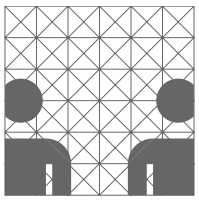


Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften
Department Informatik

Diplomarbeit

Information Hiding in XML-basierten Office-Dokumenten

Vorgelegt von:
Lars Westphal
Am Elisabethgehölz 4
20535 Hamburg

Matr.-Nr. 5224318
lars.westphal@informatik.uni-hamburg.de
Studiengang Wirtschaftsinformatik

genehmigtes Abgabedatum: 03. November 2008

Erstgutachter: Prof. Dr. Joachim Posegga, Department Informatik
Zweitgutachter: Prof. Dr. Kai von Luck, HAW Hamburg

Zusammenfassung

Mit der Zunahme vielfältiger E-Business-Anwendungen im Internet nimmt der Wert von Information in einigen Bereichen deutlich zu. Um die Autorenschaft digitalen Informationen nachweisen zu können, werden unter anderem digitale Wasserzeichen eingesetzt. Für Bild-, Audio- und Videodateien sind Verfahren dafür bereits seit den 1990ern etabliert. Für Office-Dokumente existieren bislang keine veröffentlichten Verfahren zur verdeckten Einbettung von Informationen. Diese Arbeit untersucht Möglichkeiten und Einschränkungen des Information Hiding für XML-basierte Office-Dokumente, insbesondere für das Open Document Format und Office Open XML; ihre Anwendbarkeit auf XML-basierte Dokumente wird unter Gesichtspunkten wie etwa Darstellungsgleichheit und Robustheit diskutiert. Darüber hinaus wird ein eigener Vorschlag vertiefend behandelt, der basierend auf einer Darstellungsrepräsentation von XML-Dokumenten aus der Anwendung formatierungsbasierter Wasserzeichen besteht.

Danksagungen

Ich danke Prof. Dr. Joachim Posegga für die Betreuung meiner Diplomarbeit unter den extrem schwierigen Bedingungen des Wechsels an die Universität Passau; ebenso Prof. Dr. Kai von Luck für die Betreuung in Hamburg; Henrich C. Pöhls, den die technische Betreuung wertvolle Zeit mit seiner Familie gekostet hat; Stephanie Knab für die denkbar beste Bürogemeinschaft; meinen Eltern Gudrun und Hans-Joachim Westphal für vielseitige und herzliche Unterstützung; Axel Sylvester für wertvolle Anregungen und Diskurs sowie Britta Westphal, Nils Bertling und ganz besonders Mirko Marquardt, die sich der undankbaren Aufgabe angenommen haben, dieser Arbeit den Fehlerteufel auszutreiben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1. Einleitung

Die stärkere Vernetzung von Informationssystemen hat in jüngerer Zeit zu vereinfachtem Informationsaustausch geführt. Mit dem Internet sind reichhaltige E-Business-Anwendungen entstanden. Im Business-to-Business-Sektor (B2B) Auktionsplattformen, Kundenformen und Online-Marktplätze, um nur einige Beispiele zu nennen. Sie ermöglichen Kunden die zeitnahe Kontaktaufnahme mit einer Vielzahl von Händlern und Herstellern. Die Dokumentation von Warenflüssen, technische Beschreibungen und Kundenkontakte sind Beispiele für Daten, die einen intrinsischen Wert haben, der losgelöst vom ursprünglich gehandelten Warenwert existiert. Diese Entwicklung führt zu neuen Problemen und Missbrauchspotentialen. Um diesen entgegenzutreten, werden in der Literatur unter anderem verschiedene Buyer-Seller-Protokolle diskutiert. Sie modellieren die beteiligten Parteien und Systeme und legen Abläufe fest, die einen sicheren Nachrichtenaustausch bei E-Business-Transaktionen ermöglichen sollen. Solche Protokolle beinhalten regelmäßig Verfahren, die Beziehungen zwischen Dokumenten und ihrem Ersteller herstellen, um nach der Veröffentlichung eines Dokuments den Urheber zuverlässig bestimmen zu können.

Eines dieser Verfahren ist die Einbettung digitaler Wasserzeichen. Wasserzeichen versprechen, dass sie – einmal in ein Dokument eingebracht – nicht entfernt werden können, ohne das Dokument zu beschädigen und somit wertlos zu machen. Darüber hinaus sind sie für einen nicht-eingeweihten Betrachter unsichtbar. Üblicherweise werden Wasserzeichen durch ein Modell von G. Simmons beschrieben, das sogenannte Gefangenenproblem mit unterschwelligem Kanal (*The prisoners' problem and the subliminal channel* [Sim84]): Alice und Bob werden inhaftiert und von einem Wärter bewacht. Er erlaubt ihnen, sich unverschlüsselte Nachrichten zu schreiben. Alice und Bob können sich nur über einen verdeckten Kanal verständigen, indem sie Informationen austauschen, die beim Wärter keinen Verdacht erregen. Dieser verdeckte Kanal, die Methode, die Alice und Bob zur Übermittlung ihrer geheimen Nachrichten benutzen, ist der Forschungsgegenstand des Information Hiding – jener Disziplin, die als Teilbereich auch die Untersuchung digitaler Wasserzeichen einschließt. Die Lösungen, die das Information Hiding liefert, sind stark an das jeweilige Medium gebunden, in dem Informationen eingebettet werden. Daher lassen sich Erkenntnisse über existierende Methoden nur bedingt auf andere Medien übertragen. Zudem setzt die Einbettung gute Kenntnisse des jeweiligen Mediums voraus.

Bislang wurden vorwiegend Methoden für die Einbettung von Wasserzeichen in Audio-, Grafik- und Videoformaten entwickelt. In einem E-Marketplace werden allerdings andere Dokumentarten verwendet. Eine Möglichkeit ist, Office-Dokumente digital zu versenden. Diese werden im Büro-Alltag ohnehin vielseitig eingesetzt [Hil08]. Es bietet sich daher an, ein solches, bereits etabliertes Format einzusetzen, um kostenintensive Einweisungen und Umschulungen zu reduzieren. Es gibt derzeit zwei offene Standards für Office-Dokumente: Das Open Document Format for Office Applications (ODF) ist ein XML-basierter Dokumentenstandard, der seit der Veröffentlichung von OpenOffice [OAS07] Version 2 eingesetzt wird. Version 1.0 des Standards wurde zum ISO-Standard erhoben [ISO06]. Des weiteren hat Microsoft ebenfalls ein XML-basiertes Dokumentenformat namens Office Open XML [Mic07] vorgestellt, das von der ECMA als Standard ECMA-376 [ECM06] bestätigt wurde. Beide Dokumentenformate erlauben nicht nur die einfache Untersuchung des darunter liegenden XML-Codes. Es handelt sich dabei auch offene Standards ohne Lizenzgebühren. Das rückt sie stärker in den Fokus wissenschaftlicher Untersuchungen – und lädt leider auch zu bösartigen Manipulationen ein.

Die Standards beschreiben die zulässigen Elemente im Dokument sehr detailliert (Containerformat, Quelldateien, XML-Auszeichnungen). Dennoch bleibt an einigen Stellen Raum für unterschiedliche Varianten bestehen. So können mehrere unterschiedliche XML-Auszeichnungen [BPSM$^+$06] zu derselben Formatierung im Dokument führen. Dann können diese äquivalenten Eingaben als informationstragende Kennzeichen interpretiert werden. Damit entsteht eine Bandbreite, um zusätzliche Informationen im Dokument zu speichern. Da solche Veränderungen nur anhand der XML-basierten Dokumentquellen bemerkt werden können, bleiben sie den meisten Benutzern verborgen. Kann dieser Umstand helfen, ein Verfahren zu entwickeln, um Informationen gezielt in Office-Dokumenten zu verbergen, wie es bei Audio und Video mit Wasserzeichen möglich ist? Eine besondere Herausforderung ist, dass die Office-Dokumente komplexe Formate sind, die zusätzlich zu XML noch eigene Eigenarten mitbringen. Insbesondere, weil XML für die Dokumente nur zur persistenten Speicherung der Daten benutzt wird. Während der Laufzeit werden die Daten anders (effizienter) im Speicher repräsentiert und erst beim Speichern in eine Datei transformiert. Wasserzeichen müssen diese Transformation überstehen können.

Im Zusammenhang mit Information Hiding ergeben sich verschiedene Herausforderungen. Diese Probleme sind nicht neu und für bestehende Information Hiding-Systeme wie Audio-Watermarking bereits untersucht worden. Das bedeutet leider nicht, dass die für diese Formate entwickelten Lösungen unmittelbar auf XML übertragbar sind. Welche technischen Eigenschaften muss ein Verfahren haben, um eine darstellungsäquivalente Einbettung von Informationen in XML-basierte Office-Dokumente zu ermöglichen? Darstellungsäquivalenz bedeutet, dass zwei Dokumente zwar aus unterschiedlichen Daten bestehen können, sie aber zur gleichen gerenderten Ansicht führen.

Zu den wesentlichen Problemen bei Wasserzeichen zählt ihre Robustheit. Ein Wasserzeichen, das eingebettet ist, sollte möglichst eine Reihe von Veränderungen am Cover-Objekt überstehen. Wasserzeichen in Bildern werden oft in eine DCT (Diskrete Cosinus-Transformation) des Bildes eingebettet, damit Bildmanipulationen wie Vergrößern oder Zuschneiden das Wasserzeichen nicht entfernen. Gibt es eine Alternative zu dieser Transformation für XML? Besteht die Möglichkeit, Wasserzeichen so einzubetten, dass sie auch bei einer nachträglichen Bearbeitung des Dokuments nicht zerstört werden?

Ein weiteres Problem bei Wasserzeichen ist, dass die einzelnen Bits eines eingebetteten Wasserzeichens beim Auslesen in die richtige Reihenfolge gebracht werden müssen. Ansonsten kann das eingebettete Wasserzeichen nicht erfolgreich mit dem Original verglichen werden. Zu einer Veränderung der Reihenfolge kann es zum einen kommen, wenn im Dokument Teile verschoben werden, die Bits des Wasserzeichens enthalten. Diese Synchronisation muss beim Einbettungsschema auch deshalb beachtet werden, damit Markierungen nicht einfach von Unbefugten ausgelesen werden können. Bringt man die Markierungen so in das Dokument ein, dass die Reihenfolge nicht trivial zu bestimmen ist (sofern man überhaupt in der Lage ist, die Markierungen als solche zu erkennen), stellt das eine zusätzliche Hürde für einen Angreifer dar.

Welche Rahmenbedingungen müssen außer dem technischen Verfahren erfüllt sein, damit eine Anwendung sinnvoll Information Hiding in XML-basierten Dokumenten einsetzen kann? In dieser Arbeit soll anhand eines fiktiven, jedoch realen Anforderungen entsprechenden Szenarios betrachtet werden, wie eine Anwendung von Information Hiding in XML-Dokumenten nutzbringend sein kann. Anhand eines Szenarios lässt sich auch anschaulicher diskutieren, was sind die sicherheitsrelevanten Eigenschaften sind, die ein Einbettungsverfahren erfüllen muss.

Die vorliegende Arbeit gliedert sich im Weiteren wie folgt: In Kapitel 2 werden grundlegende Begriffe und Konzepte erklärt, die für das Verständnis der weiteren Arbeit erforderlich sind. Dazu gehören die Beschreibungen der Dokumentenformate sowie eine Klassifikation von Information Hiding.

Kapitel 3 enthält das Szenario, in welchem beispielhaft der Anwendungsbereich geschildert wird. Dieses Szenario beschreibt eine E-Business Anwendung mit Käufer- und Verkäuferrollen, in der digitale Dokumente eingesetzt werden.

In Kapitel 4 werden bereits bestehende Verfahren in Bezug auf ihre Anwendbarkeit auf das entwickelte Szenario hin untersucht. Dazu gehören Verfahren für gedruckten Text, Datenbanken und XML, sowie linguistische Verfahren; aber auch Methoden für Bilder und Multimedia.

Kapitel 5 enthält einen eigenen Entwurf eines möglichen Entwurfes und dessen prototypische Umsetzung. Diese Umsetzung basiert dabei auf einem auf XML angepassten Schema für linguistische Wasserzeichen.

Kapitel 6 fasst schließlich die in dieser Arbeit gewonnenen Erkentnisse zusammen und gibt einen Ausblick auf mögliche weiterführende Untersuchungen.

2. Grundlagen

Dieses Kapitel führt zunächst in die Begrifflichkeiten und Konzepte des Information Hiding ein, sowie sie für diese Arbeit relevant sind. Anschließend folgt eine kurze Beschreibung der Dokumentenformate, die in dieser Arbeit betrachtet werden.

2.1. Allgemeine Definitionen

Im ersten Abschnitt dieses Kapitels folgen allgemeine Definitionen zu Datensicherheit, Textverarbeitung und Sprachverarbeitung die im Kontext des Information Hiding in der Arbeit verwendet werden.

2.1.1. Informationen und Daten

Nach Rich Smith [Smi08] besteht ein formaler und praktischer Unterschied zwischen Informationen und Daten:

Daten: Daten sind logisch gruppierte Informationseinheiten.

Informationen: Informationen sind eine subjektive Interpretation von Daten.

Daraus folgt, dass unterschiedliche Daten die gleichen Informationen enthalten können. Veränderungen der Daten schließen nicht zwingend Änderungen der Informationen ein.

2.1.2. Sicherheitsziele

Pfleeger [PP06] führt folgende Sicherheitsziele auf:

Vertraulichkeit: Nur authorisierte Personen oder Systeme können auf geschütze Daten zugreifen.

Integrität: Die Daten eines Systems werden keinen versehentlichen oder absichtlichen Veränderungen oder Zerstörungen unterworfen.

Verfügbarkeit: Daten und Dienste sind in einem benutzbaren Zustand und in ausreichender Kapazität vorhanden. Außerdem kann auf sie in einem akzeptablen Zeitrahmen zugegriffen werden.

2.1.3. Angriffe

Einer freien Übersetzung der Definition aus dem RFC 2828 [Shi00] folgend sind Angriffe auf die Sicherheit eines Systems ein bewusster und absichtlicher Versuch, die Sicherheit eines Systems zu umgehen und die Sicherheitsrichtlinien eines Systems zu verletzen. Es werden zudem unterschieden:

Passive Angriffe: Ein passiver Angriff zielt darauf ab, Erkenntnisse von einem System zu erlangen oder Nutzen aus Informationen über ein System zu ziehen, ohne die Ressourcen zu beeinträchtigen.

Aktive Angriffe: Ein aktiver Angriff ist darauf ausgerichtet, Systemressourcen zu verändern oder ihre Arbeit zu beeinträchtigen.

2.1.4. Maßnahmen für Sicherheit

Nach [CCI91, Anhang A.4] können Sicherheitsmaßnahmen in drei Kategorien eingeteilt werden:

- Prevention (Verhinderung),
- Detection (Erkennung) und
- Recovery (Schadensbeseitigung).

2.1.5. Look-Up-Table (Nachschlagetabelle)

Eine Nachschlagetabelle, zumeist unter dem englischen Begriff Look-Up-Table auch im Deutschen bekannt, ist eine Datenstruktur, in der zur Laufzeit benötigte Informationen vorab erzeugt und abgelegt werden. Zumeist werden sie aus Perfomanzgründen eingesetzt, wenn z. B. die Informationen sonst aufwändig berechnet werden müssten [Wik08].

In dieser Arbeit werden Look-Up-Tables verwendet, um darstellungsäquivalente XML-Teilbäume zu speichern, zu indizieren und die Varianten mit einem Bit-Wert zu belegen.

2.1.6. Schriftattribut

Schriftattribute sind Eigenschaften eines Schriftzeichens. Beispiele sind „gefettete“, kursive und unterstrichene Schrift. Diese können heute meist beliebig kombiniert werden. Im klassischen Bleisatz gab es aber so genannte Schriftschnitte, für die getrennte Lettern hergestellt wurden. Auch bei den heute verwendeten Vektor-Schriftarten sind nicht für alle Kombinationen aus Attributen entsprechende Schriftschnitte verfügbar.

2.1.7. Style

Ein „Style“ oder auch „Stil“ oder „Formatvorlage“ ist eine logische Formatierung von Textteilen. Dabei wird einem Style einen eindeutigen Bezeichner und eine Reihe von Formatierungen zugeordnet.

Es gibt Absatzstyles und Textstyles. Dies hängt damit zusammen, dass es Formatierungen gibt, die für Absätze anwendbar sind, aber nicht auf beliebige Textstellen, z. B. Einzüge, Ausrichtung und Absatzabstände. Textformatierungen sind jedoch auf Absatzstyles anwendbar.

2.1.8. Dokumentinhalt

Im Gegensatz zum Inhalt des Containers, in dem die enthaltenen Dateien festgelegt sind, bezieht sich der Inhalt des Dokuments auf die sachliche Ebene. Am besten ist der Dokumentinhalt durch Aufzählung seiner Elemente zu beschreiben. Dies können unter anderem (abhängig vom Dokumenttyp) sein:

- Texte
- Bilder
- Tabellen
- Fussnoten
- Verzeichnisse
- usw.

2.1.9. Darstellungsgleichheit

Die Darstellung eines Dokuments ist eine gerenderte Ansicht, beispielsweise am Bildschirm in Form einer WYSIWYG-Ansicht (What You See Is What You Get) in einem Editor oder bei einem Ausdruck. Zwei Dokumente sind darstellungsgleich, wenn beide Dokumente die gleichen Informationen enthalten und zu der gleichen gerenderten Ansicht führen, der aber unterschiedliche Formatierungsanweisungen zugrunde liegen können.

2.1.10. Bedeutungsrepräsentation von Text

Bei einer Bedeutungsrepräsentation von Text (Text-Meaning-Representation oder TMR) handelt es sich um eine sprachunabhängige Beschreibung der Informationen in natürlichsprachlichem Text[1], wie sie für maschinelle Sprachverarbeitung (Natural Language Processing, NLP) [ON92] verwendet wird. Gleichbedeutenden Wörter werden dabei zu einem Konzept gruppiert, das entweder einen eindeutigen Bezeichner oder einfach eine eindeutige Nummer erhält. Zum Beispiel könnte man aus den Begriffen (sein, existieren) ein Konzept bilden, das die Nummer „1“ zugewiesen bekommt. Andere Wortgruppen wie (hinweisen, aufzeigen, darstellen) würden einen anderen Wert wie „2“ zugewiesen bekommen und so weiter. Jetzt kann die Bedeutung eines Wortes w durch eine Funktion $M\left(w\right)$ ermittelt werden (Das M steht an dieser Stelle für „Meaning“). Sie erhält ein Wort als Eingabe und gibt dessen Bedeutung als numerischen Wert aus. Somit können Sätze, die sich durch einzelne Wörter unterscheiden, die aber die gleiche Bedeutung haben, einfach maschinell verglichen werden[2]. Ein TMR kann eine Datenbank oder Look-Up-Table (siehe Abschnitt 2.1.5) sein, in der den Wörtern entsprechende Bedeutungen zugeordnet sind.

2.1.11. Darstellungsrepräsentation von XML

Analog zur Bedeutungsrepräsentation von Text kann man für XML-basierte Office-Dokumente eine Darstellungsrepräsentation definieren. Hier werden darstellungsgleiche (siehe Abschnitt 2.1.9) Styles (Abschnitt 2.1.7) oder XML-Teilbäume in einer Look-Up-Table (Abschnitt 2.1.5) gespeichert. Wie oben ist eine Funktion $M\left(s_i\right)$

1 Das bedeutet, dass die Bedeutungsrepräsentation sowohl unabhängig von der Wahl eines Synonyms ist, als auch Wörtern einer anderen Sprache (wie Deutsch oder Englisch).

2 Es ist für Anwendungen des NLP wie maschinelle Übersetzung nicht ausreichend, wenn nur Wörter im TMR erfasst werden. Dort müssen auch syntaktische Zusammenhänge zwischen Wörtern verarbeitet werden. Deshalb wird durch übergeordnete Strukturen (so genannte Propositions) zusätzlich die Satzbedeutung in einem TMR erfasst. In dieser Arbeit wird aber keine Sprachverarbeitung durchgeführt, so dass weiterführende Erklärungen nicht zweckdienlich sind.

dann so definiert, dass die einem Teilbaum zugeordneten Bedeutung als numerischer Wert ausgegeben wird.

2.1.12. Editierabstand

Der *Editierabstand* (Edit-Distance) oder *Levenshtein-Distanz* ist ein Maß für die Veränderung von zwei Zeichenketten. Er gibt das Minimum der Zeichenoperationen an, die zur Transformation von einer Zeichenkette in die andere führen [Lev66] (wobei diese aus Einfügen, Löschen und Ersetzen eines einzelnen Zeichens bestehen). Mit diesem Editierabstand ist es also möglich, den Unterschied zwischen zwei Zeichenketten anzugeben. Mathematisch lässt sich die Levenshtein-Distanz mit dynamischer Optimierung durch den so genannten Wagner-Fischer-Algorithmus [WF74] oder den Hirschberg-Algorithmus [Hir75] lösen.

2.1.13. Buyer-Seller-Protokolle

Bei E-Business-Anwendungen, in denen elektronische Dokumente über das Internet verteilt werden, können digitale Fingerprints und Wasserzeichen nicht alleine als Beweis der Autorenschaft und zur Verfolgung der Dokumente dienen. Ohne ein geeignetes Protokoll, dass festlegt, welche der beteiligten Parteien an welcher Stelle des Verfahrens ein Wasserzeichen oder Fingerprint in das Dokument einbringen muss, ist das Verfahren insgesamt unbrauchbar. Zum Beispiel reicht es nicht aus, wenn ein Verfahren vorsieht, dass Autoren ein Dokument optional mit einem Wasserzeichen versehen können. Dieses kann nicht als Beweis der Autorenschaft dienen, weil nicht verhindert wird, dass ein Angreifer sein Wasserzeichen in Dokumente einbringt, die er nicht selber verfasst hat.

Die typischen Parteien in E-Business-Anwendungen sind Käufer und Verkäufer (oder englisch Buyer und Seller). Eine Reihe so genannter Buyer-Seller-Protokolle (zum Beispiel [MW01]) wurden veröffentlicht, um für solche Verfahren zu gewährleisten, dass sie auch dann noch funktionieren, wenn man den teilnehmenden Parteien ein gewisses kriminelles Interesse unterstellt und nicht von allen Parteien absolute Fairness erwartet.

2.2. Information Hiding

Einer Klassifikation von Peticolas et al. [KP00] zufolge ist Information Hiding ein Oberbegriff, der Covert Channels (verdeckte Kanäle), Steganographie, Anonymi-

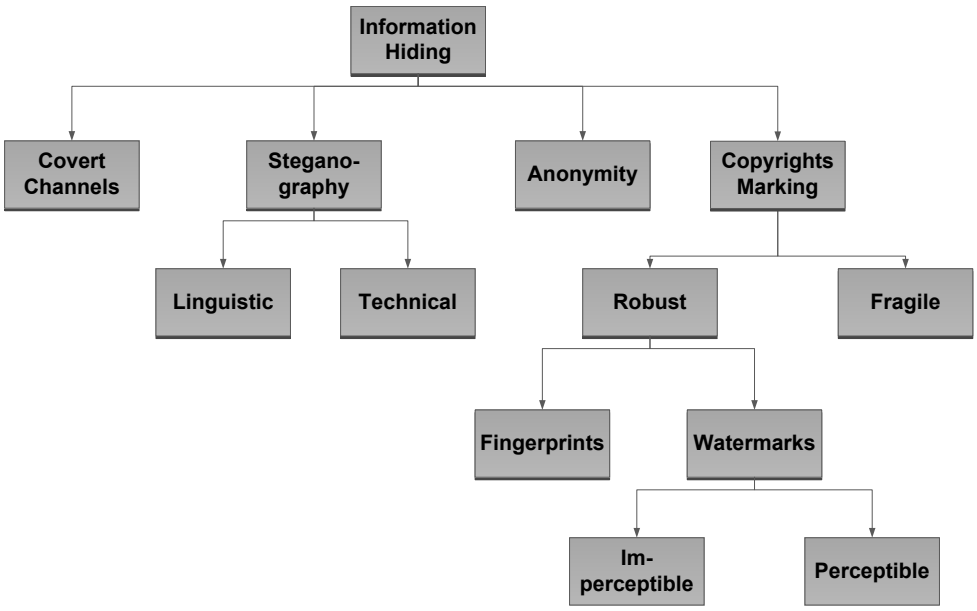


Abbildung 2.1.: *Klassifikation von Information Hiding nach Peticolas et al. [KP00]*

tät und Copyrights Marking (Urheberrechtskennzeichnung) umfasst (siehe Abbildung 2.1).

Information Hiding bedeutet, dass beliebige Informationen in einer anderen Nachricht verborgen werden. Die Kategorien ergeben sich aus dem Sinn oder Ziel dieses Vorgangs. Die Bezeichnungen haben sich teilweise aus historischen Entwicklungen ergeben und werden in der Fachliteratur teilweise auch anders verwendet – so werden Information Hiding und Steganographie bisweilen synonym gebraucht. Gelegentlich heißt Information Hiding aber auch Data Hiding.

2.2.1. Information Hiding Terminology

Auf dem First International Workshop on Information Hiding im Jahr 1996 haben sich die Teilnehmer auf eine Terminologie geeinigt [Pfi96], die seitdem verwendet wird. Für formale Definitionen sind Quellen wie [AKS03] empfehlenswert.

Embedded Data: Informationen, die im Dokument verborgen werden (eingebettete Daten).

Cover-Document: Eine unveränderte Originalversion eines Dokuments.

Stego-Dokument: Ein Dokument, das die Embedded Data enthält.

Stego- Schlüssel (Stego Key): Ein geheimer Wert, der als Parameter im Einbettungsverfahren benutzt wird (nach [Pfi96]). Üblicherweise wird derselbe Schlüssel auch benutzt, um die Informationen wieder herauszulesen.

Ein Dokument kann im allgemeinen Kontext von Information Hiding ein Text, Bild, Video oder Audio sein. Es soll daher in der Terminologie durch das entsprechende Wort ersetzt werden, was im Fall dieser Arbeit aber nicht notwendig ist. Pfitzmann [Pfi96] gibt außerdem Vorschläge zur Benennung der Verfahren:

Einbettung (Embedding): Verbergen von Embedded Data in einem Cover Document.

Einbetter (Embedder): Eine Entität oder Person, die eine Einbettung durchführt, heißt Embedder.

Extraktion (Extraction): Wiederherstellen der Embedded Data aus dem Stego-Document wird als Extracting (Herauslesen) bezeichnet. Dies ergibt sowohl die Embedded Data als auch das Originaldokument.

Extraktor (Extractor): Eine Entität oder Person, die das Herauslesen ausführt, heißt Extractor.

Verifikation: Der Prozess der Verifikation vergleicht eingebettete Daten mit Vergleichsdaten. Er liefert als Ergebnis *wahr*, wenn Eingebettete Daten identisch sind mit den Vergleichsdaten, ansonsten *falsch*.

Stegoanalyst: Eine Entität oder Person, die ein Stego-Document analysiert (mit der Absicht, das Verfahren zu überwinden, den Schlüssel herauszufinden oder – ohne im Besitzt dieser zu sein – die Embedded Data zu gewinnen) ist ein Stegoanalyst.

2.2.2. Anonymität

Das Ziel von Anonymität ist, die tatsächliche Identität einer Person zu verbergen. Daher werden keine Informationen zugänglich gemacht (oder verborgen), die diese enthüllen könnten. Es gibt Anwendungen von Information Hiding, die in diesem Zusammenhang denkbar sind:

So werden beispielsweise Röntgenbilder mit verborgenen Informationen versehen, die die Patientendaten enthalten. Dies hängt damit zusammen, dass diese Daten zwar für die Versicherung erhoben und festgehalten werden müssen, jedoch nicht alle Beteiligten, die mit den Aufnahmen arbeiten, die Daten einsehen dürfen. Eine ähnliche Anwendung wäre auch für Office-Dokumente denkbar.

Für diese Arbeit spielt Anonymität nur eine untergeordnete Rolle und wird nicht weiter betrachtet.

2.2.3. Steganographie

Obwohl Steganographie wörtlich mit „verdecktes Schreiben“ (von $\sigma\tau\epsilon\gamma\alpha\nu o\varsigma$ und $\gamma\rho\alpha\phi\epsilon\iota\nu$) übersetzt werden kann, wird es heute teilweise synonym zu Information Hiding benutzt. Anstatt eine Kommunikation für einen Abhörer unlesbar zu machen, wird hierbei die Kommunikation an sich verborgen. Es wird ein Kommunikationskanal zur Tarnung verwendet, in dem der eigentliche verdeckte Kanal enthalten ist. Wenn es sich eher um ein einziges Objekt handelt als einen Medienstrom, kann es auch als Cover-Objekt (oder Tarnobjekt) bezeichnet werden.

2.2.3.1. Pure Steganographie

Unter purer Steganographie versteht man Verfahren, bei denen die Informationen im Stego-Objekt so eingebettet werden, dass diese ausschließlich mit Kenntnis des richtigen Verfahrens wieder ausgelesen werden können. Pure Steganographie verletzt somit Kerckhoffs' Prinzip [Ker83], das besagt, dass die Sicherheit eines Verschlüsselungsverfahrens nicht von der Geheimhaltung des Verfahrens, sondern nur von einem Schlüssel abhängen darf[3].

