Krohn, M. J. Freedman, and D. Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 226–240. IEEE, 2004.

[17] R. Küsters and T. Wilke. Moderne Kryptographie. *Vieweg+ Teubner*, 2011.

[18] L. Nöbel. *Sicherheitsaspekte kryptographischer Verfahren beim Homebanking*. PhD thesis, Universität Leipzig, DEUTSCHLAND, 2002.

[19] R. Oppliger. *Contemporary cryptography*. Artech House, 2011.

[20] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[21] P. Paillier and D. Pointcheval. Efficient public-key cryptosystems provably secure against active adversaries. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 165–179. Springer, 1999.

[22] J.-J. Quisquater, M. Quisquater, M. Quisquater, M. Quisquater, L. Guillou, M. A. Guillou, G. Guillou, A. Guillou, G. Guillou, and S. Guillou. How to explain zero-knowledge protocols to your children. In *Conference on the Theory and Application of Cryptology*, pages 628–631. Springer, 1989.

[23] M. P. S. Spitz and J. Swoboda. *Kryptographie und IT-Sicherheit*. Springer, 2011.

[24] S. A. Stuvel. Rsa library for python, 2016. `https://pypi.python.org/pypi/rsa`, zuletzt aufgerufen am: 10.12.2016.

[25] H. C. Van Tilborg and S. Jajodia. *Encyclopedia of cryptography and security*. Springer Science & Business Media, 2014.

[26] X. Yi, R. Paulet, and E. Bertino. *Homomorphic encryption and applications*. Springer, 2014.