2.2.3.2. Steganographie mit geheimen Schlüssel

Damit das Verfahren nicht geheim gehalten werden muss, sollte es von einem Geheimnis oder Schlüssel abhängen. In diesem Zusammenhang bedeutet Secret-Key-Steganographie, dass beide Parteien das gleiche Geheimnis verwenden. Es wird z. B. der gleiche Schlüssel zum Einbetten von Informationen wie zum Auslesen benutzt. Analog zur symmetrischen Verschlüsselung muss dieser Schlüssel beiden Parteien vorher bekannt sein. Er muss also zunächst sicher übertragen und verwahrt werden, was nicht trivial ist. Insbesondere bei steigender Anzahl von Teilnehmern führt die Anzahl der Schlüssel, die verwahrt und verwaltet werden müssen (gegebenenfalls müssen auch kompromittierte oder veraltete Schlüssel gelöscht werden) zu arbeitsintensiven Verwaltungsaufgaben: Für n Teilnehmer werden 2^n Schlüssel benötigt.

3 Kerckhoffs hat dies für kryptographische Verfahren gefordert, aber es ist leicht einzusehen, dass es auch für Steganographie gilt.

Hat der Empfänger die Möglichkeit, mit dem Schlüssel eingebettete Informationen zu erkennen, bedeutet das in der Regel auch, dass er sie entfernen und ersetzen kann. Dies kann ohne zusätzliche Maßnahmen nicht verhindert werden.

2.2.3.3. Steganographie mit öffentlichem Schüssel

Bei Public-Key-Steganography werden analog zur Public-Key-Verschlüsselung unterschiedliche Schlüssel zum Einbetten und Auslesen der Informationen verwendet. Während in der Kryptographie Algorithmen für asymmetrische Verschlüsselung schon länger bekannt sind [RSA83], gibt es erst seit wenigen Jahren für die Steganographie brauchbare Algorithmen. Diese sind zudem überwiegend für Watermarking ausgelegt, so dass unklar ist, inwiefern sie auf Information Hiding im Allgemeinen angewendet werden können. Ein solcher Ansatz benötigt eine Public Key Infrastructure (PKI).

2.2.4. Covert Channel

Die Definition eines Covert Channel entstammt einem technischen Bericht von Buster Lampson, „A note on the confinement problem“ [Lam73], in dem diskutiert wird, wie für ein beliebiges Programm verhindert werden kann, dass es Daten „leckt“, also sie ausgibt, ohne dass dies dem Zweck der Anwendung dient. Lampson zählt die Kanäle auf, über die eine Anwendung Daten weitergeben kann und führt dabei als einen Kanal den sogenannten Covert Channel ein: Ein verdeckter Kanal, der eigentlich überhaupt nicht für Informationsübertragung ausgelegt ist. Der Begriff wurde später unter anderem im Orange Book [DOD85] zur Evaluation vertrauenswürdiger Systeme verwendet. Ein berühmtes Beispiel für einen verdeckten Kanal sind die Fast-Food-Lieferungen an das Weiße Haus in Washington: diese nähmen angeblich zu, wenn der Krisenstab länger tage. In der Praxis geben viele Vorgänge Rückschlüsse zu anderen Informationen – wenn aus einem Schornstein Rauch aufsteigt, kann man davon ausgehen, dass geheizt wird. Der nächste Schritt, diesen Umstand bewusst zu nutzen, ist dann nicht weit – man denke an den weißen und schwarzen Rauch, der über der sixtinischen Kapelle bei der Papstwahl aufsteigt. Deutlich weitreichendere Möglichkeiten der Übertragung eröffnen sich, wenn die Bandbreite des verdeckten Kanals zunimmt. Dies kann entweder durch eine Vielzahl von Objekten mit geringer Bandbreite (wie dies bei Netzwerk-Paketen der Fall ist) oder natürlich auch durch weniger Objekte mit entsprechend höherer Bandbreite geschehen.

Im Unterschied zu anderen Unterarten von Information Hiding besteht hier kein gewollter Bezug zwischen dem Cover-Objekt und den Embedded-Daten. Während Watermarking beispielsweise oft verwendet wird, um die Autorenschaft eines Bildes zu

belegen, und die Embedded-Daten dort also die Voraussetzung für die Funktion dieses Nachweises sind, liegt der Sinn eines verdeckten Kanals darin, dass die verdeckte Nachricht (also die Embedded-Daten) übertragen werden; diese Übertragung dient nur dem Selbstzweck. Dementsprechend werden die Cover-Objekte zwar auch beliebig gewählt, aber möglichst so, dass ihre Übertragung „natürlich“ oder „normal“ erscheint und keinen Verdacht erregt. Man benutzt also in der Regel solche Objekte, die ohnehin übertragen werden.

2.2.5. Copyrights Marking

Copyrights Marking umfasst Maßnahmen aus dem Information Hiding, die zur Durchsetzung von Kopierschutzmechanismen beitragen. Dabei werden Dokumente mit Markierungen versehen, die es ermöglichen, den rechtmäßigen Urheber eines Dokumentes zu bestimmen.

2.2.5.1. Watermarking

Watermarking ist das Verfahren, mit dem Wasserzeichen in ein Dokument eingebracht werden. Es handelt sich aber nicht zwingend um eine $1:1$-Beziehung: Oftmals wird ein Wasserzeichen für mehrere Dokumente benutzt.

Wie Peticolas beschreibt, ist die Information, die in einem Wasserzeichen eingebettet wird, immer mit dem Cover-Objekt assoziiert, entweder als Eigentumsnachweis oder um das Objekt selber vor Änderungen zu schützen [KP00].

2.2.5.1.1. Blinde Wasserzeichen Ein Wasserzeichen heißt *blind*, wenn das Wasserzeichen aus dem Stego-Dokument extrahiert werden kann, ohne dass dafür das ursprüngliche Cover-Dokument benötigt wird.

2.2.5.1.2. Robuste Wasserzeichen Ein Wasserzeichen ist *robust*, wenn das Stego-Dokument geändert werden kann, ohne dass das Wasserzeichen dabei verloren geht oder abgeschwächt wird. Diese Änderungen bewegen sich innerhalb dokumentimmanenter Grenzen: Sind die Eingriffe in das Dokument so massiv, dass das Dokument selbst beschädigt wird, verliert das Dokument für einen Angreifer seinen Wert. Im Fall maximaler Robustheit sind Wasserzeichen *persistent*; In der Praxis lässt sich eine solche Persistenz der Wasserzeichen nicht erreichen, weil dies zu Einschränkung von üblichen Benutzungsweisen des Dokuments führen würde.

Werden Wasserzeichen oder Fingerprints in ein Dokument eingebettet, die bei Änderungen am Dokument verloren gehen, so heißen diese *nicht robust* oder *flüchtig*. Extrem flüchtige Wasserzeichen gehen selbst bei geringen Änderungen verloren.

2.2.5.1.3. Asymmetrische Wasserzeichen Beim Asymmetrischen Watermarking werden – analog zur Steganographie mit öffentlichem Schlüssel – anstelle desselben Schlüssels zur Einbettung und Extraktion unterschiedliche Schlüssel (oder Geheimnisse) für diese beiden Schritte benutzt. Dies wird überhaupt nur von einigen speziellen Verfahren für bestimmte Cover-Medien ermöglicht.

2.2.5.1.4. Synchronisationsproblem Li et al. [LGJ04] weisen darauf hin, dass ein wesentliches Problem bei der Einbettung und Extraktion von Wasserzeichen die sogenannte Synchronisation ist. Synchronisation bezeichnet dabei: Die Reihenfolge beim Einbetten von Markierungen eines Wasserzeichens im Cover-Dokument muss identisch mit der Reihenfolge beim Auslesen der Markierungen sein.

2.2.6. Distribution Tracking

Bei *Distribution Tracking* wird versucht zu verfolgen, welcher Empfänger ein Dokument an dritte weiter gibt. Ein gängiges Mittel zum Distribution Tracking ist das Fingerprinting von Dokumenten (siehe nächster Abschnitt).

2.2.7. Fingerprinting

Beim Fingerprinting wird in ein Cover-Dokument n-fach kopiert und in jede der Kopien je ein unterscheidbares Wasserzeichen eingebettet, das dem Empfänger zuzuordnen ist. Dieser so genannte Fingerprint oder Fingerabdruck ist somit nicht mit dem Autor, sondern dem Empfänger des Dokuments assoziiert.

2.2.8. Mathematische Symbole

Werden die obigen Begriffe in Formeln oder ähnlicher Darstellung verwendet, werden dafür folgende Symbole benutzt:

K**:** (Haupt-) Schlüssel.

K_D**:** Dokumentschlüssel.

$M(s)$: Bedeutungsfunktion.

W: Wasserzeichen.

2.2.9. Hash-Funktion

Eine Hash-Funktion ist eine nicht-umkehrbare Abbildung. Meistens wird dabei ein größerer Definitionsbereich auf einen kleineren Bildbereich abgebildet. Der Sinn liegt darin, einen eindeutigen, nichtumkehrbaren Wert (Identifikator) zu einer Eingabe zu bestimmen. Wichtig ist die Unterscheidung von kryptographischen und robusten Hash-Funktionen, da beide vollkommen unterschiedliche Eigenschaften und Anwendungsfelder haben.

2.2.9.1. Kryptographische Hash-Funktionen

Kryptographische Hash-Funktionen weisen dabei zusätzliche Eigenschaften auf, die gerade sicherstellen sollen, dass sich der Hash-Wert auf jeden Fall schon bei einer geringfügigen Änderung der Eingabe deutlich ändert. Ansonsten werden kryptographische Hash-Funktionen als bekannt voraus gesetzt.

2.2.9.2. Robuste Hash-Funktionen

Im Gegensatz zu kryptographischen Hash-Funktionen erzeugen robuste Hash-Funktionen nach Fridrich [FG00] einen Hash-Wert einer Eingabe derart, dass folgende Eigenschaften gelten:

1. Es ist leicht, den Hash-Wert zu einer gegebenen Eingabe zu berechnen,
2. es ist rechnerisch unmöglich, eine Eingabe zu erzeugen, die zu einem bestimmten Hash-Wert führt,
3. es ist schwer, zwei erkennbar unterschiedliche Eingaben zu erzeugen, die zu demselben Hash-Wert führen, und
4. nur eine deutlich wahrnehmbare Änderung an der Eingabe führt zu einer Veränderung im Hash-Wert.

Es ist also gerade erwünscht, dass leichte Änderungen an der Eingabe keine Auswirkungen auf den Hash-Wert haben.

2.2.10. Nachweis von Manipulationen

In einem Dokument verborgene Informationen können auch genutzt werden, um Manipulationen am Dokument nachzuweisen. Dies funktioniert relativ einfach mit flüchtigen Wasserzeichen oder Fingerprints, die abhängig vom Dokumentinhalt erzeugt werden. Da flüchtige Markierungen nach Änderungen bereits aus dem Dokument entfernt oder zumindest abgeschwächt werden, verläuft eine anschließende Verifikation nicht mehr erfolgreich.

2.2.11. Schichtenansatz

Um die Kombination von Methoden für Wasserzeichen und Kryptographie zu verbessern, haben Cox et al. [CDF06] ein Modell vorgestellt, das durch das System Interconnections Model [Zim80] inspiriert wurde. Sie warnen davor, dass die Verbindung von kryptographischen und Watermarking-Methoden ohne saubere Modellierung keinen zusätzlichen Sicherheitsgewinn mit sich bringen. Um dem zu begegnen, empfehlen die Autoren, einen Schichtenansatz zu verwenden, ähnlich wie beim ISO/OSI-Modell. Zunächst wird dabei eine Ebene modelliert, die die Funktionen des Information Hiding abbildet. Dann, auf einer zweiten Ebene, können die Vorgänge für Kryptographie dargestellt werden. Beide Ebenen werden sauber getrennt, Informationen werden zur Verarbeitung von einer Ebene zur nächsten „weitergereicht“. Dieser Schichtenansatz ist deshalb sinnvoll, weil laut Cox et al. bereits mehrere derzeitige Fingerprinting-Ansätze ein ähnliches Schichtenkonzept verfolgen. Außerdem können für beide Ebenen Standard-Anwendungen weiterverwendet werden.

2.2.12. Einbettungs-Schema

Ein Einbettungs-Schema (Embedding Scheme) legt fest, welche Informationen an welchen Stellen in einem Cover-Objekt eingebettet werden. Zudem kann es angeben, welche Schlüssel verwendet werden und mit welchen Algorithmen diese erzeugt werden. In der Regel gehört zu einem Einbettungs-Schema nicht nur eine Beschreibung der Vorgehensweise bei der Einbettung, sondern auch bei der Extraktion und Verifikation (diese beiden Schritte werden in den Darstellungen oft zusammengefasst). Das Schema kann in einigen Bereichen unabhängig vom tatsächlichen Cover-Medium sein, allerdings ist dies nicht immer der Fall.

2.2.13. Arten von Angriffen auf Wasserzeichen

In der Literatur werden Angriffe auf Information Hiding meistens für Wasserzeichen, oft beispielhaft an Bildern, beschrieben. Die Angriffe lassen sich aber grundsätzlich auf andere Bereiche übertragen. Eine gute Übersicht dieser Art bietet die Arbeit von Voloshynovskiy et al. [VPP+01], deren Kategorien im folgenden für XML-Dokumente adaptiert werden.

2.2.13.1. Entfernungsangriffe

Entfernungsangriffe (Removal Attacks) werden mit der Absicht durchgeführt, ein Wasserzeichen oder eingebettete Informationen vollständig zu entfernen. Dabei wird nicht versucht, das Verfahren an sich zu knacken oder die Schlüssel anzugreifen. Vielmehr geht es darum, durch Operationen am Cover-Objekt die Bandbreite für die Einbettung zu entfernen oder mit eigenen Informationen (auch Rauschen) zu überschreiben. Voloshynovskiy [VPP+01] führt Beispielangriffe für Bilder an, wie Rauschunterdrückung, Remodulation und Kompression (Quantisation). Analogien zu XML-basierten Office-Dokumenten sind Angriffe, die den XML-Code direkt verändern und Transformationen oder Kanonisierungen anwenden, um Informationen zu entfernen.

2.2.13.2. Layout-Angriffe

Bezogen auf Wasserzeichen in Bildern sind geometrische Angriffe (Geometric Attacks) keine, bei denen das Wasserzeichen direkt entfernt wird. Statt dessen soll das Bild so verändert werden, dass die Erkennungsfunktion gestört wird. Dies kann durch Zuschneiden, Verkleinern oder sonstige Veränderungen am Bild vorgenommen werden [VPP+01]. Analog dazu lassen sich *Layout-Angriffe* für Office-Dokumente definieren. Hier wird dann das Layout des Dokumentes soweit umgestellt, dass die Ordnung oder Vollständigkeit der eingebetteten Information reduziert wird.

2.2.13.3. Kryptographische Angriffe

Wird stattdessen versucht, die Sicherheitsmethoden eines Wasserzeichen-Schemas zu überwinden um Wasserzeichen zu entfernen oder eigene Informationen einzubetten, handelt es sich um kryptographische Angriffe (Cryptographic Attacks). Dies sind teilweise Brute-Force-Angriffe und benötigen fast immer rechnerisch komplexe Berechnungen. Da sich diese Angriffe gegen das Schema richten, hängen sie, ebenso wie die noch zu beschreibenden Protokoll-Angriffe, nicht direkt vom Medium des

Cover-Objekts ab, müssen also ebenso für XML-basierte Dokumente in Erwägung gezogen werden.

2.2.13.4. Protokoll-Angriffe

Protokoll-Angriffe (Protocol Attacks) richten sich nicht gegen die zu Grunde liegende Technik, sondern trachten danach, Fehler oder Lücken in Protokollen auszunutzen, um beispielsweise fälschlicherweise die Autorenschaft an einem Objekt „zu beweisen“. Dazu gehört auch der Versuch, eingebettete Informationen in ein anderes Objekt zu kopieren.

2.3. XML-Dokumente

XML-basierte Office-Dokumente sind komplexe Dokumentenformate. Nicht nur umfasst die Beschreibung des jeweiligen Standards mehrere hundert (oder, im Fall von Office Open XML, sogar mehrere tausend) Seiten. Ire Entstehung ist auch einer Reihe von Vorgängern zu verdanken, die eine kontinuierliche Weiterentwicklung durchgemacht haben. Bevor die beiden Dokumentenformate näher dargestellt werden, folgt daher ein kurzer Überblick über die Zwischenschritte, die zur Entwicklung der XML-basierten Formate geführt haben. Dabei liegt der Fokus nicht auf der jeweiligen Tradition der hauseigenen, proprietären Binärformate, die durch konsequente Adaption von XML als Ausgangspunkt der konkreten Umsetzung der Formate gedient haben. Statt dessen wird nur die Entwicklung der Vorgänger von XML, XSLT und CSS betrachtet.

2.3.1. Vorläufer von XML

Der Urahn des heutigen XML ist die IBM Generalized Markup Language (GML). GML wurde für einen IBM Text-Formatierer namens „SCRIPT“ entwickelt. Hier war bereits die wesentliche Eigenschaft enthalten, dass generalisierte Auszeichnungen anstelle von direkten Formatierungen verwendet wurden.

In SGML (Standard Generalized Markup Language) wurden diese Ideen überarbeitet und erweitert. Als Anwendung von SGML wurde später die Hypertext Markup Language (HTML) entwickelt, um den Anforderungen des World Wide Web von Tim Berners Lee zu entsprechen. Allerdings wurden daran Erweiterungen von Browserherstellen vorgenommen, die nicht klar standardisiert waren. Das World Wide Web Consortium geht beispielsweise nicht davon aus, dass HTML mit einem SGML-Parser verarbeitet wird [Wor06].

In den 1990ern wurde mit der so genannten Extensible Markup Language (XML) [BPSM+06] eine Anwendung von SGML geschaffen, die eine Untermenge von SGML darstellt. Das Ziel von XML ist, eine erweiterbare Markup-Sprache zu schaffen, die an spezifische Bedürfnisse angepasst werden kann und dennoch allgemeinen und spezifizierten Regeln folgt.

2.3.2. Extended Markup Language

Die betrachteten Office-Dateien basieren im Kern auf der eXtended Markup Language (XML). Ihre grundlegendes Konzept ist, in einem Dokument den Inhalt, (logische) Struktur und Formatierungen strikt zu trennen, um die Verarbeitung zu vereinfachen. Mit XML wird der Inhalt strukturiert.

Eine Datenobjekt ist ein XML-Dokument, wenn es gemäß der XML-Spezifikation [BPSM+06] wohlgeformt ist. Darüber hinaus gilt es als valide, wenn es zusätzlichen, in der Spezifikation festgelegten Kriterien entspricht. Ein XML-Dokument beginnt mit einer Wurzel-Entität (das ist genau die Entität, die von keiner anderen eingeschlossen wird), die weitere Entitäten einschließt. Diese Entitäten haben zudem eine logische Struktur, so dass sie Elementen, Kommentaren, Anweisungen etc. entsprechen.

2.3.2.1. Auszeichnung

Der XML-Quelltext besteht aus vermischten Zeichen und Auszeichnungen (Markup) [BPSM+06]. Auszeichnungen sind unter anderem Start-Tags, End-Tags, Empty-Tags (manchmal auch als Single-Element-Tags bezeichnet) und weitere Auszeichnungen wie Auszeichnungsanweisungen (Processing Instructions). Jeder Text, der nicht zu den Auszeichnungen gehört, wird „Zeichen“ (Character Data, CDATA) genannt. Auszeichnungen, die keine Zeichen zwischen dem Start-Tag und End-Tag beinhalten, heißen leeres Element (Empty Element). Sie können alternativ durch einen einzelnen Tag (Empty-Element-Tag) ausgedrückt werden.

2.3.3. Cascading Style Sheet

Cascading Style Sheet (CSS) ist eine Stylesheet-Sprache zur Anwendung von Styles auf strukturierte Dokumente wie XML oder HTML [BeHL07]. Bei einer Trennung

von Inhalt, Struktur und Formatierungen kann unter anderem CSS für die Beschreibung von Formatierungen verwendet werden. Ein Dokument kann so durch die Zuweisung eines anderen Stylesheets umformatiert werden, ohne dass der Dokumentinhalt oder die XML-Struktur bearbeitet werden müssen.

2.3.4. Extensible Stylesheet Language

Die Extensible Stylesheet Language (XSL) ist eine Familie von Transformationssprachen, die Formatierungen oder Transformationen für strukturierte Dokumente in XML beschreiben. Insgesamt besteht XSL aus der Extensible Stylesheet Language Transformations (XSLT) , XSL Formatting Objects (XSL-FO) und der XML Path Language (XPath). XSLT [Cla99] ist eine Transformationssprache, um XML-Dokumente in ein menschenlesbares oder andere XML-Formate zu bringen. XSLT wurde beeinflusst von der Document Style Semantics and Specification Language (DSSSL). XPath dient der Ausgabe von ausgewählten Knoten der Baumdarstellung eines XML-Dokuments [CD99]. Schließlich wird XSL-FO [Ber06] verwendet, die XML-Strukturbeschreibung in eine Darstellungsform zu konvertieren. Oftmals – aber nicht zwingend – ist dies das Portable Document Format (PDF), bei dem Inhalte nicht mehr von Struktur und Formatierungen getrennt sind.

2.3.5. XML Parser

Zur Verarbeitung von XML-Dateien wird ein so genannter Parser benötigt. Dieser liest die Datei ein und bildet die Elemente in einer baumartigen Struktur ab. Diese Struktur wird *Document Object Model* genannt. Durch den Parser werden die Elemente und Attribute durch Datentypen oder Objekte der jeweiligen Programmiersprache verfügbar gemacht. Dabei lassen sich vom Wurzelknoten ausgehend die Kindelemente, und somit iterativ der gesamte Baum, durchsuchen.

2.3.6. ODF Dateiformat

Das Open Document Format for Office Applications (ODF) umfasst mehrere Dokumentarten, je eines für jede Anwendung:

- Textdatei für Text Dokumente,
- Präsentationsdatei für Präsentationen,
- Tabellendatei für Tabellenkalkulationen,

- Datenbanken,
- mathematische Formeln,
- Zeichnungen.

Jedes dieser Formate korrespondiert mit einem OpenOffice-Dateityp. Zur Vereinfachung der Beschreibung wird in Zukunft immer nur von ODT-Dateien (Open Document Text) geschrieben. Tatsächlich aber können ODF-Dateien auch andere Bezeichnungen tragen, je nachdem, mit welcher Office-Anwendung sie erzeugt wurden (auch andere Office-Pakete unterstützen den Standard, z. B. KOffice). Da die Textverarbeitung die wichtigste Anwendung eines Office-Pakets ist, werden die meisten Untersuchungen an diesem Dokumenttyp erfolgen. Dasselbe gilt für die Betrachtung von Microsoft Office Open XML.

2.3.6.1. Format des Dokumentencontainers

Der Container ist eine Archivdatei, die verschiedene komprimierte Dateien und Verzeichnisse enthält. In ODF [OAS07] wird ZIP zur Komprimierung benutzt. ZIP ist ein verlustfreies Komprimierungsverfahren. Die Spezifikation ist zwar nicht standardisiert, aber offen beschrieben in [PKW07]. Bei ZIP wird jede Datei einzeln komprimiert und anschließend zusammen in einer Archivdatei abgelegt. Der Header besteht lediglich aus den Buchstaben „PK“ (den so genannten „Magic Bytes“). Am Ende der Datei wird ein Footer angehängt, in dem Metadaten aufgeführt sind.

Der direkte Vorgänger des ODF-Containerformats ist allerdings das JAR-Format (Java Archive). JAR verwendet das ZIP-Komprimierungsverfahren, unterscheidet sich vom ZIP-Container aber durch ein zusätzliches Manifest, welches unkomprimiert in das Archiv aufgenommen wird. Da für jede Datei im Archiv die Kompressionsmethode festgelegt werden kann (also auch keine Komprimierung), sind JAR-Archive abwärtskompatibel. Auf diese Weise können sie ausgelesen werden, ohne die Manifest-Datei entkomprimieren zu müssen, was bei der Indizierung vieler Dateien Performanzvorteile bietet. Auch ODF-Archive enthalten ein unkomprimiertes Manifest.

2.3.6.2. Container-Struktur

Im Wurzelverzeichnis werden Unterverzeichnisse und einige XML-Dateien gespeichert. Das Hauptdokument ist eine Datei namens `content.xml`. Hier befindet sich der gesamte Dokumentinhalt (also beispielsweise der Text in einem Textdokument). Es ist eine XML-Datei, die XML-Auszeichnungen kennzeichnen die Formate und Strukturen des Dokuments. Die genauen XML-Elemente sind an dieser Stelle noch nicht wichtig (siehe Tabelle 2.1).

In der Datei `meta.xml` werden die Meta-Daten des Dokuments vorgehalten. Diese Daten sind unter anderem der Name der Anwendung, das Erstellungsdatum, der Name des Erstellers, die Dokumentsprache, die Revisionsnummer (Anzahl der Speicherungen) und vier Felder für benutzerdefinierte Metadaten.

Der Mime-Type des Dokuments steht in einer einzeiligen Nur-Text-Datei namens `mimetype`. Zum Beispiel ist in Listing 2.1 diese Datei für eine Open Document Text-Datei angegeben.

```
application/vnd.oasis.opendocument.text
```

Listing 2.1: *Mime-Type für Open Document Text*

Die Datei `settings.xml` enthält Konfigurationen, die auf das Dokument bezogen sind, wie Druckereinstellungen. Weitere Konfigurationen sind im Verzeichnis `configurations2` enthalten.

Um die Dokumente zu formatieren, werden Cascading Style Sheets (CSS) benutzt. Auszeichnungen werden mit einem Style verknüpft. Im Hauptdokument enthält der Text Auszeichnungen, die zugehörigen Styles werden aber teilweise getrennt in der Datei `style.xml` gespeichert.

Im Verzeichnis `META-INF` befindet sich die Datei `manifest.xml`. Sie enthält eine Beschreibung jeder Datei, die im Container enthalten ist, zusammen mit dem Mime-Type dieser Datei und dem Pfad im Container.

Informationen in der Datei `layout-cache` werden benutzt, um das Rendern des Dokuments zu beschleunigen. Die Datei ist eine proprietäre Erweiterung von Open Office, die von anderen Anwendungen ignoriert wird. Die Informationen innerhalb der Datei sind redundant.

Element	Typ	Inhalt
`content.xml`	Datei	der vollständige Dokumenttext
`meta.xml`	Datei	Meta-Informationen zu Dokument und Autor
`mimetype`	Datei	Mime-Typ des Dokuments
`settings`	Datei	dokumentbezogene Einstellungen
`styles`	Datei	Styles im Dokument
`configurations2`	Verzeichnis	Konfigurationen
`layout-cache`	Datei	redundante Informationen
`META-INF`	Verzeichnis	Verzeichnis für Meta-Informationen
`Thumbnails`	Verzeichnis	Enthält Vorschaubilder des Dokuments

Tabelle 2.1.: *Container-Struktur des Open Document Formats*

Die Dateien `meta.xml` und `mimetype` werden nicht komprimiert, damit sie direkt ausgelesen werden können, ohne dass erst der Container entpackt werden muss.

2.3.7. Office Open XML-Dateiformat

Office Open XML (OOXML) wurde zusammen mit Microsoft Office 2007 eingeführt, um die bisherigen proprietären Binärformate (z. B. Microsoft Word Dokument (DOC) abzulösen. Es wurde von der ECMA standardisiert als ECMA-376 [ECM06]. Anschließend hat Microsoft das Format ebenfalls bei der ISO zur Standardisierung im Schnellverfahren (Fast-Track) vorgelegt. Im Spätsommer 2008 gab die ISO bekannt, dass der Standardisierung von ECMA-376 nichts mehr im Weg stehe.

Der wichtigste Dateityp, auch für diese Arbeit, ist vermutlich das Microsoft Word Office Open XML Dokument (DOCX). Allerdings enthält der Standard auch Dateiformate für Tabellenkalkulation und Präsentationen.

Microsoft unterscheidet zwischen zwei Varianten von Dokumenten[4]. Die erste Variante darf keine Macro-Skripte enthalten und wird durch ein `-x` am Ende der Dateiendung gekennzeichnet (z. B. `*.docx`). Die zweite Variante enthält ausführbare Makros und endet immer auf `-m` (z. B. `*.docm`).

2.3.7.1. OOXML Dokument Container Format

Der Container von *Office Open XML* ist ebenfalls ein ZIP-Archiv. Der Inhalt des Containers, die Struktur der Dateien und Verzeichnisse, unterscheidet sich aber vom ODF-Format.

2.3.7.2. OOXML Container Struktur

Der Container von Office Open XML gliedert sich grob in drei Bereiche (siehe [Ehr06] und [ECM06, erstes Buch]):

- *Part Items*: Entsprechen einem Dokumententeil, so enthält z. B. die Datei `/word/document.xml` den eigentlichen Dokumentinhalt.
- *Content Type Items*: Enthalten zusätzlich hinzugefügte Objekte (z. B. Bilder).

4 Zusätzlich gibt es eine Binärvariante des Tabellendokuments für Excel, die als Excel Binary Workbook bezeichnet wird. Diese Variante enthält aus Performanzgründen überhaupt kein XML. Da sich diese Arbeit mit XML-basierten Dokumenten befasst, wird der letzte Dokumenttyp nicht weiter betrachtet.

- *Relationship Items*: Verbinden mehrere Part und Content Type Items zu dem vollständigen Dokument. `/_rels/.rels` verknüpft das Hauptdokument `document.xml` mit Metainformationen (z. B. `/docProps/core.xml`) oder einer digitalen Signatur.

Im Gegensatz zu den in Abschnitt 2.3.6.2 beschriebenen ODF-Containern befindet sich die Hauptdatei also nicht im Wurzelverzeichnis des Containers. Stattdessen gibt es anwendungsspezifische Unterverzeichnisse, die auch entsprechend benannt sind (also z. B. „word“ für ein Mircosoft Word Dokument). In diesem Unterverzeichnis ist dann die Hauptdatei namens `document.xml` enthalten.

Element	**Typ**	**Inhalt**
`word`	Verzeichnis	anwendungsspezifisches Unterverzeichnis
`document.xml`	Datei	der vollständige Dokumenttext
`.rels`	Datei	Relationship items

Tabelle 2.2.: *OOXML Container Struktur*

OOXML erlaubt, dass benutzerdefinierte Daten in Binärteilen (binary parts) gespeichert werden. Um dies zu erreichen muss laut [ECM06, Sec. 12.3.5] eine Relationship als `customProperty` in dem zugehörigen Package Relationship Item (der `.rels` Datei) typisiert werden.

3. Szenario

Um die in dieser Arbeit beschriebenen Verfahren in den Kontext eines Anwendungsfalls zu stellen, wird zunächst ein realistisches, jedoch fiktives Szenario vorgestellt. Zuerst wird der Markt geschildert, in dem das Szenario angesiedelt ist (Abschnitt 3.1). Dabei handelt es sich um ein Online-Marketplace, in dem Händler als Anbieter und Käufer auftreten. Exemplarisch wird dabei ein Marktsegment betrachtet, in dem Dichtungen für spezielle technische Anwendungen gehandelt werden. Um das Szenario realitätsnäher zu gestalten, werden Produkte erwähnt, die in einigen Grundzügen von den Informationsseiten einiger Dichtungshersteller stammen, beispielsweise [Alw08].

Anschließend wird das Online-System beschrieben (Abschnitt 3.2). Darauf folgt das eigentliche Szenario (Abschnitt 3.3) und eine kurze Darstellung möglicher Angriffe (Abschnitt 3.4). Schließlich wird kurz beschrieben, wie das System nach Beseitungung der genannnten Schwachstellen arbeiten soll (Abschnitt 3.5).

3.1. Der Markt für Spezialdichtungen

Die fiktiven Firmen *Alois Tender Dichtungstechnik*, *Ring-O-Matic*, *Brass GmbH* und *Evegasket* fertigen Dichtungen aus Elastomer oder Polytetrafluorethylen (PTFE) mit Spezialabmessungen und unterschiedlichen Shore-Härten (Härtegraden)[1]. Die Produkte werden über *OnGebot*, ein System zur Online-Angebotsabgabe, gehandelt.

Die Firmen der Branche bieten ihren Kunden eine umfangreiche Angebotspalette an Dichtungen und O-Ringen aus Rundschnur oder Formteilen an. Allerdings können die angebotenen Produkte von Firma zu Firma unterschiedlich sein. Nicht alle dieser Produkte werden selbst gefertigt. Die Produktions- und Lagerkapazitäten der meisten Firmen reichen dafür lange nicht aus, aber man möchte die Kunden langfristig an die Firma binden. So hat sich schon lange die Praxis bewährt, Produkte, deren Fertigung sich für eine Firma nicht rentiert, bei Wettbewerbern oder Zulieferern zu bestellen. Es ist außerdem kein Geheimnis, dass es Firmen in der Branche gibt, die Angebote abgeben und ihrerseits wieder nach einem günstigeren Anbieter suchen, um als Reseller an einem gewissen Preisaufschlag zu verdienen.

1 Nach DIN 53505 und DIN 7868.

Die Preisspanne der Dichtungen kann dabei von Massenware in Form kleiner Dichtungen, die einem geringen Preis pro Gramm verkauft werden, bis hin zu Spezialfertigungen über mehrere hundert Euro pro Stück reichen. Auch die Bestellmengen sind variabel. Oftmals werden naturgemäß kleine, günstige Dichtungen als Massenware vertrieben, während größere Spezialfertigungen teilweise als Einzelexemplar verkauft werden. Es ist auch möglich, dass Ware auf Abruf bestellt wird, die dann im Voraus hergestellt und eingelagert wird. Dann wird in regelmäßigen Abständen eine bestimmte Menge geliefert, die sich nach dem aktuellen Bedarf richtet.

Die Preise für die Fertigung oder den Einkauf von Dichtungen können aus verschiedenen Gründen stark variieren. Zum einen sind die verwendeten Kunststoffe letztlich in der Preisstruktur anfällig für die Schwankung von Rohölpreisen. Zum anderen können Anbieter, die größere Mengen des gleichen Produkts absetzen auf Grund von Skaleneffekten meistens auch günstiger Preise kalkulieren, als eine Firma, die für wenige Stückzahlen die Maschinen umrüsten müsste. Die Lagerung von nicht abgesetzten Dichtungen ist wegen der Kapitalbindung auch teuer - ein Anbieter kann Dichtungen, die lange eingelagert sind, günstig anbieten, weil dadurch Kapazitäten und Kapital frei werden. Außerdem sind eventuell Transportkosten unterschiedlich. All dies muss jedoch der Verkäufer bei seinen Preisen einkalkulieren, die Ware wird stets ohne Aufschläge zum dem ausgehandelten Preis verkauft. Die Angebote können also selbst für das gleiche Produkt vom selben Verkäufer schon nach wenigen Tagen unterschiedlich ausfallen.

Die Produkte werden im gesamten EU-Gebiet gehandelt, teilweise sogar weltweit. Vor allem aber werden solche Angebote sehr häufig eingeholt und abgegeben. Damit ist ein gewisser Verwaltungsaufwand verbunden. Das Online-System wurde gemeinsam von allen beteiligten Anbieter eingerichtet, um Ausschreibungen und Angebote einfacher abzuwickeln.

3.2. Das Online-Angebotssystem OnGebot

Bei der Benutzung des Systems *OnGebot* zur Online-Angebotsabgabe, schreibt die bestellende Firma (Käufer) zunächst die Produkte aus. Eine Ausschreibung kann aus mehreren Teilen in verschiedenen Dokumenten bestehen. Mindestens ist ein Anschreiben enthalten, in dem die Kontaktdaten der Firma im Kopf und im Textkörper eine Liste der Produkte und Mengen angegeben ist. Optional können auch zusätzliche Textinformationen zu den Produkten oder deren Einsatzzweck beigefügt werden. Zudem sind auch Skizzen oder technische Datenblätte möglich, um die Anforderungen in Bezug auf die Shore-Härte und Qualitätssicherung zu erläutern. Diese Informationen werden in Form von XML-Office-Dokumenten erzeugt. Anschließend werden sie mit einem digitalen Wasserzeichen versehen. Erst jetzt werden die Dokumente in

das System *OnGebot* hochgeladen und für Teilnehmer der Ausschreibung zugänglich gemacht (siehe Abbildung 3.1).

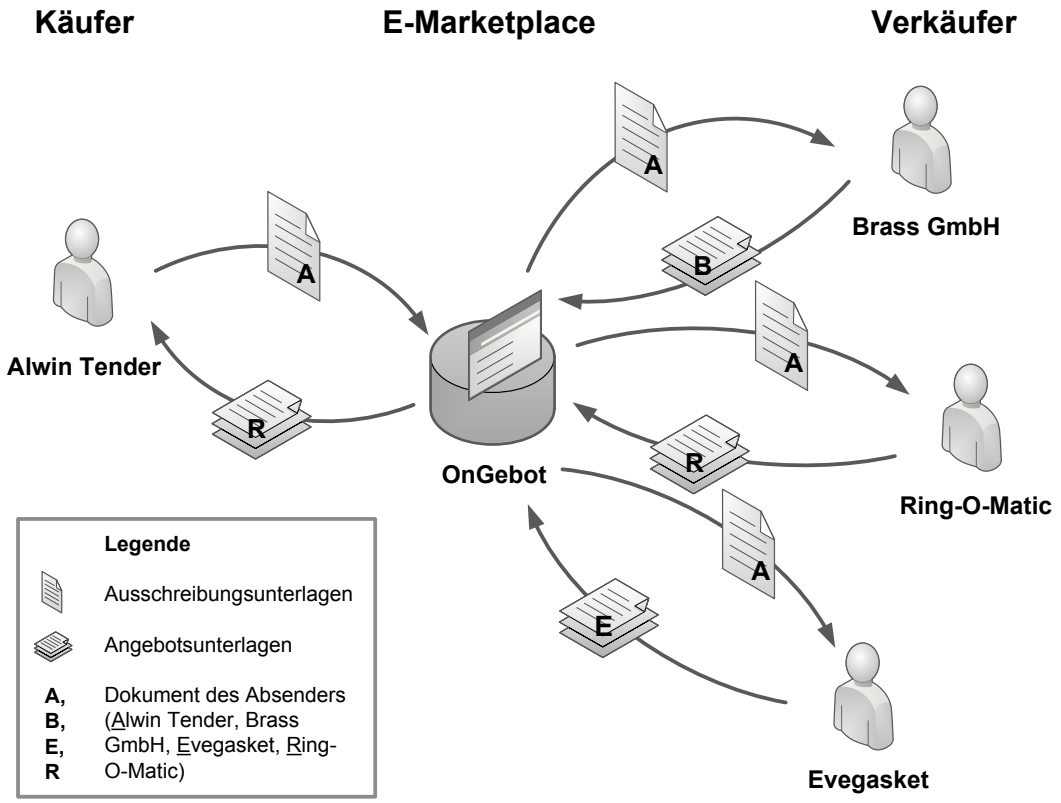


Abbildung 3.1.: *Online-Ausschreibungssysstem. Die Firma* Alois Tender *stellt eine Ausschreibung über das System in das Internet (1). Interressenten geben an das System ein Angebot in Form von Office-Dokumenten mit Wasserzeichen ab (2).*

Interessenten (Verkäufer) haben nun die Möglichkeit, auf die Ausschreibung hin ein Angebot zu erstellen. Dieses Angebot enthält in jedem Fall ein Anschreiben mit den Kontaktdaten der Firma, eine Veranschlagung der Kosten sowie eine Auflistung der zu liefernden Produkte. Weitere Dokumente sind wiederum optional. Dies können wieder Skizzen, Erläuterungen oder Datenblätter sein. Es wird vorausgesetzt, dass Standard-Office-Anwendungen benutzt werden. Die Firma *Alois Tender* lässt daher nur Dateien zu, die entweder dem Open Document Standard (also erzeugt mit OpenOffice oder Star Office) oder mit Office Open XML (also Microsoft Office) erstellt wurden. Alle Dokumente, die zum Angebot gehören, sollen vor der Abgabe mit einem Wasserzeichen versehen werden. Dieses Wasserzeichen identifiziert den Anbieter eindeutig. Der Betrieb *Ring-O-Matic* hat also ein anderes Wasserzeichen als die Firma *Brass GmbH*. Nach Einbringen des Wasserzeichens werden die Dokumente nicht mehr verändert. Anschließend werden die Dokumente mithilfe des

Online-Systems eingereicht. Ab dem Zeitpunkt der Abgabe kann davon ausgegangen werden, dass die Dokumente nicht mehr verändert werden können.

Die Serveranwendung wird in dieser Arbeit nicht weiter betrachtet. Es wird jedoch angenommen, dass es sich um eine Web-Anwendung handelt, in die Dokumente entweder per Upload über ein Web-Frontend oder per E-Mail über einen E-Mail-Gateway abgeben werden – das Verfahren ist in jedem Fall einfach und nicht zeitaufwändig. Die Serveranwendung gilt als neutrales System, d. h. es hat keine Kenntnisse vom Inhalt der Angebote und der Betreiber hat auch kein Interesse, diese zu erlangen. *OnGebot* wird regelmäßig von einer Vielzahl von Anwendern benutzt, so dass die Übersicht leicht verloren geht. Obwohl es eine Suchfunktion gibt, die bei der Auffindung von Angeboten und Ausschreibungen hilft.

3.3. Beispielausschreibung

Die Firma *Alois Tender* hat einen eiligen Kundenauftrag erhalten. Acht unterschiedliche Formteile aus Elastomer namens SuperDicht 1000 bis SuperDicht 1008 wurde bestellt, die dringend für die Reparatur einer Fertigungsmaschine benötigt werden. Nicht alle dieser Teile werden von *Alwin Tender* selbst hergestellt. Fünf dieser Teile werden daher im Fertigungssystem *OnGebot* ausgeschrieben. Die Unterlagen (ein Textdokument mit einem Anschreiben und je eine Tabelle mit dem Datenblatt für jedes Formteil) werden mit einem digitalen Wasserzeichen versehen. Dazu wird ein Schlüssel benutzt, der nur der Firma Alwin Tender bekannt ist. Die Dokumente (Dateien) werden in OnGebot eingestellt (Abbildung 3.2).

Daraufhin erstellen zwei Firmen je ein Angebot, *Brass GmbH* und *Evegasket*. *Brass GmbH* erstellt ein Angebot für die Teile und beabsichtigt, alle Teile selber zu fertigen. Weil für jedes Teil eine eigene Form hergestellt werden muss und die Rüstzeiten bei Einzelteilen einen hohen Anteil der Gesamtkosten ausmachen, fällt das Angebot preislich entsprechend hoch aus.

Evegasket hingegen betätigt sich als Reseller. Die Firma ist sehr erfahren in der Benutzung des Angebotssystems. Schnell findet sich mit *Ring-O-Matic* ein Anbieter, der im Auftrag von Evegasket ein Angebot erstellt und es an Evegasket sendet. Hierin sind neben den üblichen Teilen auch aufwändige Skizzen enthalten, sowie Zusagen der Abteilung für Qualitätssicherung, dass die Dichtungen mit der erforderlichen Shore-Härte hergestellt werden können. Evegasket fehlt die technische Kompetenz und Zeit, um solche Angebote selber zu erstellen. Statt dessen wird nur die Anschrift im Dokument ersetzt und der Preis um 2% erhöht. Anschließend wird das Angebot in OnGebot als Antwort auf die Ausschreibung von Alwin Tender hochgeladen.

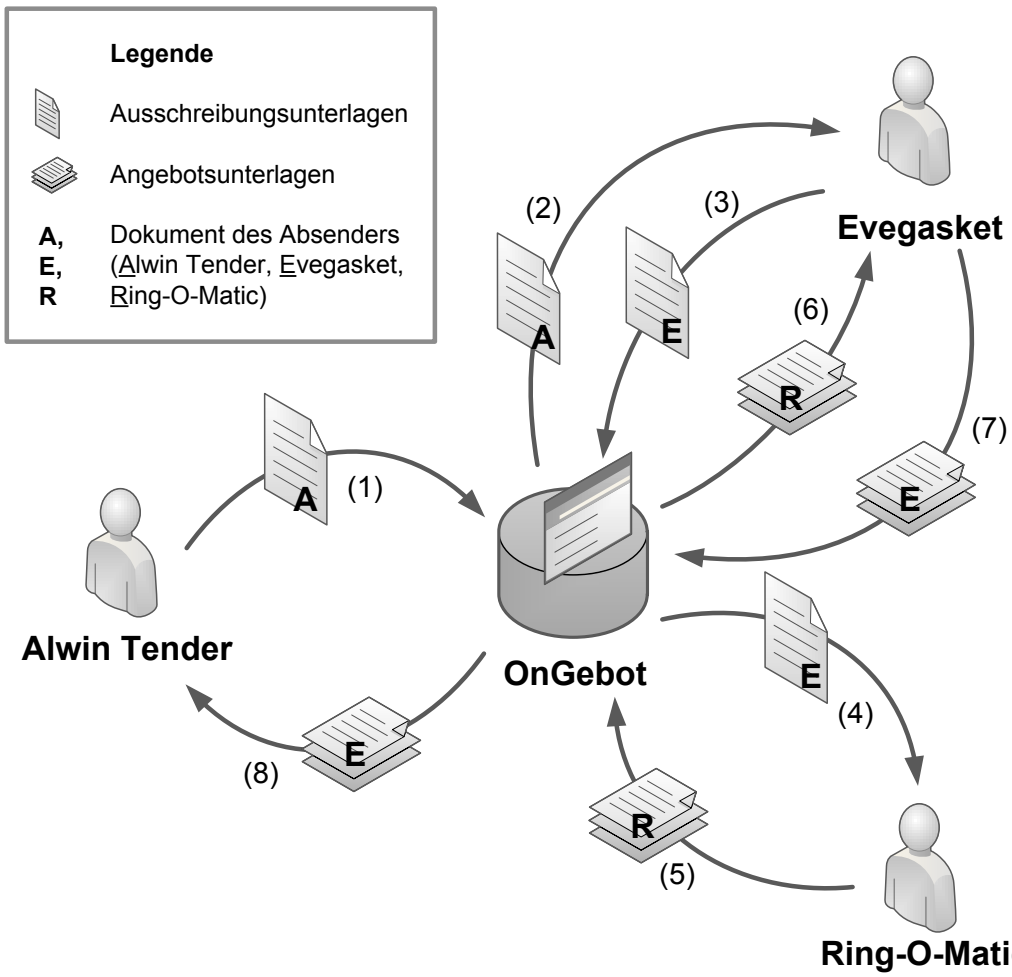


Abbildung 3.2.: *(1)* Alwin Tender *stellt Ausschreibungsunterlagen ein, die (2) von* Evegasket *abgerufen werden.* Evegasket *stellt diese Auschreibungsunterlagen als eigene ein (3), die Ring-O-Matic findet (4). (5)* Ring-O-Matic *gibt ein Angebot ab, das über das System an* Evegasket *(6) gelangt. (7)* Evegasket *weist das Angebot als eigenes aus und gibt es über das Ausschreibungssystem an* Alwin Tender *(8) weiter.*

Die Firma *Alwin Tender* wählt nun eines der vorliegenden Angebote nach wirtschaftlichen Kriterien aus und entscheidet sich für *Evegasket*, weil das Angebot insgesamt technisch fundiert und günstiger ist.

3.4. Mögliche Angriffe

Im Regelfall werden die Teilnehmer der Ausschreibung nicht die Angebote der anderen Teilnehmer erfahren. Ein Anbieter wie *Ring-O-Matic* weiß also nicht, ob ein Wettbewerber ein teureres oder günstigeres Angebot abgegeben hat. Ebenfalls unbekannt ist die Angebotsstruktur, also aus welchen Leistungen sich das Angebot genau zusammensetzt. Wie zuvor beschrieben, können solche Angebote auch Zeichnungen und Pläne und weitere Informationen enthalten.

Es muss aber damit gerechnet werden, dass einzelne Benutzer des Ausschreibungssystems versuchen können, Eigenschaften des Systems zu ihrem Vorteil zu missbrauchen.

3.4.1. Kopie von Dokumenten durch einen Reseller

Ein Reseller wie *Evegasket* kann die technische Kompetenz anderer Firmen ausnutzen. Dazu geht die Firma wie folgt vor: Ausschreibungen werden unter eigenem Namen erneut eingestellt, indem einfach der Briefkopf von *Alwin Tender* in den Dokumenten ersetzt wird. Geht daraufhin ein Angebot von *Ring-O-Matic* ein (zu deren Erstellung technisches Know-How vorhanden sein muss) wird dies in eigenem Namen der ausschreibenden Firma (*Alwin Tender*) angeboten.

Auf diese Weise kann *Evegasket* Angebote ohne wesentlichen Aufwand erstellen. Die Dokumente können leicht bearbeitet werden, da es sich um Office-Dateien handelt. Es gibt kein Risiko, Arbeit zu investierten, ohne dass sich das durch einen Auftrag auszahlt. Die Bestellung an *Ring-O-Matic* läuft ihrerseits erst an, wenn der Auftrag von *Alwin Tender* vorliegt.

Durch eine solche Kopie wird neben den Auswirkungen auf den fairen Wettbewerb, auch noch das Urheberrecht auf das Angebot verletzt, schließlich handelt es sich dabei um das geistige Eigentum der Firma *Ring-O-Matic*.

Wenn *Alwin Tender* im System die Möglichkeit hätte, jede Ausschreibung und jedes Angebot mit den eigenen abzugleichen, wäre ein derartiges Geschäftsmodell nicht möglich. Es wird aber an dieser Stelle davon ausgegangen, dass dies nicht effizient möglich ist. Es ist auch denkbar, dass *Evegasket* eine weitere Marktreichweite und einen größeren Kundenstamm hat als *Alwin Tender* oder *Ring-O-Matic*.

In einer verschärften Variante fängt *Evegasket* die Angebote und Ausschreibungen ab, bevor diese in das System eingegeben werden. Dazu ist ein aktiver Angriff notwendig: *Evegasket* muss entweder die Kommunikation eines oder mehrerer Käufer oder Verkäufer überwachen und abfangen – oder jegliche Kommunikation des Systems. Hier sind verschiedene Varianten denkbar, es wird daher beispielhaft folgende

betrachtet, bei der *Evegasket* die Verbindungen zwischen *Alwin Tender* und *OnGebot* abfängt:

1. *Alwin Tender* will eine Bestellung bei *OnGebot* einstellen. Diese Bestellung wird von *Evegasket* abgefangen.
2. *Evegasket* gibt die Bestellung als eigene aus und stellt sie bei *OnGebot* ein.
3. *Ring-O-Matic* antwortet, indem ein Angebot eingestellt wird.
4. Wenn *Alwin Tender* sich bei *OnGebot* einloggt, um zu prüfen, ob Angebote zu seiner Bestellung eingegangen sind, fängt *Evegasket* diese Suchanfrage ab.
5. Wenn *Evegasket* daraufhin das Angebot von *Ring-O-Matic* bei *OnGebot* findet, sendet sie Firma es als eigenes Angebot an *Alwin Tender*.

Bei dieser Variante kann *Alwin Tender* nie feststellen, dass ursprünglich bereits ein Angebot von von einem anderen Anbieter abgegeben wurde, was ansonsten bei gründlicher Suche möglich wäre. Allerdings spielt es für den weiteren Verlauf der Arbeit keine Rolle, ob ein aktiver oder passiver Angriff erfolgt. Eine umfangreiche Diskussion der verschiedenen Angriffsszenarien kann daher ausgelassen werden.

3.4.2. Abstreiten eines Angebots

Unter Umständen kann es sein, dass *Ring-O-Matic* ein angebotenes Produkt nicht mehr wirtschaftlich zum genannten Preis fertigen kann. Das kann daran liegen, dass Rohmaterialien teurer geworden sind. Oder daran, dass das Produkt auch nicht von *Ring-O-Matic* selber gefertigt wird, sondern von einem Zulieferer, der nicht wie angenommen liefern kann.

Das fällt nicht auf, wenn eine andere Firma den Zuschlag erhält. Wenn *Alwin Tender* jedoch den Auftrag erteilt, kann *Ring-O-Matic* behaupten, das Angebot gar nicht abgegeben zu haben. Immerhin ist es nicht schwer, ein Office-Dokument zu erstellen, das wie ein gültiges Angebot aussieht. *Alwin Tender* kann sich dann aber nicht auf die Echtheit und somit Wirksamkeit der Angebote verlassen.

3.4.3. Abstreiten der Ausschreibung

Alwin Tender kann beschließen, dass es wirtschaftlicher ist, die Produkte selber zu fertigen. Eine Ausschreibung ist eine sogenannte *Invitatio ad offerendum*, eine Aufforderung zur Abgabe eines Angebots [Bor01, Rn. 705]. Damit hat eine solche Aufforderung nicht den Charakter einer Willenserklärung. Eine Firma, die ein Angebot erstellt, hat also keinen Rechtsanspruch auf das Zustandekommen eines Vertrages.

Firma *Tender* kann aber auch behaupten, dass ein Außenstehender das Ausschreibungssystem benutzt hat, um eine Ausschreibung zu fälschen. Auch die Dokumente der Ausschreibung sind Office-Dokumente und können leicht erzeugt oder bearbeitet werden.

3.4.4. Fälschung eines Angebots

Die Firma *Evegasket* will den Betrieb *Ring-O-Matic* übernehmen. Der alte Geschäftsführer weigert sich jedoch noch den Betrieb abzugeben. Die Chefin von *Evegasket* versucht daher, *Ring-O-Matic* finanziellen Schaden zuzufügen, um ihn zum Verkauf zu bringen.

Sie erstellt daher ein Angebot im Namen von *Ring-O-Matic* und reicht es im Rahmen einer Ausschreibung bei *Alwin Tender* ein. Das Angebot ist günstig und entspricht den Vorstellungen von *Alwin Tender*, so dass der Zuschlag erteilt wird. *Ring-O-Matic* behauptet nun natürlich, wie im unter Abschnitt 3.4.2 geschilderten Fall, das Angebot sei gefälscht worden. *Alwin Tender* wird zu beweisen versuchen, dass das Angebot nicht gefälscht war, damit die Firma zumindest eine Entschädigung einfordern kann, wenn *Ring-O-Matic* nicht liefern kann.

3.4.5. Fälschung einer Ausschreibung

Außerdem ist es denkbar, dass es *Evegasket* gelingt, eine Lücke im Ausschreibungssystem zu nutzen und eine Ausschreibung zu fälschen. Wiederum ist die Ausschreibung an sich hier nicht bindend. Allerdings kann Evegasket auf diese Weise versuchen, an Dokumente anderer Firmen zu gelangen.

In diesem Fall muss *Alwin Tender*, damit er seine Kunden und Lieferanten nicht verprellt, glaubhaft nachweisen, dass er tatsächlich nicht der Ersteller der betroffenen Dokumente ist.

3.4.6. Überprüfung durch vertrauenswürdige Dritte

Um Streitfälle zu schlichten, wie sie in den Abschnitten 3.4.5 bis 3.4.1 beschrieben werden, können die betroffenen Parteien vor Gericht Klage einreichen. Der Richter wird nach Bedarf Sachverständige vorladen, die alle Dokumente und ihre Wasserzeichen untersuchen. Nach dieser Prüfung kann das Gericht im Idealfall entscheiden, welche der Parteien im Recht ist.

3.5. Erwünschtes Systemverhalten

Käufer und Verkäufer, die im E-Marketplace ihre Dokumente einstellen, sollen darauf vertrauen können, dass andere Teilnehmer nicht dazu ermuntert werden, schädliche Interessen zu verfolgen. Angriffe sollten verhindert oder abgeschwächt werden. In einem Handelssystem kann Missbrauch nicht vollständig verhindert werden. Dieser sollte aber nachverfolgbar und aufklärbar sein. Das bedeutet, dass es möglich sein muss, Täter im Nachhinein zu ermitteln. Die im vorigen Abschnitt geschilderten Angriffe müssen bei der Suche nach geeigneten Lösungsverfahren berücksichtigt werden.

3.6. Erkenntnisse aus dem Szenario

Der Online-Marktplatz OnGebot hat systembedingte einige Schwächen, die von Benutzern zu Ihrem Vorteil ausgenutzt werden können. Einige dieser Mängel lassen sich durch den Einsatz erprobter Protokolle und Maßnahmen zum Nachweis der Autorenschaft von Dokumenten lösen oder verringern. Durch die besondere Anforderung, dass handelsübliche XML-basierte Office-Dokumente ausgetauscht werden können, existiert derzeit nach bestem Wissen des Autors keine Lösung, um digitale Wasserzeichen in diesen Dokumenten einzubetten. Im den folgenden Kapiteln dieser Arbeit werden daher zunächst existierende Mechanismen untersucht, die ein solches Verfahren ermöglichen können. Anschließend wird ein Eigener Ansatz vorgestellt.

4. Analyse existierender Arbeiten

Im Szenario des vorausgegangenen Kapitel 3 wurde eine Anwendung beschrieben, bei der Käufer und Verkäufer auf einem E-Marketplace Geschäfte mit Unterstützung von XML-Office-Dokumenten tätigen. Es hat sich gezeigt, dass diese Anwendung Techniken und Methoden erfordert, um die immateriellen Werte (geistiges Eigentum) der Kunden und Händler zu schützen. Offensichtlich müssen die elektronischen Dokumente, die in dem System verwendet werden, geschützt werden, um unfaires oder schlichtweg kriminelles Verhalten seitens eines oder mehrerer Teilnehmer zu unterbinden.

Im folgenden Kapitel werden bestehende Ansätze aus der Literatur analysiert, die bei der Lösung dieser Fragen angewendet können. Ausgehend vom Grundgedanken des Urheberschutzes und dem Nachweis der Autorenschaft eines Dokuments werden zunächst generelle Ansätze diskutiert. Im Anschluss folgen eine Beschreibung sowie Analyse spezifischer Methoden und Techniken.

4.1. Vergleich verschiedener Ansätze zum Urheberschutz

Das beschriebene Szenario macht die Erfordernis deutlich, dass Autoren in der Lage sein sollten, ihre Autorenschaft gegenüber dritten zu belegen. Als Lösungen stehen Digital-Rights-Management-Systeme, digitale Signaturen und digitale Wasserzeichen zur Verfügung. Diese Verfahren werden an dieser Stelle kurz erörtert.

4.1.1. Digital Rights Management

Bei Systemen für Digital Rights Management werden technische Beschränkungen für die Nutzung und Verbreitung eines Dokuments eingeführt. Dies können Beschränkungen der Nutzungsdauer, des Ausführungssystems (Dateien arbeiten nur auf einem System korrekt), des Anwenders (nur ein Anwender kann die Datei ausführen) oder der Anwendung (nur bestimmte Anwendungen können die Datei öffnen) sein. Solche Einschränkungen werden in der Regel mit dem Ziel eingesetzt, dass ein Benutzer Dateien nicht an andere Personen weitergeben kann (weil diese sie nicht ausführen

können) oder nicht auf unautorisierten Geräten ausführen können (weil diese Geräte eine Transformation in ein ungeschütztes Format erlauben könnten).

Insbesondere bei digitaler Musik werden solche Lösungen derzeit eingesetzt. Kunden von Apples iTunes-Store können sich zwischen zwei Versionen von Musiktiteln entscheiden, einer DRM-geschützten Version und einer Version ohne DRM. DRM-freie Titel werden von Apple mit einem Fingerprint versehen, der den Käufer identifiziert [pte07]. Der Nachteil der DRM-Version ist, dass diese Titel nur mit der iTunes-Software abgespielt werden können. Außerdem ist der Song an das Gerät gebunden (oder zusätzlich an den Apple iPod-Musikplayer). Möchte ein Kunde die Musik auf einem anderen Gerät verwenden, so muss er die Titel auf CD brennen und auf dem Zielgerät wieder einlesen, was nicht ohne (geringen) Qualitätsverlust und (bei vielen Titeln hohen) Zeitaufwand sowie zusätzliche Kosten für die Rohlinge möglich ist.

Die bestehenden DRM-Systeme zeichnen sich bisher durch eine einseitige Verwaltung der „Rechte“ aus. Der Kunde wird in seinen Möglichkeiten nur eingeschränkt – der Rechteinhaber jedoch nicht. Gegner sprechen deshalb von „Digital Restriction Management„ [DRM]. Zudem gibt es Argumente, weshalb DRM-Systeme aus betriebswirtschaftlicher Sicht nicht für Rechteinhaber beliebiger Medien sinnvoll sind [Odl07]: Das Internet führe zu einem Anstieg der Menge frei verfügbarer Informationen, die in direkter Konkurrenz zu kostenpflichtigen Quellen stehen; Vertreiber hätten andere Möglichkeiten (wie Segmentierung und Versionierung), um den Ertrag erhöhen.

Außerdem muss ein DRM-System auf jedem angeschlossenen System durchgesetzt werden. Gibt es also eine Software oder Hardware, auf der mit Dateien aus dem System gearbeitet wird, muss dies DRM-unterstützende Soft- oder Hardware sein. Ansonsten kann der Schutz der Dateien umgangen werden und das System wäre nutzlos. Diese Durchsetzung kann zwar im Falle von OnGebot auf dem Server gewährleistet werden. Auf den Clients müsste aber ebenfalls eine entsprechende Software bereitgestellt werden. Weil aber nur mit Standard-Tools (Office-Anwendungen) gearbeitet werden soll, besteht diese Möglichkeit nicht.

4.1.2. Digitale Signatur

Für digitalen Signaturen werden XML-basierte Dokumente nach einem Verfahren signiert, das als XML-Digital Signature (XML-DSig) bezeichnet wird [ERS02]. Dabei wird zunächst eine Kanonisierung des XML-Codes durchgeführt[1]. Anschließend

1 Weil es bei XML möglich ist, unterschiedliche Syntax für dieselbe Auszeichnung zu benutzen, muss vor dem Signieren sichergestellt werden, dass exakt die gleiche Syntax signiert und später verifiziert wird. Dazu wird ein Verfahren verwendet, das Canonical XML (XML-C14N) [Boy00] heißt.

wird ein Digest erzeugt, eine kryptographische Zusammenfassung des Inhalts eben dieser kanonischen Syntax. Dieser Digest wird zusammen mit der Bezeichnung des Digest-Algorithmus in einem neuen XML-Element gespeichert. Nun werden Signaturinformationen erzeugt, unter anderem mit dem Digest, erzeugt und signiert[2]. Die Signatur und die Signaturinformationen werden als Signaturobjekt im Dokument gespeichert.

Das Verfahren hat für die im Szenario (Kapitel 3) geschilderten Anwendungen einige Nachteile. Zunächst einmal wird für ein asymmetrisches Verfahren eine Möglichkeit benötigt, die Zertifikate bzw. öffentlichen Schlüssel zu verteilen. Eine solche Public-Key-Infrastructure auszurollen, ist in der Praxis kein triviales Unterfangen. Außerdem ist ein häufig kritisiertes Problem [PW08, Gut04] bei XML-DSig, dass nicht das tatsächliche Dokument, sondern eine kanonische Form signiert wird. Der Grundsatz „What You See Is What You Sign“ (man signiert genau das, was man auch sieht), wird somit nicht erfüllt. Es ist für den Benutzer also nicht transparent, welche Inhalte von ihm genau unterschrieben werden. Da digitale Unterschriften aber rechtlich relevant sind, wäre umfassende Klarheit für den Benutzer jedoch wünschenswert. Vor allem aber können digitale Signaturen einfach entfernt werden: Das Signaturobjekt wird einfach aus dem Dokument entfernt. Anschließend kann, bei Bedarf, das Dokument erneut mit einer eigenen Signatur unterzeichnet werden.

4.1.3. Digitale Wasserzeichen

Für Wasserzeichen in XML-Dokumenten gibt es in der Literatur drei wesentliche Ansätze: Wasserzeichen, die innerhalb der Daten[3] von XML-Dokumenten, in den Formatierungen, die durch XML beschrieben werden und dem eigentlichen XML, den Auszeichnungen im XML-Code. Wenn Informationen in den Daten eingebettet werden, werden diese immer geringfügig geändert. Das Szenario in Kapitel 3 sieht jedoch vor, dass vor allem kaufmännische und technische Dokumente eingesetzt werden. In Dokumenten dieser Art ist zumeist eine hohe Genauigkeit der Angaben erforderlich. Die Daten sollten daher möglichst unverändert bleiben. Bei Markierungen, die durch Formatierungen in ein Dokument eingebracht werden, besteht die Gefahr, dass sich die Änderung der Formatierungen dem Benutzer gegenüber bemerkbar macht. Allerdings sind diese Markierungen zumindest bei XML-basierten Office-Dokumenten robuster als solche, die direkt im XML-Code eingebettet werden. Das hängt damit zusammen, dass letzterer beim Speichern jedes Dokuments neu erzeugt wird.

Grundsätzlich bietet die Verwendung von Wasserzeichen *Alois Tender* die Möglichkeit, die Nutzung seiner Dokumente nachzuvollziehen, ohne die Verbreitung einzu-

2 Für Details sei hier auf [ERS02] verwiesen.

3 Mit Daten sind in diesem Fall die Zeichen gemeint, also Character Data (Vgl. Abschnitt 2.3.2.1).

schränken. Das verwendete Verfahren sollte aber im besten Fall nicht nur *Alois Tender* nützen, sondern auch und den anderen Partizipienten des Online-Systems zumindest keinen Nachteil bringen. Welche Variante von Wasserzeichen am besten für das System geeignet ist, wird im Folgenden noch untersucht.

4.2. Desiderata

Die Anforderungen an das Verfahren werden anhand der im Abschnitt Kapitel 3 beschriebenen Angriffe innerhalb des Szenarios geschildert. Man kann dabei zunächst Anforderungen, die sich aus der Anwendung ergeben, von solchen unterscheiden, die sich aus der technischen Umsetzung ergeben.

4.2.1. Anforderungen aus der Anwendung

Aus der Anwendung ergeben sich folgende Anforderungen:

Eindeutigkeit Der Autor eines Dokuments (egal, ob es sich dabei um eine natürliche oder juristische Person handelt) soll eindeutig ermittelt werden können.

Da zu diesem Zweck Wasserzeichen eingefügt werden sollen, müssen diese Markierungen einem Autor oder einem Empfänger zugeordnet werden können. Je nachdem, ob es sich um ein Wasserzeichen oder Fingerprinting handelt, müssen die Markierungen Daten ergeben, die ausreichend unterschiedlich sind, um einem entsprechend großen Autoren- oder Empfängerkreis zugeordnet werden zu können.

Nicht-Abstreitbarkeit Ein Autor soll nicht abstreiten können, ein Dokument mit einem Wasserzeichen verfasst zu haben.

Fähigkeit, Manipulationen zu erkennen Wenn versucht wurde, Markierungen im Dokument anzugreifen, sollte es möglich sein, dies hinterher zu erkennen (Manipulationsschutz oder Tamperproofing). Im einfachsten Fall kann dies durch ein Vergleichsdokument erreicht werden, aber ein blindes Verfahren ist zu bevorzugen – also ein Verfahren, dass dieses Feature auch ohne ein Vergleichsdokument erbringt.

Unversehrtheit der Informationen Das Verfahren sollte nach Möglichkeit die Inhalte von Dokumenten unangetastet lassen. Veränderungen sollen sich also nur auf Daten auswirken, jedoch nicht auf die Informationen.

4.2.2. Anforderungen aus der Umsetzung

Zusätzlich ergeben sich Anforderungen aus der technischen Umsetzung:

Nicht-Wahrnehmbarkeit Markierungen in einem Dokument dürfen nicht zu Artefakten führen, die bei einer Rendering-Darstellung des Dokuments wahrnehmbar sind.

Verifizierbarkeit Die Markierungen im Dokument müssen als solche erkannt werden können, wenn der passende Schlüssel vorliegt und das richtige Verfahren angewendet wird. Wenn Markierungen nicht mit ausreichender Wahrscheinlichkeit erkannt werden, können eingebettete Informationen nicht mehr herausgelesen werden. Das Verfahren wäre dann insgesamt sinnlos.

Robustheit Eingefügte Markierungen müssen bis zu einem gewissen Grad Änderungen am Dokument überstehen können. Ansonsten wäre es einfach, Markierungen zu entfernen. Da laut Szenario das Dokument nach dem Einbetten des Wasserzeichens nicht mehr bearbeitet wird, brauchen solche Änderungen nicht als erlaubte Operationen berücksichtigt werden. Man kann also davon ausgehen, dass Änderungen am Dokument mutwillig vorgenommen werden, um Markierungen zu beeinflussen.

Kollisionsfreiheit Außerdem soll es rechnerisch aufwändig sein, einen Identifikator zu einem Empfänger zu berechnen. Somit ist sichergestellt, dass es nicht möglich ist, den Identifikator eines anderen Benutzers zu erzeugen und nach Veränderungen am Dokument dessen Wasserzeichen wieder einzubetten.

Ressourcenschonend Die Anwendung soll möglichst effizient arbeiten und damit wenig Rechenzeit in Anspruch nehmen. Für den Benutzer sollen keine langen Wartezeiten entstehen.

Plattformunabhängigkeit Es soll sichergestellt werden, dass die Lösung auf verschiedenen Betriebssystemen und mit unterschiedlichen Office-Anwendungen funktioniert.

Offenheit Die Lösung soll offen sein in der Hinsicht, dass das Verfahren bekannt sein und weitergegeben werden kann, ohne dass dadurch die Sicherheit des Verfahrens beeinträchtigt wird (Kerckhoffs Prinzip [Ker83]).

Ende-zu-Ende-Prinzip Saltzer [SRC84] fordert, dass Systeme soweit implementiert und durchgesetzt werden müssen, dass sie den gesamten Ablauf der Kommunikation zwischen den Partnern lückenlos abdecken. In Bezug auf das skizzierte Szenario bedeutet das, dass eine Funktion in der Client-Anwendung des Benutzers und der Server-Anwenung des Anbieters integriert werden muss.

Synchronistation Wasserzeichen müssen so eingebettet werden können, dass die einzelnen Markierungen wieder ausgelesen werden können, ohne dass die Reihenfolge der einzelnen Bits verloren geht.

4.2.3. Vergleichskriterien

Die zu untersuchenden Methoden werden anhand der obigen Kriterien auf ihre Eignung überprüft. Dafür muss zunächst ein Maß gefunden werden, damit die einzelnen Eigenschaften quantitativ vergleichbar sind.

Robustheit ist dann gegeben, wenn Markierungen einen Angriff überstehen. Offensichtlich kann man die Markierungen zählen, die nach einem Angriff noch vorhanden sind. Das allein ist aber keine sinnvolle Aussage, weil es unterschiedliche Arten von Angriffen gibt:

Subtraktive Angriffe werden mit dem Ziel eingesetzt, Markierungen zu entfernen. Hier kann man tatsächlich ein Verhältnis zwischen der Anzahl der Markierungen, die sich vor einem Angriff im Dokument befinden, zu der Anzahl der Markierungen im Dokument nach einem Angriff ausrechnen.

Allerdings werden bei einem subtraktiven Angriff möglicherweise auch Teile des Dokumentes entfernt, die keine Markierungen, sondern Bestandteile des Originaldokumentes sind. Dies verringert möglicherweise den Nutzen des Dokumentes, je nachdem wie stark der Inhalt oder evtl. sogar das Dokumentenformat beeinträchtigt

sind. Im schlimmsten Fall lässt sich das Dokument nicht mehr öffnen oder ist inhaltlich unlesbar geworden. Das Problem ist, wie hier die Verluste am Dokument zu bewerten sind. Eine rein quantitative Aussage wie „es fehlen 10 Bytes des Originaldokuments“ ist nicht aussagekräftig: Diese 10 Byte können Formatierungen sein, die kaum weiter auffallen, Zeichen, die wie ein Tippfehler aussehen und vielleicht vom Betrachter gar nicht wahrgenommen werden, oder aber fehlende XML-Tags, die zu einem ungültigen Dokument führen.

Ein Verfahren ist nicht ausreichend, wenn die Dokumente ihre Informationen durch die Einbettung verlieren. Es sei an dieser Stelle erneut auf die Unterscheidung von Informationen und Daten erinnert. Verlust oder Veränderung von Daten ist prinzipbedingt unvermeidlich. Da sich Daten aber nur auf die Speicherart von Informationen beziehen, wird der Integrität der Informationen ein höherer Stellenwert beigemessen als der Datenintegrität. Informationen gehen aus einem Textdokument beispielsweise dann verloren, wenn Teile des Textes (angefangen bei einzelnen Zeichen) verändert, gelöscht oder hinzugefügt[4] werden. Es ist zwar relativ einfach, Änderungen an den Daten zu überprüfen, indem man mit dem Programm `diff` den Originaltext mit dem veränderten Text vergleicht. Um den Unterschied quantifizieren zu können, kann man auch den Editierabstand der Texte berechnen. Die Aussage bezieht sich aber nur auf die Daten. Um wirklichen Informationsverlust festzustellen, muss Text semantisch verstanden werden. Daher kann man darüber keine Aussage treffen.

Natürlich kann ein Informationsverlust auch in anderen Dokumentteilen (wie etwa Bildern) stattfinden. Hier ist die Frage, ab wann Informationen verloren gehen, allerdings nicht mehr leicht zu beantworten. Artefakte aus einem verlustbehafteten Komprimierungsverfahren wie JPEG oder einem Watermarking-Algorithmus können zwar die Ästhetik des Bildes beeinträchtigen. Aber stellt das schon einen Verlust von Informationen dar? Der Betrachter kann meistens noch sehr gut erkennen, was auf dem Bild abgebildet ist. Oftmals werden die Artefakte gar nicht bewusst wahrgenommen. Daher argumentiert Smith, dass durch die Konvertierung eines Grafikformates keine Informationen verloren gehen [Smi08].

Additive Angriffe Bei additiven Angriffen werden einem Dokument Markierungen hinzugefügt, damit beim Erkennungsprozess zu viele Markierungen gefunden und die Informationen nicht ordentlich extrahiert werden können. Hier werden zum einen neue Markierungen eingefügt, zum anderen aber auch bereits vorhandene Markierungen überschrieben. Es werden daher die zusätzlich eingefügten Markierungen und die noch vorhandenen Markierungen gezählt.

4 Es ist leicht einzusehen, dass auch das Hinzufügen eines Zeichens keinen Informationsgewinn, sondern auch einen Informationsverlust darstellen kann, wenn man den folgenden Satz im Original (1.) und verändert (2.) betrachtet: 1. Hans ist ein Freund. 2. Hans ist *k*ein Freund. Die Aussage des Originalsatzes ist verloren gegangen.

Bei alldem stellt sich ferner die Frage, wie ein einheitlicher Angriff auf versteckte Informationen in einem Dokument beschaffen sein muss, damit einzelne Methoden nicht allein schon durch den Testfall bevorzugt werden. Außerdem werden viele Methoden gar nicht soweit überprüft werden können, weil es mit ihnen gar nicht oder nur mit Anpassungen möglich ist, in XML-basierte Dokumente Einbettungen vorzunehmen.

4.3. Existierende Methoden für Wasserzeichen

Im folgenden Abschnitt werden bereits existierende Methoden für Wasserzeichen oder Fingerprinting untersucht. Dabei wird geprüft, ob sie für XML-Office-Dokumente oder Teile davon anwendbar sind. Außerdem wird kontrolliert, ob sie anhand ihrer sonstigen verfahrensbedingten Eigenschaften geeignet oder, relativ zu anderen, besser sind.

Wichtig für das Verständnis der folgenden Analyse ist, dass die Änderungen, die an einem Office-Dokument vorgenommen werden, um ein Wasserzeichen einzubetten, auf verschiedenen Ebenen stattfinden. Die Ursache hierfür liegt zum einen in der Verarbeitung des Dokuments bei der Darstellung. Die Containerdatei wird entkomprimiert und die XML-Dateien werden verarbeitet. Für die Darstellung am Bildschirm und Drucker werden die Formatierungsinformationen gerendert, die aus dem XML gelesen werden. An diesen Stellen werden die Informationen aus dem Dokument in eine andere Darstellungsform überführt, und dies sind nie lückenlose Vorgänge. Hier bietet sich immer Raum für zusätzliche Informationen. Zum anderen hat ein Dokument verschiedene Bestandteile. So enthält ein Textdokument aller Wahrscheinlichkeit nach zunächst mal Text, aber auch Bilder oder andere eingebettete Dateien wie Videos, Audio-Dateien oder mathematische Formeln (die in einer eigenen Beschreibungssprache vorliegen). Diese ganz unterschiedlichen Elemente können mit verschiedenen Methoden ebenfalls Gegenstand der Einbettung von Wasserzeichen werden.

Verborgene Daten in gerendertem Text: Die Daten werden in der Text-Darstellung verborgen, die für ein Ausgabegerät gerendert wurde.

Linguistisches Verbergen von Daten: Hier werden Daten verborgen, in dem sie auf Sprachebene eingebettet werden.

Verbergen von Daten in XML: Spezifische Methoden, um Daten in XML-Dokuenten einzubetten.

Verbergen von Daten in nicht-textuellen Teilen: Office-Dokumente enthalten außer dem Text noch weitere Teile, in denen auch Informationen verborgen werden können.

Verbergen von Informationen in ZIP-Archiven: Die Container von XML-basierten Office-Dokumenten können ebenfalls benutzt werden, um Daten zu verstecken.

Sonstige Methoden: Methoden, die nicht in eine der genannten Kategorien passen.

Entsprechend dieser verschiedenen Ebenen werden die untersuchten Methoden in Unterabschnitte gegliedert. Da die Methoden für andere Anwendungsfälle entwickelt wurden, wird die Eignungsprüfung nicht zu den gleichen Ergebnissen führen als in der Domäne, für die sie entwickelt wurden. Neben der Diskussion wird zur Übersicht in jedem Abschnitt eine Tabelle nachgestellt, in der die wesentlichen Punkte kurz einheitlich zusammengefasst werden.

Robustheit: Die eingefügten Markierungen bleiben zu einem ausreichenden Grad erhalten.

Nicht-Invasivität: Die Informationen des Dokuments und nicht nur die Daten werden verändert.

XML-basiert: Die Methode nutzt die Bandbreite, die durch die Verwendung von XML entsteht.

Kapazität: Durch den Einsatz kann eine ausreichende Anzahl von Markierungen eingebracht werden.

Anwendbarkeit: Die Methode kann, ggf. mit Anpassungen oder Transformation des Kerns des Verfahrens, auf XML-basierte Dokumente angewendet werden.

Offensichtlich gibt es absolute Kriterien für ein Verfahren, wie die Anwendbarkeit. Die anderen Werte können nicht absolut bewertet werden. Da die Bewertungstabellen nur eine Übersicht darstellen sollen, wird trotzdem nur unterschieden zwischen „+“ für „ist gegeben“ und „–“ für „ist nicht gegeben“.

4.3.1. Verborgene Daten in gerendertem Text

Gerenderter Text ist entweder Nur-Text oder formatierter Text, der für ein Ausgabegerät gerendert wurdet (zum Beispiel Bildschirm oder Drucker). Vor diesem Rendering befindet sich der Quelltext dazu natürlich in einem ODF oder DOCX-Dokument und somit auch in einer Datei. Die Methoden, die in diesem Abschnitt diskutiert werden, gelten jedoch nur für eine gerenderte Ausgabe. Es ist sogar möglich, einige der

Methoden mit einer Schreibmaschine oder sogar handgeschriebenem Text zu verwenden[5]. Es wird daher zunächst geschildert, wie im Text selbst Informationen verborgen werden können. Die Methoden für XML werden weiter unten beschrieben.

Sogar einzelne Zeichen können zusätzliche Informationen enthalten. Im Unicode-Zeichensatz werden Zeichen für verschiedene Sprachen festgelegt, unter anderem auch für historische Sprachen. Einige Sprachen nutzen gleiche oder ähnliche Symbole, so dass eine 1-zu-n Abbildung von einem Glyph (Zeichensymbol) zu verschiedenen Unicode-Zeichen möglich ist. Zum Beispiel sieht das lateinische „o“ aus wie das griechische Omikron (sowohl in Altgriechisch (Koine) als auch im modernen Griechisch). Das Unicode-Zeichen für das lateinische „o“ kann dann eine 0 repräsentieren. Ersetzt man es durch ein Omikron, stellt das Zeichen eine „1“ dar. Es gibt auch weitere Zeichen, mit denen das funktioniert [MJF07].

Für formatiertem Text präsentierten Alattar und Alattar [AA04] eine Methode, mit der der Wortabstand innerhalb von als Block gesetzten Absätzen erhöht oder verringert wird. Zusätzlich wird der Zeilenabstand leicht verändert. In XML-basierten Office-Dokumenten könnten diese Formatierungen durch die Veränderung von Absatz-Styles erzielt werden.

Unicode unterstützt außerdem auch Hinweise zur Silbentrennung. So genannte *break points* und *no-break points* geben im Unicode [Con07] an, ob an einer Stelle eine Trennung erfolgen kann oder nicht. Eingefügte *break points* oder *no-break points* können als 0 oder 1 interpretiert werden. Dadurch könnte man an jeder Stelle mit einem solchen Trennzeichen ein Bit unterbringen. Diese Stellen werden immer im Unicode-Text markiert, die Anwendung entscheidet dann zur Laufzeit, ob an einem *break point* wirklich ein Umbruch erforderlich ist[6]. Bei XML-basierten Office-Anwendungen werden diese Unicode-Vorschläge zur Silbentrennung aber nicht beachtet, da die Anwendungen über eine eigene Silbentrennung verfügen.

Bei gedruckten Dokumenten gibt es zudem weitere Möglichkeiten, wie das von Mikkilileni et al. beschriebene Verfahren, durch Amplitudenmodulation des Lasers bei Laserdruckern [MCS+06] oder das Einfügen kaum sichtbarer Bildpunkte auf dem Ausdruck. Solche Verfahren eignen sich in erster Regel, um die verwendeten Drucker oder Kopierer identifizieren zu können und die Verbreitung der Dokumente auf diese Weise zu verfolgen.

5 Offensichtlich gilt das nicht für die weiter unten beschriebene Unicode-Methode.

6 Das ist deshalb so, weil man bei Unicode nicht nur von einem Zeichensatz für ein statisch formatiertes Textdokument ausgeht sondern auch für Texte, die an verschiedene Ausgaben angepasst werden. Zum Beispiel kann der Text an einem Fenster umgebrochen werden, dessen Größe geändert wird – dadurch ändert sich unmittelbar die Zeilenlänge, so dass die Silbentrennung erneut berechnet werden muss. Auch in einer normalen Textverarbeitung muss die Silbentrennung bei Änderungen am Dokument erneut berechnet werden.

Methode	Robust	Nicht-Invasiv	XML-basiert	Kapazität	Anwendbar	Bemerkung
Unicode	+	+	–	+	+	Nicht XML-basiert.
Wort / Zeilenabstand	+	+	–	+	+	Nicht XML-basiert.
Umbrüche	–	+	–	+	–	Bei Office intern anders gelöst.
Laser-Amplitude	–	+	–	–	–	Nicht anwendbar.

Tabelle 4.1.: *Vergleich Methoden für gedruckten Text*

4.3.1.1. Fazit

Werden Daten in gedrucktem Text verborgen, ist der wesentliche Vorteil, dass der Text beim Speichern und Drucken nicht von der Office-Anwendung verändert wird. Wird auf diese Weise ein Wasserzeichen eingebracht, bleibt dieses auch dann noch intakt, wenn eine Datei ausgedruckt und wieder eingescannt wird. Nur wenn ein Benutzer selber Änderungen am Text vornimmt, kann das Wasserzeichen zerstört oder beeinträchtigt werden. Das gleiche gilt natürlich für Daten, die in formatiertem Text verborgen werden.

Allerdings verändern solche Methoden den Inhalt oder das Layout des Dokumentes. Dabei ist unklar, inwieweit diese Änderungen rückgängig gemacht werden können. Zudem wirken diese Methoden auf die Darstellung des Dokuments, also auf die Daten. Wird ein Dokument nach dem Scannen auch noch mit einer OCR-Anwendung (Optical Character Recognition, Zeichenerkennung) bearbeitet, gehen die Einbettungen auch verloren, weil die OCR-Anwendung die Formatierung des Textes erneut vornimmt. Außerdem wird nicht die Bandbreite genutzt, die bei XML-basierten Dokumenten durch die Kodierung in XML entsteht. Das Verfahren zur Modulation der Laser-Amplitude ist grundsätzlich ungeeignet: Das in Kapitel 3 vorgestellte Szenario erlaubt neben einem Ausdruck auch die Darstellung auf dem Bildschirm. Eine Vorgehensweise, die nur auf einem Ausdruck funktioniert erfüllt daher verschiedene Grundvorraussetzung nicht. Am ehesten erscheint es vielversprechend, die Methode von Alattar und Alattar [AA04] anzupassen.

4.3.2. Linguistisches Verbergen von Informationen

Lingistisches Verbergen von Informationen (Linguistic Information Hiding) umfasst Natural Language Steganography (NLS) und Natural Language Watermarking (NLW). Der Unterschied liegt vor allem darin, dass bei NLS beliebige Informationen

eingebettet werden, bei NLW dienen die Einbettungen einem Sachzweck (zur genauen Unterscheidung von Steganographie und Watermarking siehe Abschnitt 2.2). Beide Forschungsrichtungen stützen sich auf Methoden und Erkenntnisse der natürlichen Sprachverarbeitung (Natural Language Processing, NLP). Eine Einführung in das Feld der natürlichen Sprachverarbeitung liefert [JM00, 1. Kapitel]. NLS und NLW sind überhaupt erst möglich, weil mehrere Sätze mit unterschiedlicher grammatikalischer Struktur den gleichen Informationsinhalt haben können. Sie sind daher ohne Informationsverlust austauschbar [TTD05]. Zum Beispiel lassen sich Aktivsätze in Passivsätze umformulieren, ohne dass der Inhalt verloren geht (Beispiel in Tabelle 4.2).

Originalsatz		**Transformierter Satz**
Vater fuhr Kevin zur Schule	$\Rightarrow$	Kevin wurde von Vater zur Schule gefahren.

Tabelle 4.2.: *Syntaktische Umformung eines Satzes durch Passivierung*

Die Verarbeitung der Sprache kann auf zwei Arten erfolgen. Entweder werden in einem ersten Schritt Sätze eingelesen (Natural Language Parsing) und dann nach vordefinierten Regeln die Ausgabe erzeugt (Natural Language Generation). Oder aber der Text wird in eine neutrale, kanonische Form gebracht. In dieser *Interlingua* werden Sätze auf eine Form zurückgeführt, in denen nicht zwei Sätze die gleiche Bedeutung haben [TTD05]. Die Datenbasis für die Übersetzung, große Wörterbücher, werden auch als *Corpus* bezeichnet[7]. Darüber hinaus gibt es elektronische Wörterbücher, die semantische Beziehungen zwischen synonymen Begriffen berücksichtigen, wie WordNet [Mil95].

Die Transformationen werden in Synonymsubstitutionen, syntaktische und semantische Transformationen unterschieden. Bei Synonymsubstitionen werden Wörter durch gleichbedeutende ersetzt (siehe Tabelle 4.3). Sie stellen die einfachsten Umformungen dar. Syntaktische Transformationen verändern die Syntax eines Satzes bei Bewahrung des Inhalts, wie die bereits genannte Passivierung (Tabelle 4.2). Semantische Transformationen sind deutlich komplexer, weil zum Beispiel Querverweise (Koreferenzen) von Nomen gebildet werden müssen. Auf diese Weise können Begriffe ersetzt werden, die sich auf dieselbe Entität beziehen (Beispiel in Tabelle 4.4). Die Schwierigkeit liegt darin, dass der NL-Prozessor erkennen muss, wie eine Entität alternativ referenziert werden kann [TTD05].

Die richtige Methode zur Einbettung von Informationen ist abhängig von der Art des jeweiligen Textes. So gibt es bei unterschiedlichen Textarten wie Briefen, Auf-

7 Die tatsächliche Größe der Corpora unterscheidet sich je nach Sprache. So wird im englischen ein deutlich größerer Anteil der existierenden Wörter regelmäßig verwendet (ca. 200.000) als im französischen (ca. 100.000) [Bry91]. Entscheidend ist daher, dass das Wörterbuch für die jeweilige Sprache repräsentativ ist.

Originalsatz		**Transformierter Satz**
Vater fuhr Kevin zur Schule	⇒	Papa fuhr Kevin zur Schule.

Tabelle 4.3.: *Synonymsubsitution eines Satzes*

Originalsatz		**Transformierter Satz**
Vater fuhr Kevin zur Schule	⇒	Vater fuhr ihn zur Schule.

Tabelle 4.4.: *Semantische Substitution durch Auflösung einer Koreferenz*

sätzen, Zeitschriftenartikeln, Bedienungsanleitungen oder Verträgen andere Anforderungen an Stil, Form und Ausdruck (und die Bewahrung davon). Zudem unterscheiden sich die Texte in der Sprache (Englisch, Deutsch...) und im Wortschatz. Boulevard-Reporter haben verständlicherweise einen anderen Wortschatz als Mediziner.

Ein System zur reinen Synonumsubstition präsentierten Topkara, Topkara und Atallah [TTA06b]. Auch wenn Synonyme über semantische Abhängigkeiten (wie in WordNet) aufgelöst werden, verbleibt die Gefahr, dass die beiden Wörter nicht exakt dieselbe Bedeutung tragen. In diesem Ansatz wird versucht, die Wörter eines Textes durch solche zu ersetzen, die ohnehin „mehrdeutiger“ sind. In [TTA06a] stellen die Autoren zudem ein Watermarking-System vor, das zusätzlich auf Satzebene mit Syntaxumformungen arbeitet. Das System liest Texte satzweise mit dem Parser des XTAG-Projektes [Pro07] ein, nimmt Umformungen zur Einbettung vor und schreibt den Text anschließend mit dem NLG RealPro [LR97]. Dabei muss jede mögliche syntaktische Umformung als Funktion implementiert werden. Für jeden Satz muss erst getestet werden, welche Umformungen überhaupt möglich sind. Dann erst kann eine Einbettung erfolgen. Das System ist allerdings auch fehleranfällig, weil die semantische Bedeutung der Sätze nicht erfasst wird. Als Beispiel für einen Fehler führen die Autoren die Umformung aus Tabelle 4.5 an.

Originalsatz		**Transformierter Satz**
Presidential elections must be held by october.	⇒	October had to hold presidential elections.

Tabelle 4.5.: *Fehlerhafte syntaktische Umformung (nach [TTA06a])*

Topkara [TTA07] schlägt zudem vor, künstlich eingefügte Rechtschreibfehler für das Verbergen von Daten in flüchtig verfassten (kursorischen) Texten zu verwenden. In einer E-Mail werden Fehler tendenziell eher toleriert als in einem förmlichen Brief. Dort kann eine solche Methode daher auch funktionieren. In einem geschäftlichen Schreiben wie einer Ausschreibung ist dies jedoch kaum geeignet.

Grothoff et al. stellen ein Verfahren vor, das eine andere Bandbreite zur Einbettung verwendet. Es wird der Raum genutzt, der durch maschinelle Übersetzung (Machine Translation, MT) entsteht [GGA+05]. Da diese Übersetzungen sehr fehlerbehaftet sind, fallen zusätzlich eingeführte Veränderungen des Textes nicht so stark auf wie in korrekten Texten. Offensichtlich gehen die Vorteile dieses Verfahrens auch sofort bei Texten verloren, die dieser Grundvoraussetzung entbehren.

Atallah und Raskin stellen ein Schema vor, in dem Text durch Umordnung von Wörtern, Phrasen und Buchstaben geringfügig verändert wird. Die Änderungen werden durch einen Parser als Syntaxbaum interpretiert und dann in eine Byte-Darstellung umgeformt [AR01]. Umformungen sind wiederum Passivierungen und ähnliche syntaktische Transformationen. Die Entwickler führen an, dass diese Wasserzeichen einige Bearbeitungen im Text überstehen. Allerdings räumen sie ein, dass ihre Lösung nicht bei Schriftstücken funktioniert, in denen es auf den Stil des Textes ankommt.

Atallah et. al erweiterten in [ARH+02] ihre früheren Arbeiten [AR01] durch die Möglichkeit, die Methoden nicht direkt auf einen Satz, sondern auf seine Bedeutungsrepräsentation (TMR) des Satzes anzuwenden. In ihrer Arbeit legen sie den Schwerpunkt auf die Technik der Einbettung, wie unter anderem die Auflösung von Koreferenzen von natürlichsprachlichem Text. Diese Erkenntnisse sind nicht für XML-basierte Dokumente zu verwenden. Allerdings wurde das Konzept, Markierungen in einer TMR des Satzes zu verwenden, von anderen Autoren aufgegriffen. Darunter sind Wu und Stinson [WS08], deren Verfahren „Authorship Proof for Textual Document“ (sic) weiter hinten ausführlich beschrieben wird, weil diese die Grundlage für die eigene Umsetzung bildet. An dieser Stelle soll nur kurz erwähnt werden, dass es sich dabei nicht um eine Einbettungstechnik, sondern um ein Schema handelt, das ursprünglich für linguistische Wasserzeichen entwickelt wurde. Wu und Stinson lösen das Synchronisationsproblem sehr gut, in dem sowohl bei der Einbettung als auch der Extraktion zunächst alle Sätze abhängig vom Schlüssel sortiert werden. Außerdem erhöht diese Vorsortierung auch die Robustheit: Werden von einem Angreifer zu einem späteren Zeitpunkt Sätze in eine andere Reihenfolge gebracht, um das Dokument zu verändern, bleibt das Wasserzeichen dennoch vollständig erhalten. Um die Robustheit weiter zu steigern, wird bei der Verifikation das extrahierte Wasserzeichen nicht einfach mit dem Original verglichen, sondern der Editierabstand berechnet. Darüber lassen sich quantitative Aussagen darüber treffen, inwieweit sich das ausgelesene Wasserzeichen vom Original unterscheidet.

Das NICETEXT-Paket von Mark T. Chapman [CD97] arbeitet auf andere Weise: Es wurde konzipiert, um bereits kryptographisch verschlüsselten Informationen ein natürlichsprachliches Aussehen zu verleihen. Damit handelt es sich um einen Natural Language Generator, der Cover-Text zu vorhandenen Informationen generiert. Offensichtlich kann es nicht benutzt werden, um Informationen in XML-Dokumenten einzubetten. Der Ansatz ist aber interessant, falls in Meta-Informationen, in denen

eigentlich natürlichsprachlicher Text oder „Tags“ stehen, Informationen eingebettet werden sollen, die möglichst keinen Verdacht erregen dürfen.

Satzoperationen sind sensibel gegenüber der syntaktischen und formalen Struktur, die sich aus den Satzbestandteilen ergibt. Änderungen an der Satzstruktur können kritische oder nicht-kritische Auswirkungen haben – letzteres lässt sich nur durch Erfassung der Semantik ermitteln. Unterstellt man, dass die Robustheit eines Wasserzeichens mit der Anzahl der Veränderungen zunimmt, geht damit auch eine zunehmende relative Wahrscheinlichkeit kritischer Auswirkungen auf den Informationsgehalt des Textes einher. Daher führt der Ansatz von Topkara, durch Induzierung von künstlichen Fehlern Informationen einzubetten nicht zu einem viel höheren Informationsverlust im Cover-Text als bei den anderen verwendeten Methoden.

Methode	Robust	Nicht-Invasiv	XML-basiert	Kapazität	Anwendbar	Bemerkung
Atallah & Raskin	+	–	–	+	–	Nicht XML-basiert.
Topkara et. al	+	–	–	+	+	Nicht XML-basiert.
Topkara	–	–	–	+	+	Stark invasiv.
Grothoff et al.	–	–	–	–	–	Nicht bewertet.
Wu & Stinson	+	–	–	+	–	Mit Anpassungen anwendbar, s. u.
NICETEXT	–	–	–	–	–	Nicht bewertet.

Tabelle 4.6.: *Vergleich Methoden für Linguistische Methoden*

4.3.2.1. Fazit

Auch wenn es nicht sofort auffällt, gibt es zwischen den Disziplinen des Information Hiding in natürlicher Sprache und in XML mehr Gemeinsamkeiten als zwischen Information Hiding in XML und anderen Medien wie Audio und Video. Die Herausforderungen und Anforderungen ähneln sich durchaus: Beide verfügen über eine vergleichsweise geringe Bandbreite. Und nicht jede Transformation kann auf einen beliebigen Satz angewendet werden (auch auf einen XML-Teilbaum können nur bestimmte Transformationen angewendet werden). Dazu kommt, dass die Verwendung des Dokuments bei beiden die Möglichkeiten der Einbettung einschränkt. Leider folgt daraus nicht, dass auch die Lösungen automatisch für beide Felder anwendet werden können – die Ergebnisse des Natural Language Watermarking sind überaus vielversprechend angesichts der Herausforderungen.

Für XML-Dokumente sind die verwendeten Methoden jedoch nur dann anwendbar, wenn man entweder den Text in einem Dokument verwendet oder versucht, Teile der Verfahren auf die XML-Struktur anzupassen. Im ersten Fall handelt man sich damit neben allen Nachteilen der geschilderten Verfahren (Informationsverlust und Zerstörung der semantischen Struktur des Textes) zusätzliche Probleme ein: Nur textuelle Teile des Dokuments können mit Informationen versehen werden. Der Vorteil, der durch die Verwendung von elektronischen Dokumenten entsteht bleibt ungenutzt – nämlich die Bandbreite an Informationen, die in der Struktur des Dateiformats eingebettet werden können. Gerade diese Einbettungen versprechen aber, den Informationsgehalt des Dokumentes am wenigsten zu beeinträchtigen. Im Szenario (Kapitel 3) werden jedoch auch technische Beschreibungen vorgesehen. Gerade bei derartigen Dokumenten kommt es auf ein hohes Maß an Korrektheit an. Diese Korrektheit der Informationen kann durch linguistisches Information Hiding nicht ausreichend bewahrt werden.

Im Fall der Anpassung der Verfahren an XML-Strukturen kann nicht einfach eine der beschriebenen Techniken wie die Synonymsubstitution oder die syntaktische Transformation von Sätzen verwendet werden. Synonyme zu einzelnen XML-Elementen gibt es eigentlich nicht. Allerdings können Kombinationen verschiedener Elemente und deren Attribute funktional gleichwertig sein. Anstelle eines äquivalentes Verfahren zu einer Synonymsubstitution in natürlicher Sprache für XML ist es daher sinnvoller, das Scheme von Wu und Stinson [WS08] zu übernehmen und für XML-basierte Dokumente anzupassen.

4.3.3. Verbergen von Daten in XML

Unter Wasserzeichen oder Steganographie in Bezug auf XML-Dokumente werden in der Literatur unterschiedliche Ansätze verstanden. Um Missverständnisse zu vermeiden, wird im folgenden unterschieden zwischen

XML-Wasserzeichen im weiteren Sinne umfassen alle Arten von Wasserzeichen, die in XML-Dokumente, XML-Datenbanken oder XML-basierte Strukturen allgemein (manchmal als abstrakte Semi-Strukturen bezeichnet [SAP03]) eingefügt werden, und

XML-Wasserzeichen im engeren Sinne bezeichnet die Arten von Wasserzeichen, die in die XML-Auszeichnungen, also nicht in die Daten oder den Text, eingebracht werden.

Bei den meisten in der Literatur diskutierten Methoden handelt es sich um Methoden für XML-Wasserzeichen im weiteren Sinne. Diese werden im folgenden kurz dargestellt.

4.3.3.1. XML-Wasserzeichen im weiteren Sinne

Obwohl die Autoren von Ng und Lau [NL05], ihre Arbeit „Effective approaches for watermarking XML data“ genannt haben, werden Informationen eigentlich nicht in den XML-Auszeichnungen, sondern im Text verborgen[8]. Die Arbeit beschreibt zwei Vorgehensweisen: Zum einen werden im Text synonyme Wörter anhand des Synonymwörterbuchs WordNet [Mil95] ersetzt. Zum anderen werden numerische Werte leicht modifiziert, indem das niederwertigste Bit geändert wird[9]. Die Autoren behaupten, dass dadurch die Bedeutung des Textes nicht verändert werde. Das mag für einige Fälle stimmen, aber nicht für alle. Es können leicht für beide Methoden Beispiele gefunden werden in denen die Bedeutung nicht übereinstimmt. Für die erste Methode kann man beispielsweise das Wort „car“ betrachten. WordNet führt hierfür nicht nur ein, sondern fünf semantische Verbindungen auf. Darin enthalten sind „automobile“, aber auch „railway car“, also Eisenbahnwaggon. Es ist offensichtlich, dass die semantische Bedeutung der beiden Synonyme nicht identisch ist. Würde in einem Satz wie „The train will arrive at platform three, meet me in front of the second car“, also „Der Zug wird an Bahnsteig drei ankommen, treffen Sie mich vor dem zweiten Waggon.“ könnte im Ergebnis das Wort „car“ durch „automobile“ ersetzt werden. Der Empfänger würde vermutlich vergebens vor der Taxi-Warteschlange oder auf dem Parkplatz warten. Zweitens kommt es bei numerischen Werten wie einem Datum, Uhrzeit[10] oder einer Adresse auch auf die präzise Wiedergabe der Zahl an. Wenn man aber z. B. XML-Attribute betrachtet, ist es wohl möglich, die niederwertigsten Bis von beispielsweise Farbwerten zu verändern.

Die Struktur eines XML-Dokuments entspricht einem Baum mit einer Wurzel, Zweigen und Blättern. Sie kann daher als Graph dargestellt und interpretiert werden. Wie oben beschrieben, können Graphen wiederum in eine numerische Darstellung überführt werden. Änderungen in der Struktur führen somit zu einem anderen numerischen Ausdruck. Auf diese Weise ist die Struktur selber auch informationsbehaftet. Sion, Atallah and Prabhakar [SAP03] haben einen Weg aufgezeigt, wie Knoten in einem Baum mit je einem Bit markiert werden können. Die tatsächliche Einbettung erfolgt dann in den markierten Knoten mit einer geeigneten Methode (also beispielsweise für Text oder Bilder). Einen ähnlichen Ansatz verfolgen Chen et al. [CCHD05],

8 Sie stellen außerdem eine Methode vor, um Wasserzeichen im ZIP-Container einzubetten; diese wird weiter unten beschrieben.

9 Es ist somit an ein linguistisches Verfahren angelehnt.

10 Es ist prinzipiell möglich, alternative und äquivalente Darstellungen für die Uhrzeit zu finden. Ein Wert wie „10:45“ kann auch als „Viertel vor Elf“ ausgerückt werden. Allerdings sind hier dann regionale Unterschiede zu berücksichtigen: „10:45“ wird in einigen teilen Deutschlands auch als „dreiviertel Elf“ bezeichnet, was aber nicht überall verstanden wird. Wie auch immer, dies ist nur möglich, wenn die nummerischen Werte semantisch erfasst werden können und nicht einfach „blind“ ersetzt werden.

allerdings wird dort nicht nur der jeweilig markierte Knoten, sondern auch die Baumstruktur an sich durch ein Wasserzeichen geschützt. Dadurch wird verhindert, dass die Knoten bestehen bleiben, aber der XML-Baum an sich neu geordnet wird.

4.3.3.2. XML-Wasserzeichen im engeren Sinne

Inoue et al. zeigen an einigen Beispielen, wie Informationen in XML verborgen werden können [IMM+01]. Die Vorschläge werden hier im Gegensatz zur Originalveröffentlichung mit Auszeichnungen illustriert, die in Open Document Formaten vorkommen. XML-Tags können entweder zusätzliche Leerzeichen vor der schließenden Klammer enthalten oder die Klammer kann direkt nach dem Tag geschlossen werden (Listing 4.1):

```
<text:p text:style-name="P1">Einige Dichtungsarten:</text:p>
<text:p text:style-name="P1" >Einige Dichtungsarten:</text:p
   >
```

Listing 4.1: *Leerzeichen in Tags*

Einzelelemente (im englischen Single Element Tags), zu denen keine schließende Variante notwendig ist, können aufgrund dieser Eigenschaft in zwei Varianten benutzt werden (Listing 4.2):

```
<text:p text:style-name="P1"/>
<text:p text:style-name="P1"></text:p text:style-name="P1">
```

Listing 4.2: *Varianten von Single-Element-Tags*

Wenn Tags verschachtelt sind, kann die Reihenfolge der Schachtelung geändert werden (Listing 4.3).

Und schließlich kann auch die Reihenfolge von Attributen innerhalb von Auszeichnungen geändert werden (Listing 4.4).

Alle diese Beispiele mit Auszeichnungselementen des Open Document Format erlauben es, ein Bit pro Element einzubetten, das jeweils eine der beiden Varianten 0 und die jeweils andere 1 darstellt. Die Reihenfolgevarianten können, je nach Anzahl der Tags oder Attributen, auch mehrere Bits enthalten, weil sich daraus mehrere Varianten ergeben, nämlich für n Elemente oder Attribute $n!$.

```
<text:span text:style-name="Hervorgehoben">
  <text:span text:style-name="Fett">
    Einige Dichtungsarten:
  </text:span>
</text:span>

<text:span text:style-name="Fett">
  <text:span text:style-name="Hervorgehoben">
    Einige Dichtungsarten:
  </text:span>
</text:span>
```

Listing 4.3: *Unterschiedliche Reihenfolge von Tags*

```
<text:h text:style-name="Heading_20_3" text:outline-level="3"
   >
<text:h text:outline-level="3" text:style-name="Heading_20_3"
   >
```

Listing 4.4: *Unterschiedliche Reihenfolge von Attributen*

4.3.3.3. Fazit

XML-Wasserzeichen im weiteren Sinne sind für XML-basierte Office-Dokumente nicht geeignet. Methoden, die nur die Daten und nicht die XML-Struktur markieren (wie [NL05]), sind vergleichbar mit linguistischem Information Hiding, sofern es sich bei den Daten um Text handelt. Und dafür gibt es, wie zuvor beschrieben, besser durchdachte Verfahren als die reine Synonymersetzung. Wird die XML-Struktur benutzt, um Teilbäume zu markieren, in denen Informationen eingebettet sind, entsteht dadurch für Office-Dokumente kaum zusätzlicher Nutzen. In XML-Datenbanken ist eine solche Markierung deshalb sinnvoll um Rechenzeit bei der Extraktion zu sparen. Solche Datenbanken sind, verglichen mit Textdokumenten, jedoch auch relativ groß.

Der einzige echte Vorschlag für Wasserzeichen in der XML-Struktur von Inoue [IMM+01] ist zwar prinzipiell geeignet, um Markierungen in die XML-Struktur einzubringen. Allerdings sind die Markierungen flüchtig bei der Anwendung in XML-basierten Office-Dokumenten. Bereits nach einem erneuten Speichern des Dokuments wären sie komplett entfernt. Um ein System wie im beschriebenen fiktiven Szenario unter diesen Bedingungen umzusetzen, wären zusätzliche Maßnahmen notwendig: Es müsste gewährleistet werden, dass ein Dokument nach der Einbettung

Methode	Robust	Nicht-Invasiv	XML-basiert	Kapazität	Anwendbar	Bemerkung
Ng und Lau	+	–	–	+	+	Kein XML-WM im engeren Sinne
Sion2003	+	–	–	+	+	Kein XML-WM im engeren Sinne
Inoue et al.	–	+	+	+	+	Sehr flüchtige Markierungen.

Tabelle 4.7.: *Vergleich Methoden für XML*

der Wasserzeichen nicht erneut bearbeitet werden kann. Das würde nicht nur die Idee eines offenen Dokumentformats konterkarieren, sondern wäre auch mit proprietären Formaten einfacher zu erreichen. Dementsprechend sind flüchtige Wasserzeichen für dieses Szenario nicht geeignet.

4.3.4. Verbergen von Daten in nicht-textuellen Teilen

Wie oben geschildert können Wasserzeichen in abstrakten Semi-Strukturen untergebracht werden. Dabei schlagen die Autoren meistens vor, im XML-Baum die Knoten zu markieren, die Wasserzeichen enthalten, und anschließend das eigentliche Wasserzeichen in die eingebetteten Dateien einzubringen. Das Vorgehen ist dabei nicht anders als bei herkömmlichen Wasserzeichen. Deshalb werden im folgenden Abschnitt nur kurz einige dieser Techniken geschildert.

Wasserzeichen in Bildern einzubetten ist eine verbreitete steganographische Technik [KP00]. Dabei können Anwendungen wie PGE [Sch94] benutzt werden, eine Software, um beliebige Informationen in Bildern zu verbergen. Das Ergebnis enthält zusätzliches Rauschen, was für die meisten Anwendungen in Bildern akzeptabel ist. Theoretisch kann ein solcher Algorithmus auch auf Bilder angewendet werden, die im Text eingebettet sind. Es ist sogar so, dass die Herausforderungen, denen steganographische Anwendungen normalerweise gewachsen sein müssen (Beschneiden, Ändern der Größe, Spiegeln des Bildes...) hier nicht zwingend existieren: Beide Office-Anwendungen behalten nach Bearbeitungen das Originalbild bei. Bei Microsoft kann ein Benutzer allerdings manuell eine Komprimierungsfunktion starten. Bilder können außerdem auch Meta-Daten enthalten (z. B. EXIF [JEI02], IPTC [IPT99], JPEG-Kommentare), in die beliebige Informationen geschrieben werden können [PW08].

Natürlich mangelt es dieser Methode an Erfolgsaussichten, weil nicht jedes Dokument Bilder enthält. Aber Office-Dokumente können beispielsweise ein Bild namens

`thumbnail.png` zur Vorschau enthalten. Auch wenn dieses Bild relativ klein ist, kann es möglicherweise einige Bits eines Wasserzeichens aufnehmen.

Eine andere sehr einfache Idee ist, Informationen in Meta-Daten zu speichern. Diese werden im Fall von OpenOffice in der Datei `meta.xml` gespeichert. Die Annahme ist, dass nicht alle Benutzer jedes Feld ausfüllen und einige Felder daher unberührt bleiben [CD04].

In beiden Office-Paketen wird ein Archivformat als Container für alle Dateien des Dokuments benutzt. Um ein Office Open XML White Paper [Ngo06] zu zitieren, wird das Archiv so zusammengestellt, dass „eine Möglichkeit bereitgestellt wird, um verschiedene Arten von Inhalten (z. B. XML, Bilder und Meta-Daten) in einem Container wie einem ZIP-Archiv abzulegen, um ein Dokument vollständig abzubilden“[11]. Es ist für diese Arbeit sehr bequem, dass im Open Document Format für Office-Anwendungen ebenfalls ZIP für das Containerformat benutzt wird. Das Format eines ZIP-Archives ist in [PKW07] detailiert beschrieben.

Der Dokumentencontainer und seine Unterverzeichnisse können versteckte Daten enthalten. Grundsätzlich besteht der Container aus den Hauptdateien, Meta-Daten, Konfigurationsdateien, Style Sheets, eingebetteten Objekten wie Bildern, Videos und anderen. Wenn ein Benutzer eine Datei in das Dokument importiert, zum Beispiel ein Bild, wird das Bild selbst im Dokument abgelegt. Eine Referenz auf die Datei wird im Dokument an der Stelle eingefügt, an der das Bild visuell dargestellt werden soll. Zusätzliche Dateien, die nicht referenziert werden, können schlichtweg im Container abgelegt werden und beliebige Informationen enthalten [PW08].

Methode	Robust	Nicht-Invasiv	XML-basiert	Kapazität	Anwendbar	Bemerkung
In Bildern einbetten	+	+	–	–	+	Kann durchaus sinnvoll sein.

Tabelle 4.8.: *Vergleich Methoden für nicht-textuelle Teile*

11 Das Zitat im englischen Original [Ngo06] lautet: „provide a way to store multiple types of content (e.g., XML, images and meta data) in a container, such as a ZIP archive, to fully represent a document“.

4.3.4.1. Fazit

Auch wenn die beschriebenen Methoden nicht direkt zur Markierung der XML-Struktur beitragen können, ist es sinnvoll, auch Bits in Elemente eines Dokuments einbetten zu können, die nicht aus XML bestehen. Ansonsten wären eingebettete Bilder zum Beispiel nicht durch ein Wasserzeichen geschützt. Für eine vollständige Lösung zur Markierung von XML-basierten Dokumenten sollten daher auf jeden Fall auch diese Dokumentteile durch Wasserzeichens oder Fingerprints einbezogen werden können.

4.3.5. Verbergen von Informationen in ZIP-Archiven

Wie bereits erwähnt, sind ODF-Dateien im wesentlichen ZIP-Archive mit einigen enthaltenen Ordnern und Dateien. Aus diesem Grund werden sie oft als Container bezeichnet. Jede Methode, die Informationen in ZIP-Archiven verstecken kann, wie [YSTM06] oder [AL03], lässt sich daher auch auf ODF-Dokumente anwenden.

Eine sehr einfache Methode aus [Tra07] ist, beliebige Inhalte an ein ZIP-Dokument anzubringen, indem die Daten einfach an das Archiv vorangestellt werden. Das funktioniert deshalb, weil für das ZIP-Dateiformat in [PKW07] definiert ist, dass es einen Header und einen Footer gibt, in dem Informationen zu Bestandteilen des Archivs aufgeführt sind. Einige Anwendungen beziehen nur die Informationen aus dem Footer und können damit korrekt auf alle Dateien im Archiv zugreifen. Vorstehende Daten werden einfach ignoriert. Allerdings melden diese Anwendungen einen Fehler, wenn Daten hinten an das Archiv gehängt werden. Der eigentliche Inhalt des Archivs ist unverändert, was zur Folge hat, dass es keine Auswirkungen auf die enthaltenen Dateien gibt. Soll diese Methode auf OpenOffice-Container angewendet werden, verhält es sich allerdings genau andersherum. Hier werden die Informationen aus dem Header gelesen. Dafür ist es möglich, Daten anzuhängen.

Der Nachteil dieser Methoden liegt in der fehlenden Robustheit. Jedes Mal, wenn ein Dokument gespeichert wird, erzeugt die Anwendung erneut alle XML-Dateien und den Container. Damit sind alle Daten, die einfach nur in den Container kopiert oder an den Container angehängt werden, wieder verloren. Allerdings ist es die schnellste Möglichkeit, um ein Dokument mit einem Präprozessor zu bearbeiten und zu markieren, so dass es eventuell für bestimmte Anwendungen dennoch nutzbringend sein kann.

Ein Beispiel: Dokumente an Netzwerkgrenzen nach Viren zu scannen. Wenn das Resultat negativ ist, kann mit der obigen, schnellen Methode ein Tag an das Dokument

angebracht werden. An diesem Tag kann später erkannt werden, dass die Virenprüfung erfolgreich durchgeführt wurde. Wenn das Tag durch Ändern des Dokuments nicht mehr vorhanden ist, können Virenscanner einen erneuten Test durchführen (dieser ist dann ohnehin notwendig).

Beispiel für ZIP-Archive

```
copy /b Embedded-Daten.txt + Cover-Archiv.zip Stego-Archiv.
   zip (on Windows)
cat Embedded-Daten.txt Cover-Archiv.zip >> Stego-Archiv.zip (
   on *nix / Mac)
```

Listing 4.5: *Beispiel für ZIP-Archive*

Beispiel für ODF-Dateien

```
copy /b Cover-Dokument.odf + Embedded-Daten.txt Stego-
   Dokument.odf (on Windows)
cat Cover-Dokument.odf Embedded-Daten.txt >> Stego-Dokument.
   odf (on *nix / Mac)
```

Listing 4.6: *Beispiel für ODF-Dateien*

Diese verborgenen Daten werden allerdings nicht in der Datei bestehen bleiben, wenn diese geändert wird. Sobald das Dokument neu gespeichert wird, gehen die angehängten Daten verloren.

Fazit: Ein großer Vorteil dieses Vorgehens ist die Einfachheit: Mit einem einzelnen Befehl können Dateien einem ZIP-Archiv hinzugefügt werden. Aber zusätzliche Dateien können ebenso leicht erkannt werden.

4.3.6. Sonstige Methoden

Li et al. stellen in ihrer Arbeit [LGJ04] eine Methode vor, um mittels flüchtiger Wasserzeichen (fragile watermarks) Datenbanken vor Manipulationen zu schützen (tamper protection). Dabei werden zunächst alle Tupel der kategorischen Daten abhängig

Methode	Robust	Nicht-Invasiv	XML-basiert	Kapazität	Anwendbar	Bemerkung
Atallah	+	+	–	+	+	Nicht XML-basiert.
Yoshioka	+	+	–	+	+	Nicht XML-basiert.
Einfaches Anhängen	–	+	–	+	+	Flüchtig, nicht XML-basiert.

Tabelle 4.9.: *Vergleich Methoden für ZIP-Archive*

von einem geheimen Schlüssel zu Gruppen zusammengefasst. Anschließend wird in jede der Gruppen unabhängig ein Wasserzeichen eingebettet. Die Einbettung erfolgt dadurch, dass die Reihenfolge der Daten des Tupels, wiederum abhängig von einem geheimen Schlüssel, verändert wird. Das Verfahren hat die Eigenschaft, dass die Daten selber nicht zerstört werden (nur die Reihenfolge). Werden die Daten später verändert, gehen diese Markierungen sehr leicht verloren. Aus diesem Grund ist die Methode dafür geeignet, Manipulationen an Daten aufzudecken. Außerdem ist durch die vorherige Gruppierung der Daten eine Lokalisierung eventueller Änderungen möglich: Wenn die Reihenfolge der Dateien bei der Verifikation nur in bestimmten Gruppen verändert ist, wurde der Rest der Daten mit hoher Wahrscheinlichkeit[12] nicht verändert.

Methode	Robust	Nicht-Invasiv	XML-basiert	Kapazität	Anwendbar	Bemerkung
Li	–	–	–	+	–	Muss modifiziert werden.

Tabelle 4.10.: *Vergleich sonstige Methoden*

4.3.6.1. Fazit

Flüchtige Wasserzeichen sind nicht für alle Anwendungen sinnvoll, aber die Arbeit von Li et al. zeigt eine interessante Verwendungsmöglichkeit auf. Die in der Veröffentlichung angewandte Methode ist indes nicht für XML-Dokumente geeignet (Tabelle 4.10). Weiter oben sind aber bereits Methoden beschrieben, die alternativ

12 Das ist keine generelle Annahme, sondern das Verfahren muss durch eine entsprechend hohe Anzahl von Markierungen sicherstellen, dass die Wahrscheinlichkeit wirklich hoch genug ist.

für flüchtige Wasserzeichen in XML-basierten Office-Dokumenten eingesetzt werden können. Dabei handelt es sich um die Methoden von Inoue et al. für XML-Dokumente [IMM+01] und von Yoshioka für ZIP-Archive [YSTM06].

4.4. Bewertung der Methoden

Aus den Ergebnissen der Untersuchung einzelner Methoden wurden bereits zuvor Rückschlüsse auf die Wirksamkeit der jeweiligen Kategorien gezogen, die zu Beginn von Abschnitt 4.3 eingeführt wurden. An dieser Stelle sollen diese Rückschlüsse noch einmal aufgegriffen und beurteilt werden.

Eine wesentliche Erkenntnis der Analyse ist, dass die Verarbeitung der Dokumente dazu führt, dass es sich bei den XML-basierten Dateiformaten um reine Speicherformate handelt. Sobald die Daten zur Verarbeitung in den Arbeitsspeicher geladen werden, werden die Dokumente in ein effizienteres Format überführt. Bei einem erneuten Speichervorgang wird der XML-Code erneut generiert. Damit haben sich Einbettungsmethoden, die ausschließlich die durch die XML-Repräsentation und das Containerformat entstehende Bandbreite zur Einbettung nutzen, als nicht robust – also als flüchtig – erwiesen. Das betrifft also die Methoden, die als XML-Wasserzeichen im engeren Sinne (Vgl. Abschnitt 4.3.3.2) aufgeführt sind und Methoden zum Verbergen von Daten in ZIP-Archiven (Abschnitt 4.3.5). Umgekehrt sind Methoden, die zur Einbettung die Daten heranziehen, wie etwa Verfahren für XML-Wasserzeichen im weiteren Sinne (Abschnitt 4.3.3.1), für nicht-textuelle Teile des Dokuments wie eingebettete Bilder oder Videos (siehe Abschnitt 4.3.4) sowie auch linguistische Methoden (Abschnitt 4.3.2).

Die letztgenannten Verfahren haben den oben beschriebenen Nachteil, dass sie Teile des Dokuments verändern, die sich auch auf die gerenderte Darstellung eines Dokuments auswirken. Im dem Fall, dass Markierungen in Bildern oder Videos eingebracht werden, ist das vermutlich zu vernachlässigen, da diese Medien für sich betrachtet über eine große Bandbreite für zusätzliche Informationen verfügen. Allerdings erstrecken sich Markierungen dann nur über einen Teil eines Dokuments. Wollte man den Text berücksichtigen, müsste man zusätzlich auf linguistische Methoden zurück greifen. Methoden für linguistisches Information Hiding können grundsätzlich auf jeden natürlichsprachlichen Text angewendet werden. Aus den Anforderungen, die sich aus einem E-Marketplace an die Korrektheit der Sprache ergeben ist, dies alledings eine suboptimale Lösung. Einbettungsmethoden, die in den Daten von XML-Dokumenten vorgenommen werden, sind immer mehr oder weniger invasiv, was die Beeinträchtigung der Informationen im Dokument betrifft. Aus dieser Erkenntnis ergibt sich ein Problem: Wie kann eine Lösung aussehen, die die Voraussetzung der Robustheit erfüllt, ohne dabei den Nachteil der Invasivität in Bezug auf

die im Dokument befindlichen Informationen in Kauf zu nehmen? Offensichtlich ist keiner der betrachteten Ansätze geeignet, um eine Antwort auf diese Frage zu liefern. Sie enthalten jedoch gute Bausteine, derer sich ein eigenes Verfahren zu Nutze machen kann.

Es gibt Ähnlichkeiten zwischen Information Hiding in XML-basierten Dokumenten und linguistischem Information Hiding. XML ist auch eine Sprache, nur eben eine formale. Das macht das Einbetten von Informationen mitnichten einfacher als in natürlicher Sprache. Die Anzahl von Alternativen zu einem Ausdruck sind in formalen Sprachen deutlich stärker eingeschränkt als in natürlichen. Dennoch gibt es bei beiden Disziplinen ähnliche Herausforderungen, etwa in Bezug auf die Bandbreite der verdeckten Informationen – zumindest verglichen mit Information Hiding in Multimedia-Dateien. Das sei mit folgendem kurzen Beispiel belegt: würde man in eine 32-Bit-Grafiken mit den Ausmaßen 1024×768 Bits das niederwertigste Bit eines Pixels bei der Einbettung ändern, beträge die maximale Bandbreite für verdeckte Informationen $\frac{1024 \times 768}{32} = 24.576$ Bits der gesamten Bilddaten. Das ist, wenn man davon ausgeht, dass ein Wasserzeichen mit einer Länge von wenigen hundert Bit eingebettet werden soll, eine Menge. In einem kurzen Brief hingegen, der etwa 20 Sätze enthält (ein Anschreiben oder eine Bestellung), könnte man mit einigen linguistischen Methoden auch nur maximal 20 Bits unterbringen, wenn dies auf Satzebene geschieht. Bei Synonymsubstitionen auf Wortebene lassen sich natürlich mehr Informationen unterbringen, aber die invasiven, zerstörerischen Effekte auf den Textinhalt nehmen damit auch deutlich zu. Mit XML verhält es sich ähnlich. Um das zu erkennen, muss man sich nur vor Augen führen, dass die XML-Struktur aus Auszeichnungen besteht, die die Zeichen (Character Data) des Dokuments umschließen. Je länger Absätze im Text sind, je weniger Formatierungen diese also enthalten, desto kleiner ist der XML-Anteil der Datei im Vergleich zum Text des Dokuments.

Aufgrund dieser Änlichkeiten wird im nächsten Kapitel beschrieben, wie Teile des linguistischen Verfahren von Wu und Stinson für einen eigenes Verfahren zum Information Hiding in XML-basierten Office-Dokumenten adaptiert werden können. Zudem wird dort die offene Frage der Einbettungstechnik behandelt, um eben den bislang fehlenden Kompromiss zwischen Robustheit und Invasivität zu erzielen.

5. Eigene Herangehensweise

Die in Kapitel 4 beschriebenen Verfahren können die Anforderungen an ein Wasserzeichen-Schema für XML-basierte Office-Dokumente nicht erfüllen, weil keine der Methoden Wasserzeichen in XML-Auszeichnungen robust einbetten kann. Methoden, die Informationen in den Daten des Dokuments einbetten, bergen hingegen eine zu große Gefahr, dass die Daten des Dokuments für das im Szenario beschriebene Online-System beschädigt werden. Im folgenden Kapitel wird daher ein neuer Vorschlag beschrieben, der den Erfordernissen gerecht werden soll. Zusätzlich zu diesem Schema muss zusätzlich eine geeignete Technik für die tatsächliche Einbettung gewählt werden. Außerdem muss ein Protokoll sicherstellen, dass mit diesem Wasserzeichen auch dann die Autorenschaft bewiesen werden kann, wenn sich nicht alle Teilnehmer fair verhalten. Aber das ist nicht Teil dieser Arbeit, für nähere Informationen sei daher auf Buyer-Seller-Protokolle wie [MW01] verwiesen.

Der Inhalt dieses Kapitels gliedert sich wie folgt: Zunächst folgt eine kurze Zusammenfassung des Authorship Proof Schemes nach Wu und Stinson. Anschließend wird die Transformation des Verfahrens in die neue Anwendungsdomäne in XML-basierten Office-Dokumenten geschildert. Dazu gehört, nicht anpassbare Teile des Verfahrens durch eigene zu ersetzen. Danach wird das vollständige Verfahren dargelegt. Es folgen eine Darstellung der technischen Umsetzung und schließlich einige Testfälle.

5.1. Authorship Proof Scheme

Das Schema basiert auf einem Authorship Proof Scheme (APS) für natürlichsprachliche Wasserzeichen von Wu und Stinson [WS08]. Da es sich um ein Verfahren für natürliche Sprache handelt, kann es nicht vollständig auf XML übertragen werden. Es muss daher an einigen Stellen angepasst werden, damit es für XML-Dokumente verwendet werden kann. In diesem Kapitel wird das Schema daher mit leichten Anpassungen wiedergegeben. Für das Original sei auf den Artikel [WS08] verwiesen. Zunächst folgt ein kurzer Überblick über die Bestandteile des Verfahrens, an denen Anpassungen vorgenommen wurden. Erst dann wird das angepasste Schema im Detail (Einbettung und Verifikation) geschildert.

5.1.1. Das Originalverfahren von Wu und Stinson

Wu und Stinson stellen in ihrer Arbeit „Authorship proof scheme for textual document“ [WS08] zwei Verfahren für den Beweis der Autorenschaft an einem Text vor, die APS2 und APS3 genannt werden. APS3 ist eine abgespeckte[1] Variante von APS2, weshalb im folgenden nur APS2 betrachtet wird. Insgesamt werden drei wesentliche Bestandteile verwendet: eine Bedeutungsrepräsentation für den Text, kryptographische Hash-Funktionen und der Editierabstand (nach Levenshtein).

Bei der Bedeutungsrepräsentation von Text (Text-Meaning-Representation oder TMR) handelt es sich um eine sprachunabhängige Beschreibung der Informationen in natürlichsprachlichem Text (siehe Abschnitt 2.1.10), die durch die Funktion $M\left(s_i\right)$ beschrieben wird. Damit kann bei veränderten Wörtern eines Satzes überprüft werden, ob sich die Bedeutung des Satzes nicht verändert hat.

Symbol	**Bedeutung**
K	Hauptschlüssel.
K_D	Dokumentschlüssel.
$W : (w_0, \ldots, w_{n-1})$	Wasserzeichen.
$M(s) : 0,1^* \rightarrow 0,1$	Funktion mit n-Bit langer Ausgabe („Bedeutung“).
$h_0 : 0,1^* \rightarrow K$	Hash-Funktion um den Schlüssel zu erzeugen.
$h_1 : K \rightarrow 0,1^n$	Hash-Funktion mit n-Bit langer Ausgabe.
$h_2 : K \times 0,1^* \rightarrow 0,1^{160}$	Hash-Funktion mit 160-Bit langer Ausgabe.
$h_3 : 0,1^* \rightarrow 0,1$	Hash-Funktion mit 1-Bit langer Ausgabe.

Tabelle 5.1.: *Symbole und Hash-Funktionen bei [WS08]*

Mittels gängiger Hash-Funktionen (Wu und Stinson schlagen vor, SHA-1 zu verwenden, lassen aber auch andere zu) werden Schlüssel und Wasserzeichen erzeugt. Die Symbole und Funktionen sind in Tabelle 5.1 aufgeführt. Es gibt Hauptschlüssel und Dokumentschlüssel. Ein Hauptschlüssel dient nur dazu, Dokumentschlüssel zu erzeugen. Der Dokumentschlüssel hängt nicht nur vom Hauptschlüssel, sondern auch vom Dokument selbst ab. Für jedes Dokument wird somit ein anderer Schlüssel erzeugt. Ein anderer Hauptschlüssel würde auch einen anderen Dokumentschlüssel erzeugen. Der Hauptschlüssel muss stets geheim bleiben. Ist dies gewährleistet, kann davon ausgegangen werden, dass nur der Besitzer des Hauptschlüssels wirklich den Dokumentschlüssel erzeugen kann. Der Dokumentschlüssel muss zum Zweck einer späteren Verifikation öffentlich gemacht werden, der Hauptschlüssel kann aber geheim bleiben.

Mit Dokumentschlüssel und Dokument wird das Wasserzeichen erzeugt. Die Länge des Wasserzeichens ist variabel und kann je nach Bedarf gewählt werden. Die Schlüs-

1 „Lightweight“ (Wu und Stinson).

sellängen hingegen sind fest vorgegeben. Außerdem wird ein Dokumentschlüssel nicht direkt eingebettet, um die Sicherheit der Schlüssel zu erhöhen. Das Wasserzeichen ist in diesem Verfahren also nicht frei wählbar.

Es wird nicht jeder Satz in einem Text mit einem Bit des Wasserzeichens markiert. Die zu markierenden Sätze und ihre Reihenfolge werden über eine Ranking-Funktion bestimmt. Diese erzeugt mit einem Hash-Algorithmus und der Bedeutungsrepräsentation des jeweiligen Satzes sowie des Dokumentschlüssels als Eingabe einen Rangwert für jeden Satz. Ein n-Bit langes Wasserzeichen wird dann in die n-Sätze mit den niedrigsten Rangwerten (in Reihenfolge mit aufsteigenden Rangwerten) eingebettet. Das Einbetten erfolgt, indem ein Satz so verändert wird, dass ein weiterer Hash-Wert des jeweiligen Satzes (der genau 1 Bit lang ist) genau dem Wasserzeichen-Bit an der korrespondierenden Stelle entspricht. Dabei muss die Bedeutung des Satzes erhalten bleiben[2].

Durch die bloße Einbettung eines Wasserzeichens ist natürlich noch nichts gewonnen. Erst durch eine spätere Verifikation kann ein Autor tatsächlich die Existenz seines Wasserzeichens im Dokument nachweisen. Im Zuge der Verifikation wird der Dokumentschlüssel offen gelegt. Mit ihm werden erneut die Rangwerte für jeden Satz im Dokument bestimmt. Anschließend werden die $n + d_{max}$ Sätze mit dem niedrigsten Rang so gehasht, dass ein 1-Bit-Wert entsteht. Diese Bits ergeben zusammen das Wasserzeichen. Die d_{max} zusätzlichen Sätze werden ausgewertet, um das Verfahren robuster zu machen, als wenn nur n Sätze gehasht würden. Nun kann nämlich mithilfe des Editierabstandes (siehe Abschnitt 2.1.12) zwischen dem eingebetteten und dem ausgelesenem Wasserzeichen (das d_{max} Bits länger ist) nach einer Entsprechung des Wasserzeichens gesucht werden, die weniger als d_{max} Bits von dem eingebetteten Wasserzeichen abweicht.

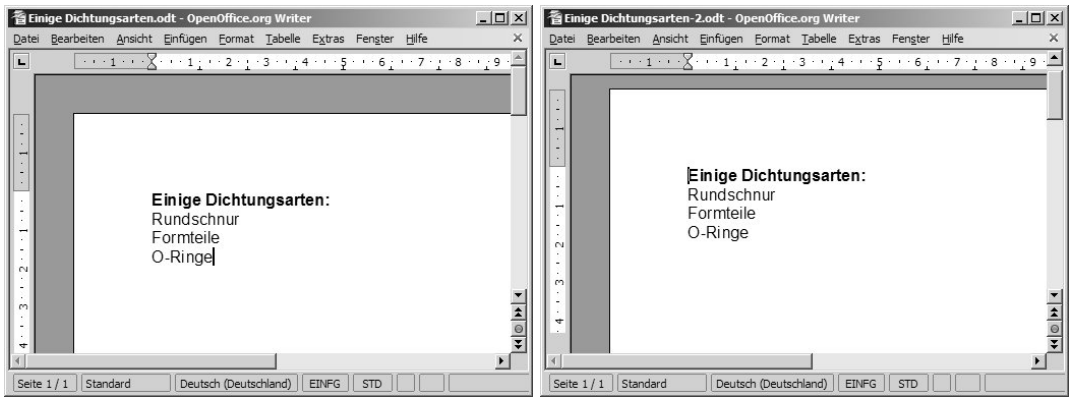
5.1.2. Eigene Anpassungen für XML-Dokumente

Das linguistische *Authorship Proof Scheme* von Wu und Stinson [WS08] wurde entwickelt, um mit einem englischsprachigem Wörterbuch als grundlegende Einbettungstechnik zu arbeiten. Im Vergleich zu anderen linguistischen Schemata wie von Atallah et al. [AR01] und Topkara [TTD05] stellen die Autoren ein Verfahren vor, dass Manipulationen auf der Wortebene vornimmt, die Bits eines Wasserzeichens aber auf Satzebene einbringt. Dieses Schema lässt sich nicht unmittelbar für XML-Dokumente anwenden. Es ist aber durchaus möglich, das Verfahren mit einigen Anpassungen zu adaptieren.

2 Das hat zwei Gründe: Erstens soll der Inhalt eines Textes offensichtlich nicht verändert werden. Zweitens wird bei der Verifikation der Rang erneut bestimmt, und nur wenn die Bedeutungsrepräsentation der Sätze sich nicht ändert, kann so die gleiche Rangfolge gebildet werden wie vor der Einbettung.

5.1.2.1. Darstellungsrepäsentation von XML anstatt Bedeutungsrepräsentation von Text

Eine der Kernideen des Verfahrens ist die Bedeutungsrepräsentation von Sätzen (TMR). XML-Dokumente verfügen zwar innerhalb des Dokuments auch über Sätze, allerdings sollen Informationen ja gerade nicht in den Daten verborgen werden. Deshalb muss ein Analogon zur Bedeutungsrepräsentation von Sätzen gewählt werden. Nahe liegend ist, eine gleiche Darstellung erreichen zu wollen. Zum einen aus rein praktischen Gründen: Wenn ein Dokument mit Wasserzeichen mit bloßem Auge von einem gleichen Dokument ohne Wasserzeichen unterscheidbar ist, ist das Verfahren mangelhaft. Zum anderen ist die Darstellung eines Dokuments das Ergebnis der Interpretation der Auszeichnungen, analog zur Bedeutung als Interpretation der Sprache. Es wird also nach Möglichkeiten gesucht, um eine äquivalente Darstellungsrepäsentation (siehe Abschnitt 2.1.11) zu erreichen.



(a) Cover-Dokument (b) Stego-Dokument

Abbildung 5.1.: *Vergleich zweier darstellungsgleicher Dokumente – der Inhalt wird gleich dargestellt.*

Zur Veranschaulichung der Idee enthält die Abbildung 5.1 zwei Screenshots eines OpenOffice Writer-Dokuments. Im XML-Code des oberen Dokuments befinden sich darin unter anderem die Zeilen aus Listing 5.1. Es handelt sich dabei um einen Ausschnitt aus der Datei `content.xml`. Der Text, der im Dokument eingegeben wurde, ist mit Formatierungen umgeben. In diesem Beispiel handelt es sich dabei immer um Absatzauszeichnungen (`<text:p></text:p>`). Der letzte Absatz ist leer, es gibt deshalb kein schließendes Tag – ein so genannter Single Element Tag[3]. Den Absätzen wird immer ein Style (Stil oder Formatvorlage) zugewiesen. Im Normalfall heißt dieser immer *Standard*. Werden den Absätzen abweichende Formatierungen zuge-

3 Alleinstehende Auszeichnung

wiesen, werden automatisch entsprechend andere Styles erzeugt. In der ersten Zeile handelt es sich um einen besonderen Absatzstil namens *P2*, in den weiteren *P1*.

```
<text:p text:style-name="P2">Einige Dichtungsarten:</text:p>
<text:p text:style-name="P1">Rundschnur</text:p>
<text:p text:style-name="P1">Formteile</text:p>
<text:p text:style-name="P1">O-Ringe</text:p>
<text:p text:style-name="Standard"/>
```

Listing 5.1: *Ein Ausschnitt aus Einige Dichtungsarten.odt, der alle Absätze des Dokuments zeigt.*

Bei OpenOffice werden automatisch Styles erzeugt, die den im Dokument benutzten Formatierungen entsprechen. Dabei sind Styles, die mit P beginnen, Absatzformate und die Styles, deren Bezeichnung mit T beginnt, Textformate. Die Styles werden einfach durchnummeriert. Zwei Beispiele für solche Styles sind im Listing 5.2 gezeigt, P1 für Absätze mit Fettdruck und T1 für Text mit Fettdruck. Alle Attribute von Textformaten können auch auf Absätze angewendet werden. Umgekehrt funktioniert das natürlich nicht – nur ein ganzer Absatz kann beispielsweise zentriert werden, und nicht ein einzelnes Wort eines Absatzes (siehe Abschnitt 2.1.7).

```
<style:style style:name="P1" style:family="paragraph"
style:parent-style-name="Standard">
  <style:text-properties style:font-name="Arial"/>
</style:style>
<style:style style:name="T1" style:family="text">
  <style:text-properties style:font-name="Arial" fo:font-
     weight="bold"
style:font-weight-asian="bold" style:font-weight-complex="
   bold"/>
</style:style>
```

Listing 5.2: *Automatisch angelegte Styles in Open Office*

Nun soll ein Dokument erstellt werden, das in seiner Darstellung dem ersten entspricht. Anstelle eines Absatzstils des ersten Absatzes wird ein Textstil verwendet, der sich in zwei Teilen so über den Absatz erstreckt, dass Leerzeichen ausgespart werden (siehe Listing 5.4). Das Problem dabei ist, dass die Anwendung beim Speichern eines Dokuments die Auszeichnungen optimiert – Redundanzen werden beseitigt, von mehreren gültigen XML-Varianten wird nur eine verwendet. Aus diesem Grund kann nicht der gesamte Absatz mit einem Textformat ausgezeichnet werden.

In dieser Variante – bei der das Leerzeichen ausgespart wird – erhält man nach einem erneuten Öffnen des Dokuments und anschließendem *Speichern* oder *Speichern unter...* allerdings wie gewünscht ein Dokument, in dem die geänderten Auszeichnungen unverändert gespeichert werden und im Dokument bestehen bleiben. Dies wird anhand von Listing 5.3 (der geänderte Absatz im Original) und Listing 5.4 (der veränderte Absatz) veranschaulicht.

```
<text:p text:style-name="P2">Einige Dichtungsarten:</text:p>
```

Listing 5.3: *Der unveränderte Ausschnitt aus Einige Dichtungsarten.odt*

```
<text:p text:style-name="P1">
<text:span text:style-name="T1">Einige</text:span> <text:span
text:style-name="T1">Dichtungsarten:</text:span></text:p>
```

Listing 5.4: *Der veränderte, aber darstellungsgleiche Ausschnitt aus Einige Dichtungsarten.odt*

Wie man anhand von Abbildung 5.1 sehen kann, unterscheiden sich die beiden Dokumente in ihrer Darstellung nicht. Dennoch führen verschiedene Auszeichnungen zu der gleichen Formatierung des Dokuments. Es konnte also mit diesem kurzen Beispiel gezeigt werden, dass es möglich ist, XML-Auszeichnungen zu finden, die zu einer identischen Darstellung führen. Dabei werden ganze Teilbäume des XML-Dokuments ersetzt. Nun kann man zwei Dinge tun. Erstens kann man eine Nummer für die gezeigte Darstellung vergeben. Man kann z. B. festlegen: Die Darstellung Nummer 5 entspricht einem Absatz, der fettgedruckt wird. Und es gibt (wie oben dargestellt) mindestens zwei Möglichkeiten, diese Darstellung mit XML-Auszeichnungen zu erreichen. Weil es praktischerweise zwei sind, genügt an dieser Stelle ein 1-Bit-Wert, um anzuzeigen, welche Variante verwendet wurde: Variante 0 ist dann der Code aus Listing 5.1 und Variante 1 ist der Code aus Listing 5.4.

Allgemeiner: Sind zwei XML-Zweige in ihrer Bedeutung gleich, erzeugen also die gleiche Darstellung, werden sie auf den gleichen Wert abgebildet. Dann kann eine Funktion $M\left(s_i\right)$ definiert werden, die zu einem Teilbaum s_i die Ausgabe oder Darstellung M zurück gibt. Die Darstellung wird über ein „Wörterbuch“ einem numerischen Wert zugewiesen. Das Wörterbuch und die Funktion $M\left(s_i\right)$ werden weiter unten im Abschnitt 5.2 beschrieben.

Es ist wichtig, dass in diesem Ansatz gesamte XML-Teilbäume manipuliert werden. Dies ist ein wesentlicher Unterschied im Vergleich zum Originalverfahren, wo

einzelne Wörter verändert werden. Die Auswirkungen erstrecken sich über Verfahrensschritte bis zu Sicherheitserwägungen (in Bezug auf die verwendete Look-Up-Tabelle).

5.1.2.2. Keine Hash-Funktionen zur Einbettung

Die Einbettung von Bits des Wasserzeichens erfolgt bei Wu und Stinson indirekt. So wird mit einer Hash-Funktion und einem geheimen Schlüssel als Parameter ein 1-Bit langer Hashwert jedes zu markierenden Satzes bestimmt. Entspricht dieser Wert nicht dem Zielwert, werden Wörter des Satzes so lange verändert, bis das Ergebnis der Hash-Funktion mit dem Zielwert übereinstimmt. Dabei wird berücksichtigt, dass sich die Bedeutung des Satzes nie verändert. Auf diese Weise hängt das eingebettete Bit eines Satzes von den verschiedenen Wörtern im Satz ab, deren Bedeutung im Wörterbuch enthalten ist. Selbst wenn die Bedeutungen der Wörter im Wörterbuch einem Angreifer bekannt sind, kann dieser die richtigen Änderungen nur erzeugen, wenn er auch die Parameter der Hash-Funktion kennt.

Wenn XML-Teilbäume ersetzt werden, kann die Einbettung nicht auf diesem Weg erfolgen. Hier wird ein Teilbaum durch einen anderen kompletten Teilbaum ersetzt. Beide Varianten enthalten direkt eine numerische 1-Bit Interpretation $I(s)$, durch die ein Bit des Wasserzeichens dargestellt wird. $I(s)$ ist eine Funktion, die zu einem Teilbaum den Bit-Wert aus dem Look-Up-Table zurück gibt (also bestimmt, welche von beiden möglichen Varianten eines Teilbaumes s entspricht). Es existiert also keine Hash-Funktion.

Ein Argument, weshalb dieses Verfahren unsicher ist: Sobald einem Angreifer das „Wörterbuch“ bekannt ist, kann dieser einfach immer eine andere Variante der XML-Teilbäume wählen und somit das Wasserzeichen zerstören[4]. Allerdings gibt es auch Argumente gegen ein Hashen wie von Wu und Stinson beschrieben: Es gibt keine deterministische Funktion, den richtigen Bit-Wert zu bestimmen. Es besteht sogar eine geringe Wahrscheinlichkeit, dass überhaupt nicht das richtige Bit erzeugt werden kann. Es ist nämlich denkbar, dass alle Kombinationen von Ersetzungen, wenn sie gehasht werden, zum gleichen, nicht erwünschten Bit führen. Zudem hängt der bei Wu und Stinson verwendete Hash, der zur Bestimmung des Bits führt, nicht von einem Schlüssel oder Geheimnis ab.

In einer weiterführenden Arbeit könnte untersucht werden, ob die Ersetzungsfunktion („Wörterbuch“) abhängig vom Dokumentschlüssel von einer Master-Version abgeleitet werden sollte. Dort müsste dann geprüft werden, wie hoch der Sicherheitsgewinn

4 Zur Erinnerung: Im Szenario ist die Anforderung enthalten, dass alle im System befindlichen Dokumente mit Markierungen versehen sind. Ein komplettes Entfernen der Markierungen ist für einen Angreifer keine Option.

ist, wenn an eine weiteren Funktion des Verfahrens diese abhängig vom gleichen Schlüssel vorgenommen wird.

5.1.2.3. Tie-Break-Regel bei Ermittlung eines Ranges

Die Autoren gehen in ihrer Veröffentlichung zur Vereinfachung des Verfahrens davon aus, dass es keine Sätze gibt, die einen gleichen Rang erhalten können. Da es durchaus vorkommen kann, dass XML-Dokumente ähnliche oder sogar gleiche Strukturen enthalten, wird hier eine Tie-Break-Regel eingeführt, der bei Teilbäumen mit gleichen Rangwerten sicherstellt, dass eine eindeutige Sortierfolge berechnet werden kann: Der Teilbaum, der im XML-Baum relativ zum anderen an vorderer Stelle steht, erhält den höheren Rang zugewiesen.

5.1.2.4. Editierabstand und Bestimmung des maximalen Abstands

Die Levenshtein-Distanz oder auch Editierabstand [Lev66] ist ein Maß für den Unterschied von Zeichenketten. Sie gibt an, wie viele Einzelschritte aus Einfügen, Löschen und Ersetzen notwendig sind, um eine Zeichenkette in eine andere zu überführen. Dieser Editierabstand wird bei der Verifikation zwischen dem ausgelesenen (aus dem Dokument extrahiertem) Wasserzeichen und dem Wasserzeichen, dass als Vorgabe von dem vermeintlichen Autor vorgelegt wird, berechnet. Die Annahme ist, dass ein Angriff auf die eingebetteten Markierungen dazu führen kann, dass nicht alle, aber einzelne Bits des Wasserzeichens ungültig werden. Vergleicht man beide Wasserzeichen nun, kann man einen Schwellenwert $d_{\max}$ für die Abweichung angeben, der nicht überschritten werden darf, damit das ausgelesene Wasserzeichen trotzdem als gültig anerkannt wird. Durch dieses Vorgehen wird die Robustheit des Verfahrens erhöht. Andere Verfahren lösen dieses Problem beispielsweise durch redundante Einbettung der Bits des Wasserzeichens.

Wie der Parameter $d_{\max}$ festgelegt wird, ist bei Wu und Stinson nicht näher beschrieben. Für XML-Office-Dokumente sind zwei Strategien denkbar: Erstens kann der Parameter festgelegt werden, indem man ihn als untere Schranke für die Sicherheit des Wasserzeichens versteht. Das bedeutet, dass man $d_{\max}$ so wählt, dass das die Länge des Wasserzeichens ohne $d_{\max}$ immer noch so viele Bits beträgt, dass ein Wasserzeichen in Bezug auf Kollisionen als sicher gilt. Je mehr Bits also erforderlich sind, um eine Kollision des Wasserzeichens zu verhindern, desto geringer muss $d_{\max}$ ausfallen. Dem entgegen steht dann das Ziel der Robustheit, die durch ein größeren Wert für $d_{\max}$ erreicht wird. Das Optimum ist in dem Fall also gleich der unteren Schranke. Zweitens kann man den Parameter bestimmen, in dem er so gewählt wird, dass die zulässigen Änderungen gerade soweit gehen, dass das Dokument noch als Kopie des Originals angesehen würde. Zur Erinnerung: Stimmt ein Bit an einer Position des

Wasserzeichens nicht mit dem eingebetteten überein, so kann dieses zwei Ursachen haben: das Dokument wurde geändert oder es ist nicht das gleiche Dokument (in letzterem Fall sind üblicherweise viele Bits nicht übereinstimmend). Da für jeden Teilbaum ein Bit eingebettet wird, wirkt sich das Löschen, Hinzufügen oder Verschieben eines Teilbaumes entsprechend auf das Wasserzeichen aus. Man kann argumentieren, dass nach einer gewissen Menge von Bearbeitungsschritten (und damit einer gewissen Anzahl unterschiedlicher Bits) das Dokument nicht mehr als „gleich“ angesehen werden soll, und das Wasserzeichen ungültig werden muss[5]. Es ist so aber deutlich schwieriger, einen konkreten Wert für d_{max} zu bemessen.

5.1.3. Einbettung

An dieser Stelle wird nun das Einbettungsverfahren beschrieben. Um das Verständnis zu erleichtern, folgt ein vereinfachtes Beispiel.

An verschiedenen Stellen des Einbettungsschemas werden kryptographische Hash-Funktionen benötigt, zunächst bei der Erzeugung von Schlüsseln. Außerdem werden Hash-Werte von XML-Zweigen berechnet. Wu und Stinson [WS08] konstruieren alle diese Funktionen mit Hilfe von SHA-1 [Nat02], beispielsweise durch Abtrennen des nicht benötigten Teils des Hash-Wertes. Insgesamt werden die in Tabelle 5.2 aufgeführten Hash-Funktionen benötigt.

Symbol	**Bedeutung**
K	Hauptschlüssel.
K_D	Dokumentschlüssel.
$W : (w_0, \ldots, w_{n-1})$	Wasserzeichen.
$M(s) : 0,1^* \rightarrow 0,1$	Funktion mit n-Bit langer Ausgabe („Bedeutung“).
$I(s) : 0,1^* \rightarrow 0,1$	Funktion mit 1-Bit langer Ausgabe („Information“).
$h_0 : 0,1^* \rightarrow K$	Hash-Funktion um den Schlüssel zu erzeugen.
$h_1 : K \rightarrow 0,1^n$	Hash-Funktion mit n-Bit langer Ausgabe.
$h_2 : K \times 0,1^* \rightarrow 0,1^{160}$	Hash-Funktion mit 160-Bit langer Ausgabe.

Tabelle 5.2.: *Symbole und Hash-Funktionen im angepassten Verfahren*

Für das Einbetten wird zunächst das Originaldokument (Cover-Dokument) benötigt. Das Originaldokument D ist die Hauptdatei aus dem Container des Office-Dokuments. Dazu muss also erst der Container entpackt werden. Im Fall von Op-

5 An dieser Stelle erkennt man sehr gut, dass Wasserzeichen grundsätzlich auch gut geeignet sind, um Veränderungen eines Dokuments nachzuweisen (Tamperproofing). Wenn ein Dokument bearbeitet wird, ist das Wasserzeichen ungültig. In diesem Fall würde man d_{max} sehr klein wählen, damit ein Wasserzeichen schon bei der Änderung von wenigen Bits ungültig wird. Oder man setzt in diesem Fall sogar $d_{\text{max}} = 0$, um gar keine Unterschiede im Wasserzeichen zuzulassen.

enOffice enthält die Datei `content.xml` das Hauptdokument und wird extrahiert. Bei Microsoft Office heißt die Hauptdatei `document.xml`.

Damit der Hauptschlüssel nicht herausgegeben werden muss, kann nun ein Dokumentschlüssel erzeugt werden. Dieser wird aus einem Hash über den Hauptschlüssel K und das Dokument berechnet: $K_D = h_0(K \| D)$. Grundsätzlich wäre auch denkbar, ein Wasserzeichen, das direkt mit dem Hauptschlüssel zu erzeugen, einzubetten oder einen Dokumentschlüssel auf andere Weise erzeugt wird. Aber dann müsste der Hauptschlüssel zur Verifikation offen gelegt werden, was diesen für die Verwendung bei weiteren Dokumenten unbrauchbar machen würde.

Anschließend werden folgende Schritte ausgeführt:

Schritt 1 ein $n-$Bit Pseudozufallswert $W = h_1(K_D)$ wird erzeugt, der vom Dokumentschlüssel abhängt. $W = w_0, \ldots, w_{n-1}$ ist das Wasserzeichen, das eingebettet werden soll. Das Wasserzeichen hat im Gegensatz zu den Schlüsseln keine fest vorgegebene Länge.

Schritt 2 Es werden Teilbäume $S = s_0, \ldots, s_m, m \geq n$ des XML-Baumes bestimmt, die Auszeichnungen enthalten, zu denen äquivalente Auszeichnungen existieren.

Schritt 3 Für jeden zuvor festgelegten Teilbaum s_i des XML-Dokumentes wird ein Rang berechnet: $o_i = h_2(K_D, M(s_i))$. Dieser Rang ist, weil er vom Schlüssel abhängt, geheim.

Es ist möglich, dass es im Dokument zwei darstellungsgleiche Teilbäume gibt. Diese erhalten so den gleichen Rang. Um das auszuschließen, wird bei einer solchen Situation der Rang des kleineren[6] Teilbaumes um eins verringert. Sollte es immer noch Konflikte um die Rangposition geben, wird eine Tie-Break-Regel angewendet: Der Teilbaum, der im XML-Baum relativ zum anderen an vorderer Stelle steht, erhält den höheren Rang. Da ein XML-Baum geordnet ist, kann eine so bestimmte Rangfolge nicht zu einem Konflikt führen[7].

Schritt 4 Jetzt werden die Teilbäume $\left\{s_{j_0}, \ldots, s_{j_i}, \ldots, s_{j_{(n-1)}}\right\}$ mit dem niedrigsten Rang umgeschrieben zu Teilbäumen $\left\{s'_{j_0}, \ldots, s'_{j_i}, \ldots, s'_{j_{(n-1)}}\right\}$, so dass $I\left(s'_{j_i}\right) = w_{j_i}$ und $M(s_{j_i}) = M\left(s'_{j_i}\right)$ ist. Jeder Teilbaum wird also, falls es notwendig ist, durch einen ersetzt, der aus einem anderen Ausdruck besteht und somit einem Bit w_j des Wasserzeichens entspricht, aber weiterhin die gleiche Darstellung hat. Wenn die Variante des Teilbaums schon dem Bit w_j entspricht, wird keine Änderung an diesem Teilbaum vorgenommen.

6 Geringere Anzahl an Zeichen (sowohl XML-Auszeichnungen als eingeschlossene Daten).

7 Wenn ein Rang erhöht oder verringert wird, müssen die anderen Ränge sukzessive geprüft werden, ob dadurch wiederum ein Konflikt entsteht, aber am Ende dieses Vorgangs ist jeder Rang eindeutig.

Nach den Ersetzungen enthält das Stego-Dokument $\hat{D}$ das Wasserzeichen. Die veränderte XML-Datei wird nun wieder in den Container eingefügt. Damit ist das Wasserzeichen im vollständigen Office-Dokument enthalten.

Das hier beschriebene Wasserzeichen hängt lediglich vom Dokumentschlüssel ab. Bei einer vollständigen Umsetzung für einen Online-Marktplatz muss, wie zuvor geschildert, der Ablauf der ausgetauschten Nachrichten durch ein Protokoll festgelegt werden. Aus diesen Buyer-Seller-Protokollen ergeben sich weitere Eingaben für die Hash-Funktion, die das Wasserzeichen erzeugt. Zum Beispiel kann ein solches Protokoll erfordern, dass eine Nachricht, deren korrekter Inhalt nur vom Betreiber erzeugt werden kann, mit in die Hash-Funktion einfließt. Damit würde sichergestellt, dass zwei Parteien den Ersteller eines Dokuments als solchen ausweisen und der Ersteller dies nicht abstreiten kann[8]. Derartige zusätzliche Informationen erfordern keine weitere Änderungen des hier vorgestellten Schemas und werden daher hier nicht weiter betrachtet.

5.1.4. Verifikation

Der Autor beweist, dass er $\hat{D}$ verfasst hat, indem er den Dokumentschlüssel K_D herausgibt. Dies ist notwendig, weil das Wasserzeichen nur mit dem richtigen Schlüssel erzeugt werden kann. Der Autor kann aber seinen Hauptschlüssel weiterhin sicher verwahren. Der Hauptschlüssel kann durch die Herausgabe des Dokumentschlüssels nicht kompromittiert werden. Nun wir die Verifikation durchgeführt. Dazu wird erst wieder das Hauptdokument aus dem Container extrahiert.

Schritt 1 Es wird erneut der Rang $o_i = h_2\left(K_D, M\left(s_i\right)\right)$ für die Teilbäume des Dokuments berechnet. Bei dem gleichen Dokument und korrektem Dokumentschlüssel führt dieser Schritt zu derselben Rangfolge wie bei der Einbettung.

Schritt 2 Es wird eine $(n + d_{\text{max}})$-lange Bitfolge W' berechnet, indem $I(s)$ auf die $n + d_{\text{max}}$ Teilbäume mit den niedrigsten Rängen angewendet wird. Dabei ist d_{max} ein Schwellwert, der als Toleranzgrenze dient: Er gibt an, wie viele Bits des Wasserzeichens durch Bearbeitungen verändert werden dürfen, damit das Wasserzeichen im nächsten Schritt noch als gültig betrachtet wird (s. u.).

8 Wie bereits in Kapitel 2 beschrieben, reicht es nicht aus, dass ein Ersteller alleine ein Wasserzeichen in sein Dokument einbringt.

Schritt 3 Wenn ein Präfix W'' für W' existiert, so dass $D_{\text{Lev}}\left(W'', h_1\left(K_D\right)\right) \leq d_{\text{max}}$, ist das Wasserzeichen verifiziert[9]. Ansonsten ist das Wasserzeichen nicht verifiziert[10].

5.1.5. Beispiel

Als Beispieldokument wird an dieser Stelle eine Abschrift von Hugo Balls Dada-Gedicht *Karawane* [Bal17] verwendet. Dieser Text ist aus verschiedenen Gründen für das Beispiel sehr gut geeignet: Er enthält in der Originalveröffentlichung mehrere Zeilen mit verschiedenen Formatierungen. Das ist sehr selten für einen so kurzen Text (weniger als eine A4-Seite). Im vorliegenen Fall wurde der Wortlaut des Originals übernommen, nicht aber die Originalschriftarten. Das grundlegende Gestaltungsprinzip wurde jedoch beibehalten. Als Vorlage wurde der Text aus dem Projekt Wikisource verwendet. Dort findet sich auch eine eingescannte Fassung des Originaltextes. Abbildung 5.2(a) zeigt einen Screenshot des unveränderten Originaldokuments. Aus diesem Dokument wird jetzt die Hauptdatei `content.xml` verwendet.

Zunächst wird eine Pseudozufallszahl als Hauptschlüssel erzeugt. Für dieses Beispiel ist dieser Schlüssel in Tabelle 5.3 angegeben (K). Mit dem Hauptschlüssel wird nun ein Dokumentschlüssel generiert, in dem $K_D = h_0(K||D)$ berechnet wird. Es wird also mit SHA-1 über den Hauptschlüssel und das Dokument gehasht. Das Ergebnis ist K_D aus Tabelle 5.3.

Hauptschlüssel	K	$36507b6cac86b1cfe72ffd6d5ddb3f0eda5933ef$
Dokumentschlüssel	K_D	$63f58774e4f86ef16b30a2533aa463ba6c318d75$
Wasserzeichen	W	$(1, 0, 1, 1)$

Tabelle 5.3.: *Schlüssel und Wasserzeichen*

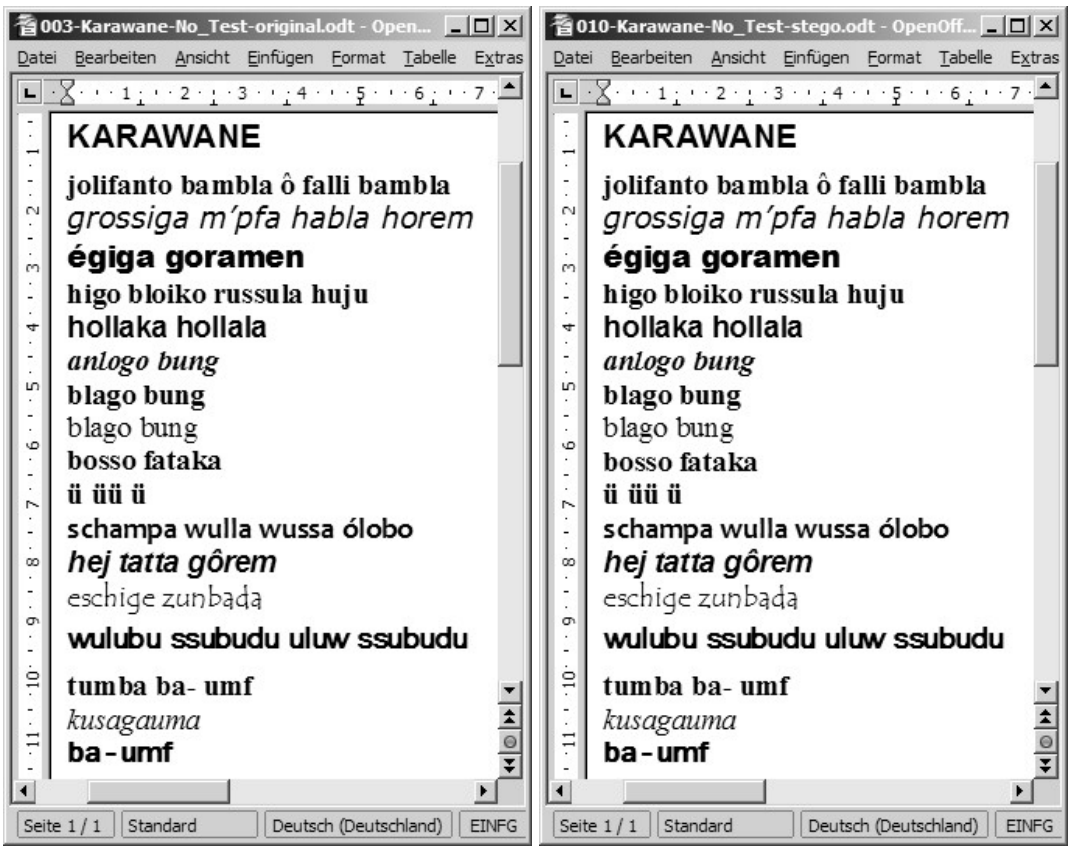
Angenommen, mit h_1 und dem Dokumentschlüssel würde ein Wasserzeichen erzeugt, das nur vier Bit lang ist: $W = (1, 0, 1, 1)$. Dann bestünde der erste Schritt darin, die Teilbäume im XML-Dokument zu sortieren. Dies geschieht abhängig vom Dokumentschlüssel. Dabei werden alle Teilbäume ignoriert, zu denen keine darstellungsäquivalente Variante existiert. Ein relevanter Ausschnitt der Teilbäume des Dokuments wird in Anhang A.1 aufgelistet.

Die Darstellungsrepäsentation for XML-Formatierungsauszeichnungen erfolgt nicht einfach durch eine Tabelle sondern ist das Ergebnis eines Funktionsaufrufs[11]. Für

9 In der Veröffentlichung von Wu und Stinson wird an dieser Stelle fälschlicherweise der Hauptschlüssel K_D zur Verifikation verwendet, das ist, wie die Autoren bestätigten, jedoch ein Tippfehler.

10 Was bedeutet, dass nicht erkannt werden konnte, dass das Wasserzeichen eingebettet wurde.

11 Wie genau diese Funktion arbeitet, wird zu einem späteren Zeitpunkt dieses Kapitels beschrieben.



(a) Cover-Dokument (b) Stego-Dokument

Abbildung 5.2.: *Die beiden darstellungsgleichen Dokumente in OpenOffice*

dieses Beispiel ist nur wichtig, dass auf den Dokumentschlüssel und die Rückgabe dieser Funktion M die Hashfunktion h_2 angewendet wird. Angenommen, für den Teilbaum mit dem Absatz „blago bung“ ergibt die Berechnung von $h_2(K_D||M_s)$ den Wert $67f4fb02b9c42ae59ff7d3f5e2b16c9c89b437cb$, der im Beispiel auf die letzten vier Zeichen, also $37cb$, gekürzt wird. Anschließend werden alle Rangwerte aufsteigend sortiert. Das Ergebnis ist die Rangfolge in Tabelle 5.4. In der Tabelle ist zur erhöhten Übersichtlichkeit nur der Text des jeweiligen Absatzes angegeben. Es ist aber sehr gut zu erkennen, dass die beiden Absätze mit dem Text „blago bung“ unterschiedliche Rangwerte erhalten, weil die Formatierungsauszeichnungen unterschiedlich sind.

Nachdem die Sortierung jetzt feststeht, werden die n Teilbäume (also vier, da das Wasserzeichen vier Bit lang ist) mit dem kleinsten Rang mit je einem der n-Bits des Wasserzeichens markiert (also die ersten vier in Tabelle 5.4). Dabei wird jeder dieser Teilbäume mit der Darstellungs-Tabelle verglichen. Ein Teilbaum, dessen Bit-

Rang	Hash	Teilbaum
1	$00eb$	hollaka hollala
2	$1aa7$	tumba ba- umf
3	$214d$	wulubu ssubudu uluw ssubudu
4	2396	bosso fataka
5	$34a6$	jolifanto bambla ô falli bambla
6	$36bb$	blago bung
7	$37cb$	blago bung
8	$4e14$	grossiga m'pfa habla horem
9	6564	hej tatta gorem
10	6630	eschige zunbada
11	$85f5$	ba - umf
12	$a611$	kusagauma
13	$b320$	schampa wulla wussa olobo
14	$b7e6$	ü üü ü
15	$bd29$	egiga goramen
16	$e97b$	anlogo bung
17	$ea49$	KARAWANE
18	$f7c2$	higo bloiko russula huju

Tabelle 5.4.: *Sortierte Teilbäume aus dem Beispiel*

Interpretation nicht der des einzubettenden Bits des Wasserzeichens entspricht, wird durch die darstellungsgleiche Variante aus der Tabelle ersetzt. Anschließend ist das Wasserzeichen eingebettet. Das Ergebnis des XML-Codes ist in Anhang A.2 aufgeführt. Wie man an Abbildung 5.2 sieht, sind beide Dokumente darstellungsgleich.

Zur Verifikation wird zunächst $d_{\text{max}} = 1$ gewählt. Damit darf das Wasserzeichen maximal um ein Bit abweichen. Jetzt wird aus dem Dokumentschlüssel ein Wasserzeichen erzeugt: $W' = (1, 0, 1, 1)$.

Nun wird die Sortierfolge der Teilbäume bestimmt. Dazu wird wie vor der Einbettung für jeden Teilbaum s die Funktion M_s berechnet, aus deren Ergebnis mit der Hash-Funktion h_2 ein eindeutiger Wert bestimmt wird. Die Implementation der Funktion M gewährleistet, dass darstellungsgleiche Varianten eines Teilbaumes die gleiche Ausgabe erzeugen. Damit ergibt sich wiederum die Sortierfolge aus Tabelle 5.4. Jetzt werden die $n + d_{max}$ Teilbäume (also fünf) mit dem kleinsten Rang aus der sortierten Tabelle 5.4 ausgewählt. Für jeden der Teilbäume wird $I(s)$ ermittelt. Das Ergebnis steht in Tabelle 5.5.

Das ausgelesene Wasserzeichen W'' ergibt sich aus den fünf Werten für I_s in der Reihenfolge des Ranges von s, also $W'' = (1, 0, 1, 1, 0)$. Der Levenshtein-Abstand von W'' und dem vorgelegten Wasserzeichen W' ist 1, das eingebettete Wasserzeichen

Rang	Teilbaum	$\mathbf{I}(\mathbf{s})$
1	hollaka hollala	1
2	tumba ba- umf	0
3	wulubu ssubudu uluw ssubudu	1
4	bosso fataka	1
5	jolifanto bambla ô falli bambla	0

Tabelle 5.5.: *Teilbäume und numerischer Wert der Varianten*

unterscheidet sich also von dem vorgelegten Wasserzeichen um eine Stelle. Dieser Unterschied ist aufgrund der Wahl von $d_{max} = 1$ zulässig. Das Wasserzeichen ist somit gültig.

5.2. Technische Umsetzung

Alle wesentlichen Dateien im Container sind XML-Dateien. In Microsoft Word-Dateien ist das Hauptdokument eine Datei namens `document.xml` welches sich in einem Unterverzeichnis `word` befindet. Bei OpenOffice.org Writer ist das Hauptdokument direkt im Wurzelverzeichnis des Containers abgelegt und heißt `content.xml`. Abhängig von der Dokumentenart (Text, Präsentation, Tabelle...) können weitere Dateien im Container enthalten sein. Für nähere Informationen sei an dieser Stelle auf die jeweilige Dokumentation verwiesen. Außerdem gibt es noch Konfigurations- und Meta-Dateien, beispielsweise die Datei `settings.xml` im Fall von OpenOffice. Am sinnvollsten ist jedoch, Wasserzeichen direkt auf das Hauptdokument und damit den Dokumentinhalt anzuwenden. Schließlich soll in den meisten Fällen das Hauptdokument geschützt werden[12].

5.2.1. Flüchtige Markierungen

Es gibt extrem flüchtige Verfahren (siehe Abschnitt 2.2.5.1.2), um Markierungen in XML-basierte Office-Dokumente einzubetten. Dies ist dem Umstand geschuldet, dass beim Speichern eines Dokuments die gesamte XML-Struktur von der Anwendung erneut erstellt wird. Mit solchen Verfahren eingebettete Markierungen werden dann komplett entfernt. Die Flüchtigkeit der Wasserzeichen ist in diesem Fall eher durch die Office-Anwendung als durch die Einbettungsmethode verursacht. Auch wenn dies augenscheinlich als Nachteil erscheint, gibt es dennoch Anwendungen, in denen solche Techniken sinnvoll sein können. Entweder dort, wo Dokumente nach

12 Das ist eine Annahme, zu der keine repräsentative Umfrage durchgeführt wurde.

der Erstellung nicht mehr verändert werden (dann können solch flüchtige Markierungen durch einen Postprozessor eingefügt werden), oder in gerade solchen Fällen, wo Änderungen an Dokumenten gerade aufgespürt werden sollen. Sind in einem solchen Fall die Markierungen nicht mehr korrekt im Dokument eingebettet, kann geschlossen werden, dass es verändert wurde.

Flüchtige Markierungen können zum Beispiel im Zusammenhang mit XML-Dokumenten interessant sein, die digital signiert werden. In diesem Fall wird die digitale Signatur ja nicht direkt für den XML-Code erzeugt. Stattdessen wird zunächst eine Kanonisierung durchgeführt, die Ausgabe wird dann signiert (siehe Abschnitt 4.1.2). Verschiedene gültige Formen eines XML-Dokuments können zur gleichen kanonischen Form führen. Ein nicht-robustes Wasserzeichen kann dem Benutzer die zusätzliche Sicherheit geben, dass das Dokument tatsächlich nicht bezüglich des XML-Codes verändert wurde, sollte das Wasserzeichen sich bei der Validierung der Signatur noch im Dokument befinden.

Es gibt deutlich mehr Möglichkeiten, Informationen flüchtig in XML-basierten Office-Dokumenten einzubetten, als robust. An dieser Stelle sollten aber nur zwei gute Ansätze festegehalten werden, da robuste Einbettungen für die Einbettung von Wasserzeichen wichtiger sind.

5.2.1.1. Flüchtige Markierungen nach Inoue et al.

Zu den flüchtigen Methoden gehören z. B. auch der bereits in Kapitel 4 diskutierte Vorschlag von Inoue et al. [IMM+01]. Nicht in dem Paper von Inoue geschildert, aber ebenfalls denkbar ist eine Variante, bei der XML-Elemente verwendet werden, die nicht dem jeweiligen Standard entsprechen und vom System ignoriert werden (Dies ist in beiden Standards vorgesehen, da es sich um zukünftige oder proprietäre anwendungsbezogene Erweiterungen handeln könnte). Um auch bei einer Begutachtung des XML-Codes durch einen Benutzer keinen Verdacht zu erregen, können diese unauffällige Bezeichner erhalten. Dann können damit auch zusätzliche, flüchtige Informationen eingebracht werden.

5.2.1.2. Synonyme Styles

Ein eigener Vorschlag zur Einbettung von flüchtigen Markierungen ist die Einführung synonymer Styles. Dabei wird eine Kopie eines Styles mit einem neuen Bezeichner angelegt. Zum Beispiel wird in Listing 5.5 zum Style namens „P1“ ein identischer Style „P3“ angelegt, der allerdings einen anderen Übergeordneten Style zugewiesen bekommt. Demnach erbt „P1“ seine Attribute von „Standard“, „P3“ jedoch von „P2“.

Hier ist natürlich wichtig, das in „P3“ alle Attribute abgeschaltet werden müssen, die „P2“ zusätzlich zu „Standard“ hat.

```
<style:style style:name="P1" style:family="paragraph"
   style:parent-style-name="Standard">
    <style:text-properties style:font-name="Arial"/>
</style:style>
<style:style style:name="P2" style:family="paragraph"
   style:parent-style-name="Standard">
    <style:text-properties style:font-name="Arial" fo:font-
       weight="bold" style:font-weight-asian="bold"
       style:font-weight-complex="bold"/>
</style:style>
<style:style style:name="P3" style:family="paragraph"
   style:parent-style-name="P2">
    <style:text-properties style:font-name="Arial"/>
</style:style>
```

Listing 5.5: *Einfügen eines zusätzlichen Absatzstils.*

Nun kann das Dokument aus dem bereits bekannten Beispieldokument „Einige Dichtungsarten.odt“ wie in Listing 5.6 verändert werden. Durch zuweisen von „P3“ anstelle von „P1“ als Absatzstil des dritten Absatzes, wurde eine Information in dem kurzen Dokument untergebracht.

```
<text:p text:style-name="P2">Einige Dichtungsarten:</text:p>
<text:p text:style-name="P1">Rundschnur</text:p>
<text:p text:style-name="P3">Formteile</text:p>
<text:p text:style-name="P1">O-Ringe</text:p>
```

Listing 5.6: *Ein Ausschnitt aus Einige Dichtungsarten.odt, mit veränderten darstellungsgleichen Absatzstilen.*

Da der direkte Nachfolger des Absatzes nicht „P1“ ist, sondern erst dessen Nachfolger, bezüglich der Formatierungsanweisungen identisch mit „P1“ ist, verbleiben die Styles nach einem Speichern im Dokument. Die Formatierung des Absatzes indes wird nach dem Speichern wieder durch „P1“ ersetzt. Dieses Verfahren ist der ebenfalls flüchtigen Methode von Inoue aufgrund eben dieser Tatsache überlegen. Es kann belegt werden, dass die Methode angewendet wurde (der Style ist noch im Dokument enthalten), aber das Dokument anschließend verändert wurde (die Absatzauszeichnungen sind nicht mehr vorhanden).

5.2.2. Beständige Markierungen

Beständige Verfahren sollen eine möglichst dauerhafte – also robuste – Einbettung eines Wasserzeichens in ein Dokument gewährleisten. Bei der Umsetzung gibt es im Kern zwei Herausforderungen, die bewältigt werden müssen. Erstens reicht es nicht, XML-Teilbäume durch festgelegte darstellungsäquivalente Teilbäume zu ersetzten, etwa mittels einer Tabelle. Zweitens führen viele der möglichen darstellungsäquivalenten Auszeichnungen beim Einlesen eines Dokuments nicht nur zur gleichen gerenderten Ansicht, sondern auch zur gleichen internen Repräsentation durch die Anwendung. Das bedeutet, dass sie bei einem erneuten Speichern auch zu dem gleichen XML-Dokument führen. Die Transformationen und die internen Abläufe der Anwendung sind nicht Teil der veröffentlichen Spezifikation des Open Document Formats. Der XML-Code wird dadurch jedenfalls „normalisiert“, und mit den Auszeichnungsvarianten gehen auch mögliche Einbettungen verloren. Ein Teil solcher Auszeichnungsvarianten ist im Abschnitt 5.2.1 beschrieben.

Die erste Hürde lässt sich nur beheben, indem die Formatierungen (also die XML-Attribute) semantisch verstanden werden. Der bei der Einbettung oder Verifikation eingesetzte Parser muss die Formatierungen in den Textauszeichnungen erfassen und eine Ausgabe erzeugen können, die wirklich darstellungsgleich ist. Erschwerend kommt hinzu, dass die Styles an einer anderen Stelle im XML-Dokument stehen als sie verwendet werden. Der Auszeichnung im Text kann die Darstellung nicht direkt entnommen werden – dazu müssen zunächst die Formatierungsanweisungen aus den dazugehörigen Style ausgelesen werden. Die Bezeichner der Styles sind dabei dynamisch, sie werden beim Speichern des Dokuments automatisch angelegt. Der Parser muss also die Styles anhand ihrer Darstellungsaspekte identifizieren und die Styles bei Bedarf verändern, wenn im Text Auszeichnungen angepasst werden.

Dabei darf natürlich nie außer Acht gelassen werden, dass die Formatierungen zum einen Korrekt sind und zum anderen auch wirklich darstellungsäquivalent. So müssen zum Beispiel Abstatzstyles von Absatzformatierungen unterschieden werden. Absatzstyles können neben Absatzformatierungen auch Textformatierungen enthalten. Textstyles hingegen können keine spezifischen Absatzformatierungen enthalten wie etwa Einzüge, Ausrichtung, Überhang, Absatzabstände und Zeilenabstände. Die Styles können also nicht beliebig verändert oder „vermengt“ werden. Darüber hinaus gibt es Formate, die sich nicht nur auf die Darstellung auswirken. Überschriften dienen zum Beispiel dazu, einem Dokument eine logische Struktur zu geben. Diese Struktur kann in einem Inhaltsverzeichnis wiedergegeben werden. Würden die Überschriften stattdessen durch darstellungsgleich formatierte Absätze ausgezeichnet werden, ginge diese logische Struktur verloren.

Ein weiteres Beispiel: Ein Absatz sei mit einem Style formatiert, der eine größere Schriftart und kursive Typen verwendet. Angenommen, diese Auszeichnung wird

zwecks Informationseinbettung so transformiert, dass die gleichen Formatierungsanweisungen einem Textstil zugewiesen würden, der Standard-Absatzstil verwendet wird und jedes Wort einzeln – also mit Aussparung der Leerzeichen – mit dem Textstil ausgezeichnet wird. Dann wäre dieser Absatz nach der Transformation nicht mehr darstellungsgleich, weil die Leerzeichen nicht mehr in der richtigen Schriftgröße formatiert wären. In der Praxis muss das keine wesentlichen Auswirkungen haben, vor allem wenn sie die beiden Schriftgrößen nur geringfügig unterscheiden. Aber im schlimmsten Fall führt die Änderung des Wortabstandes dazu, dass Zeilen eine andere Laufweite haben und anders umgebrochen werden[13].

Bei den folgenden Lösungsvorschlägen für robuste Einbettungstechniken in XML-basierten Office-Dokumenten wird vorausgesetzt, dass die hier geschilderten Aspekte bei der Umsetzung berücksichtigt werden. Dabei erhebt die Auflistung dieser Aspekte – ebensowenig wie die nachhfolgenden Lösungsvorschläge – Anspruch auf Vollständigkeit.

5.2.2.1. Transformation von Absatzformatierungen

Die in diesem Abschnitt beschriebene Methode ist schon zuvor im Beispiel angedeutet worden. Dabei werden Absatzformatierungen einem Textbereich – nämlich dem gesamten Absatz mit Ausnahme mindest eines Leerzeichens – zugewiesen. Der Absatz selber wird dann wie ein Übergeordneter Style, im Normalfall als „Standard“, formatiert.

```
<text:p text:style-name="P2">Einige Dichtungsarten:</text:p>
```

Listing 5.7: *Der unveränderte Ausschnitt aus Einige Dichtungsarten.odt*

In Listing 5.7 ist der Teilbaum im Original gezeigt, die Veränderungen sind in Listing 5.8 hervorgehoben.

```
<text:p text:style-name="P1">
<text:span text:style-name="T1">Einige</text:span> <text:span
    text:style-name="T1">Dichtungsarten:</text:span></text:p>
```

Listing 5.8: *Der veränderte, aber darstellungsgleiche Ausschnitt des Dokuments*

13 Auch wenn dieser Umstand dem Betrachter nur auffällt, wenn er das Stego-Dokument mit dem Cover-Dokument vergleichen kann, ist die Darstellung der beiden Dokumente eben nicht darstellungsgleich.

Der Vorteil beim Verändern von Absatz-Styles im Vergleich zur Anpassung von Zeilen- und Wortabständen nach der Methode von [AA04] (siehe Abschnitt 4.3.1) ist, dass keine neuen Formatierungen verwendet werden, die der Benutzer nicht selber angelegt hat, und dem versierten oder gewissenhaften Benutzer auffallen. Stattdessen werden Styles angepasst und verändert. Die verwendeten Formatierungen hat der Benutzer also selber erzeugt; lediglich die Zuteilung zu den Styles, die aber ohnehin von der Textverarbeitung automatisch vorgenommen wird, wurde angepasst. Selbst ein bei einem versierten Benutzer dürfte so kein Misstrauen erregt werden.

5.2.2.2. Invertierung von Formatierungsanweisungen

Im Gegensatz zur oben beschriebenen Methode werden hier bereits bestehende Auszeichnungen vertauscht. Es werden also keine neuen Textstile angelegt. Listing 5.9 zeigt einen unveränderten Absatz; Listing 5.10 zeigt die darstellungsgleiche Variante.

```
<text:p text:style-name="P1">Einige <text:span text:style-
   name="T1">Dichtungsarten:</text:span></text:p>
```

Listing 5.9: *Der unveränderte Ausschnitt aus Einige Dichtungsarten.odt*

```
<text:p text:style-name="P2"><text:span text:style-name="T2">
   Einige </text:span>Dichtungsarten:</text:p>
```

Listing 5.10: *Der veränderte, aber darstellungsgleiche Ausschnitt des Dokuments*

Die Styles müssen dabei angepasst oder neu erstellt werden, falls dieselben Styles auch in anderen Abstätzen verwendet werden.

5.2.2.3. Einfügen von Leerzeichen in den Character Data

Im Open Document Format werden beim Rendering eine Folge von Leerzeichen in den Character Data zu einem normalisiert. Sollen trotzdem Leerzeichen angezeigt werden, muss dafür ein besonderer XML-Tag benutzt werden [OAS07, Abschnitt 5.1.1]:

```
<text:s text:c="74"/>
```

Listing 5.11: *Einbetten mehrerer Leerzeichen*

In diesem Beispiel werden also 74 Leerzeichen angezeigt (Das Attribut `text:c` ist optional). Diese Leerzeichen sind im Text sichtbar. Sie fallen dem Benutzer nur auf, wenn er die Sonderzeichen anzeigen lässt, oder er mit dem Cursor über die Textstelle „navigiert“. Werden am Ende eines Absatzes ein oder zwei Leerzeichen eingefügt, können diese aber als Fehler durchgehen und fallen dem Leser kaum auf.

5.2.2.4. Anwendung bedingter Formatierungen

Insbesondere bei Tabellendokumenten können bedingte Formatierungen benutzt werden, um Informationen zu verbergen. Bedingte Formatierungen sind Formatierungen, die nur unter festgelegten Bedingungen zu einer definierten Darstellung führen. Es gibt zwei Möglichkeiten, diesen Umstand zur Informationseinbettung auszunutzen. Entweder können Bedingungen festgelegt werden, die niemals zutreffen können (Kontradiktionen), so dass die Formatierungen in der gerenderten Darstellung des Dokuments niemals auftreten. Oder die Formatierung, die bei erfüllter Bedingung angewendet wird, entspricht genau der Formatierung, welche bei nicht-Erfüllung angewendet wird.

```
<style:style style:name="ce2" style:family="table-cell"
   style:parent-style-name="Default">
        <style:map style:condition="cell-content()&gt;1"
           style:apply-style-name="Default" style:base-cell-
           address="Tabelle1.C3"/>
</style:style>
```

Listing 5.12: *Bedingte darstellungsgleiche Formatierung (Style-Section).*

In Listing 5.12 ist ein Auszug der Style-Section eines Tabellendokuments wiedergegeben. In diesem Testdokument wird der zweite Vorschlag umgesetzt. Die Bedingung (`style-condition`) besagt, dass die Formatierung angewendet wird, sobald der Zelleninhalt größer als 1 ist. In dem Fall wird anstelle des `parent-style-name` der `apply-style-name` zur Formatierung verwendet. Beide sind identisch, nämlich „Default“. Der Auszug aus Listing 5.13 zeigt die Formatierungszuweisung der konkreten Zelle im Dokument und Abbildung 5.3 einen Screenshot der Tabelle. Die Tabelle mit der Einbettung ist nicht von den Nachbarzellen zu unterscheiden.

5.2.3. Umsetzung der Darstellungsrepräsentation

Sowohl für die Einbettung als auch die Verifikation wird ein Hashwert einer Darstellungsrepräsentation von XML-Teilbäumen (oben als Funktion M abgekürzt) ver-

```
<table:table-cell table:style-name="ce2" office:value-type="
   float" office:value="4">
        <text:p>4</text:p>
</table:table-cell>
```

Listing 5.13: *Bedingte darstellungsgleiche Formatierung (Tabellenzelle).*

	A	B	C	D
1	**Artikel**	**Art.-Nr.**	**Anzahl**	
2	Rundschnur	100001	60	
3	Formteil	100002	4	
4	O-Ring	100003	2043	
5				

Abbildung 5.3.: *Zelle C3 ist trotz bedingter Formatierung darstellungsgleich.*

wendet, um eine eindeutige Sortierfolge für Teilbeäume festzulegen. In der Originaldomäne ist die Funktion der Bedeutungsrepräsentation so beschrieben, dass jedes ersetzbare Wort eines Satzes durch seine numerische Bedeutungsrepräsentation ersetzt wird. Dieser Satz wird dann gehasht.

Die Schwierigkeit einer analogen Funktion für XML-basierte Formatierungsauszeichnungen liegt darin, dass die nicht alle Transformationen statisch wie mit einem Wörterbuch beschrieben werden können. Um eine dokumentunabhängige darstellungsgleiche Beschreibung sicherzustellen, muss man sämtliche Darstellungsaspekte jedes einzelnen Zeichens des Teilbaumes beschreiben, die nach Spezifikation des Formats möglich sind. Dies erfordert neben einer komplexe Beschreibung dieser Darstellungsaspekte auch einen komplexen Parser, um die Dokumente entsprechend ein- und ausgeben zu können.

Eine exakte Darstellungsrepräsentation ist aber gar nicht unbedingt notwendig. Im Originalverfahren von Wu und Stinson wird die Bedeutungsrepräsentation deshalb verwendet, weil sie für die Funktionalität des Schemas wichtige Eigenschaften erfüllt:

1. Die Bedeutungsrepräsentation der ersetzten Wörter stellt sicher, dass sowohl der unveränderte als auch der veränderte Satz zum gleichen Hash-Wert führen.
2. Der Hash über den gesamten Satz bezieht den Kontext der ersetzten Wörter mit in den Hashwert und somit in den Schutz des Wasserzeichens ein. Ansonsten könnten alle Wörter, die keine Bedeutungsrepräsentation hätten, im Nachhinein verändert werden.

Für XML-Dokumente lässt sich diese Funktionalität auch nachbilden, in dem die Funktion M beide Fassungen eines Teilbaumes zurück gibt. Dabei muss die Reihenfolge invariant sein. Zum Vergleich, dass die oben geschilderte Funktionalität gewährleistet ist:

1. Die Darstellungsrepräsentation ist identisch bei Einbettung und Verifikation, da in beiden Fällen die gleiche Ausgabe zurückgegeben wird.
2. Der Hash über diese Ausgabe enthält auch den Kontext, da der gesamte Teilbaum auch die Character Data enthält. Somit ist der Vollständige Teilbaum durch das Wasserzeichen geschützt.

Es kann also auf eine vollständige Darstellungsrepräsentation verzichtet werden. Ist zum Beispiel das ein Teilbaum im Original wie in Listing 5.14 und die darstellungsgleiche Variante in Listing 5.15, dann gleicht die Ausgabe der Funktion M Listing 5.16.

```
<text:p text:style-name="P1">Einige Dichtungsarten:</text:p>
```

Listing 5.14: *Ein Teilbaum in der ersten darstellungsgleichen Variante.*

```
<text:p text:style-name="P1"><text:span text:style-name="T1">
   Einige</text:span> <text:span text:style-name="T1">
   Dichtungsarten:</text:span></text:p>
```

Listing 5.15: *Ein Teilbaum in der zweiten darstellungsgleichen Variante.*

```
<text:p text:style-name="P2">Einige Dichtungsarten:</text:p><
   text:p text:style-name="P1"><text:span text:style-name="T1
   ">Einige</text:span> <text:span text:style-name="T1">
   Dichtungsarten:</text:span></text:p>
```

Listing 5.16: *Die Ausgabe der Funktion M für zwei darstellungsgleiche Varianten.*

Damit sind alle grundlegenden Aspekte beschrieben, die für ein Verfahren für Information Hiding in XML-basierten Office Dokumenten benötigt werden.

5.3. Angriffe auf eingebettete Informationen und Schutzmaßnahmen

Im Szenario aus Kapitel 3 ist der wichtigste Angriff auf das dort beschriebene System OnGebot das Kopieren von Dokumenten (Abschnitt 3.4.1). Dabei wird das Dokument des ursprünglichen Erstellers *(Alwin Tender)* als eigenes Dokument wieder in das System eingestellt. Bei unveränderter Weiterverwendung von Ausschreibungsunterlagen durch *Evegasket* wären Wasserzeichen von *Alwin Tender* weiterhin im Dokument. Dann ist offentsichtlich, dass *Evegasket* nicht die Erstellerin ist. Kann *Evegasket* das Wasserzeichen entfernen und durch ein eigenes ersetzen, ist ein solcher Angriff aber weiterhin möglich. In diesem Abschnitt werden Angriffe auf eingebettete Methoden beschrieben – und wie sie verhindert oder eingeschränkt werden.

5.3.1. Präparierte Dokumente

Dokumente, die im Vorweg von *Evegasket* präpariert werden, bevor *Alwin Tender* sie benutzt, werden durch die im Szenario geschilderten Abläufe nicht zugelassen. Damit sind sämtliche Angriffe ausgeschlossen, die ein vorheriges Präparieren voraussetzen. Es müssen weiterhin Angriffe auf Dokumente abgewägt werden, die bereits im System abgelegte Dokumente betreffen.

5.3.2. Additive Angriffe

Bei einem Additiven Angriff bringt *Evegasket* ihre eigenen Markierungen in ein Dokument ein. In diesem Dokument müssen aufgrund der im vorigen Abschnitt bereits beschriebenen, im Szenario festgelegten Abläufe, bereits Wasserzeichen von *Alwin Tender* enthalten. Additive Angriffe auf Dokumente ohne existierende Wasserzeichen brauchen daher nicht berücksichtigt werden[14]. Da die Position der Markierungen vom Dokumentschlüssel abhängt, steigt mit der Dokumentgröße auch die Wahrscheinlichkeit, dass bei einer erneuten Einbettung eines Wasserzeichens nicht die bereits durch das existierende Wasserzeichen markierten Elemente erneut markiert werden. Das kann nur erreicht werden, wenn *Evegasket* in den Besitz des Master- oder Dokumentschhlüssels von *Alwin Tender* ist. In dem Fall kann versucht werden, ein Wasserzeichen zu finden, das eine möglichst hohe Übereinstimmmung mit dem eingebetteten hat (Kollision). Ab einer ausreichenden Länge des Wasserzeichens ist dies aber durch die Verwendung kryptographischer Hashfunktionen (SHA-1 oder besser) bei der Erzeugung des Wasserzeichenz nach derzeitig bekannnten Angriffen nicht

14 Technisch wäre es in einem solchen Fall natürlich einfach, ein eigenes Wasserzeichen einzubetten.

rechnerisch möglich. Lediglich in Kombination mit einem subtraktiven Angiff kann das Wasserzeichen möglicherweise entfernt werden – solche Angiffe werden in Abschnitt 5.3.3 gesondert betrachtet.

Damit verbliebe das Wasserzeichen des echten Erstellers *Alwin Tender* also mit hoher Wahrscheinlichkeit zu wesentlichen Teilen im Dokument, weil dessen Markierungen nicht gezielt überschrieben werden können. Bei einer Verifikation würden dann möglicherweise beide Wasserzeichen erfolgreich validiert. Allerdings stellt in dem im Szenario beschriebenen System zusätzlich ein Buyer-Seller-Protokoll sicher, dass eine weitere Partei außer dem Ersteller in die Erzeugung des Wasserzeichens involviert werden muss. Damit ist in einem solchen Fall das nachträglich einbettete Wasserzeichen als invalide erkennbar, weil dieses eben nicht mit Kenntnis der anderen Partei erzeugt wurde[15]. Es ist also für einen Angreifer schon aufgrund des Einsatzes solcher Protokolle nicht ohne weiters möglich, ein gültiges Wasserzeichen zu einem beliebigen Dokument zu generieren.

Ein additiver Angriff ist also aus Sicht eines Angreifers somit nicht die sinnvollste Variante. *Evegasket* muss stattdessen zunächst alle auffindbaren Markierungen aus dem Dokument entfernen, bevor sie ihr eigenes Wassserzeichen einbettet (subtraktiver Angriff).

5.3.3. Subtraktive Angriffe

Subtraktive Angriffe auf Wasserzeichen zielen darauf ab, die Markierungen im Dokument zu entfernen. Es gibt verschiedene Varianten solcher Angriffe, die diskutiert werden müssen. Darunter fallen Sonderfälle, wie etwa die Konvertierung des Dokuments in ein anderes Format, der im nächsten Abschnitt behandelt wird. Zunächst einmal gilt es zu unterscheiden, ob der Angreifer im Besitz des zugehörigen Dokumentschlüssels ist oder nicht. Im ersten Fall ist ein subtraktiver Angriff wesentlich einfacher. Der Angreifer kann dann die Teilbäume des Dokuments anhand des Schlüssels sortieren und somit die markierten Elemente bestimmen. Im zweiten Fall müssen die Markierungen zunächst aufgespürt werden. Das Auffinden der markierten Stellen wird dadurch erschwert, dass ab einer ausreichenden Dokumentgröße[16]

15 Das in dieser Arbeit vorgestellte Verfahren soll mit verschiedenen Protokollen eingesetzt werden können; es ist daher kein spezifisches Protokoll festgelegt worden. Es ist aber eine vertretbare Annahme, dass eine dritte Partei, die in einem solchen Protokoll zum Beweis der Echtheit an der Einbettung eines Wasserzeichens beteiligt ist, das Dokument vor der Einbettung vorgelegt bekommt.

16 Wie groß ein Dokument sein muss, um ausreichend markierbare Stellen zu enthalten, hängt ab von der Art des Dokuments und welche Formatierungen im Dokument enhalten sind. Der einseitige Dr. Grauert-Brief enthält nach seiner Konvertierung ins Open Document Format 54 markierbare Absätze, was für ein sichers Wasserzeichen zu wenig wäre.

eine größere Anzahl markierbarer Elemente zur Verfügung stehen, als tatsächlich zur Einbettung verwendet werden. Ohne dass *Evegasket* die Sortierung bestimmen kann – es gibt bei n Elementen immerhin $n!$ mögliche Permutationen, so dass sich die Reihenfolge der Teilbäume bei großen Dokumenten rechnerisch nicht bestimmen lässt – muss sie jedes markierbare Element überprüfen.

Die markierbaren Elemente kann ein Angreifer ermitteln, weil das Verfahren selber öffentlich beschrieben ist (Kerkoffs' Prinzip). Da auch die Einbettungstechnik beschrieben ist, kann sie auch eine gewisse Anzahl der tatsächlich markierten Elemente erkennen. Aber eben nicht alle. Zum einen hängt das damit zusammen, dass die Elemente nur dann verändert werden, wenn das Bit, das in dem Element eingebettet werden soll, das erfordert. Dem Gesetz der großen Zahl zufolge ist die Wahrscheinlichkeit, dass ein Bit eines Wasserzeichens 0 oder 1 ist je 0.5 (Bernoulli-Experiment). Dann wäre im statistischen Mittel nur jedes zweite markierte Element überhaupt von einem nichtmarkierten Element zu unterscheiden. Noch komplizierter wird es, wenn sich bei den Auszeichnungen nicht mit Sicherheit festellen lässt, welche Variante in der Praxis normalerweise nicht vorkommen würde. Auch wenn ein Angriffer im Besitz des Dokumentschlüssels ist und damit das Wasserzeichen erzeugen kann, ist damit nicht sicher, welche Variante eines Teilbaums vor der Einbettung im Dokument stand. Dann muss *Evegasket* nämlich zunächst Untersuchungen tätigen und Daten darüber sammeln, welche Variante wahrscheinlicher ist, um ein Wahrscheinlichkeitsmodell für die Varianten der XML-Auszeichungen anzulegen. Nur so kann es einem Angreifer ein wirklich guter Subtraktiver Angriff gelingen. Es bleibt natürlich immer noch ein Brute-Force-Angriff, nämlich sämtliche „verdächtig“ wirkenden Elemente zu normalisieren.

Unabhänig davon, ob die Elemente durch ein eigenes Modell statistisch ermittelt oder einfach nach der Brute-Force-Methode oder einen gestohlenen Schlüssel identifiziert werden: Sind die markierten Elemente gefunden, müssen sie noch entfernt werden. Und dabei gilt es für *Evegasket*, das Dokument nicht zu beschädigen. Sie muss also markierte Elemente darstellungsgleich ersetzen, ähnlich wie ein Einbettungsalgorithmus.

Ein erweiterter Mechanismus, der Teile des Wasserzeichens auch in ins Dokument eingebettete Objekte (wie zum Beispiel Bilder, Videos, Tabellen und Formeln) einbettet, trägt ebenfalls zur Minderung subtraktiver Angriffe bei. Dadurch, dass diese Objekte dann in den Schutz des Wasserzeichens einbezogen wären, müssten die Markierungen bei einem Angriff auch aus ihnen entfernt werden. Für eine solche Einbettung wäre eine eigene Technik notwendig, da es sich um ein anderes Cover-Medium handelt. Ebenso müsste ein Angriff dann die spezifischen Eigenschaften dieses Mediums zusätzlich zu denen des XML-basierten Dokumentformats berücksichtigen.

5.3.4. Konvertierung in andere Formate

Eine sehr naheliegende Form subtraktiver Angriffe ist das Konvertieren in andere Formate oder Medien. Daher wird diesem Angriff ein eigener Abschnitt gewidmet. Andere Formate sind Dateiformate wie beispielsweise das abgelöste, binäre Dateiformat von Microsoft Word 2000. Tests mit der formatierungsbasierten Einbettungsmethode, wie sie in Abschnitt 5.2.2 beschrieben wurden, haben ergeben: Es ist möglich, die Dokumente in das binäre Word-Format (`*.doc`) zu konvertieren, neu zu laden und wieder als ODF-Dokument zu speichern, ohne dass die Markierungen verloren gehen. Die mit der unter Abschnitt 5.2.1 vorgestellten Technik für flüchtige Wasserzeichen eingebetteten Informationen gehen allerdings verloren, wie es auch zu erwarten ist.

Unter Konvertierung in andere Medien wird etwa eine Umwandlung der Textseiten in Bilder verstanden – unabhängig davon, ob dies per Bildschirmfoto, Exportfunktion oder Ausdrucken und Einscannen geschieht. Solche Verarbeitungsschritte überstehen die Markierungen nicht. Dies lässt sich mit XML-basierten Wasserzeichen auch nicht erreichen, weil dazu die gerenderte Ansicht des Stego-Document sich vom Cover Document zumindest geringfügig unterscheiden müsste. In der gewählten Methode wird aber gerade gewährleistet, dass das nicht der Fall ist. Das bedeutet dann auch: Konvertiert man ein Originaldokument und das Dokument mit eingebetteten Wasserzeichen jeweils in Bildformate und vergleicht diese, so stimmen beide Bilder pixelgenau überein.

5.3.5. Gewöhnliche Bearbeitung

Ein allgemeiner Schutzmechanismus gegen Angriffe ergibt sich aus der Wahl der Einbettungstechnik mittels Formatierungsauszeichnungen. Für Information Hiding gilt immer, dass Markierungen, die in für den Nutzer wichtige Stellen eingebettet werden, sich schwerer entfernen lassen als an Stellen, die Benutzer nicht oder nur wenig wahrnehmen. Der linguistische Inhalt des Textes ist für den Benutzer der wichtigste Bestandteil eines Dokuments. Ähnlich wichtig sind die Formatierung eines Dokuments für die Darstellung der Informationen, die ein Benutzer in einem Dokument vermitteln möchte. Damit ist gewährleistet, dass Angriffe erschwert werden, damit ein Angriff keine sichtbaren Artefakte hinterlässt.

5.3.6. Erkennung von Angriffen

Die Kombination der Einbettungsmehode mit einer Methode zur Einbettung flüchtiger Wasserzeichen steigert die Erkennungsmöglichkeiten von Angriffen im Nach-

hinein. Zwar können auch diese Markierungen angegriffen werden, aber das ist immerhin eine zusätzliche Hürde. Der Ablauf wäre etwa folgender: Zunächst versucht der Angreifer, die robusten Markierungen zu entfernen. Die flüchtigen Markierungen werden dabei teilweise mit entfernt. Er muss diese nun ebenfalls vollständig entfernen und anschließend eigene gültige Markierungen einfügen.

5.3.7. Angriffe durch mehreren Teilnehmer

Denkbar wäre auch, dass *Evegasket* einen Angriff während des Verifikationsprozesses versucht – und zwar zusammmen mit *OnGebot*. In diesem Fall hätte *Alwin Tender* beim Einsatz von Buyer-Seller-Protokollen, die eine Trusted Third Party (TTP) beinhalten, wie beispielsweise dem von Memon und Wong beschriebenen [MW01] nicht die Möglichkeit, die Autorenschaft gegenüber anderen Parteien zu beweisen. *OnGebot* könnte schlichtweg behaupten, das Wasserzeichen wäre ungültig. Man muss daher bei Protokollen dieser Art vorraussetzen, dass diese dritte Partei auch wirklich vertrauenswürdig ist. Es gibt allerdings auch Protokolle, die ohne eine TPP auskommen, wie etwa das von Choi et al. [CSP03]. Für eine Beschreibung dieses Protokolls sei auf die Veröffentlichung der Autoren verwiesen.

5.4. Beispielimplementation mit Python

Zu Demonstrationszwecken werden die beschriebenen Abläufe mithilfe der Sprache Python 2.5 [Pyt08e] umgesetzt. Die Entscheidung, Python zu verwenden, begründet sich vor allem auf dessen Eignung für schnelles Prototyping. Mit Python ist es im Vergleich zu Java möglich, mit relativ wenig programmiersprachlichen Konstrukten bereits praktische Ergebnisse zu erzielen. Darüber hinaus existieren Module, die die Arbeit mit XML sehr vereinfachen (solche Pakete gibt es allerdings auch für Java). Die verwendeten Module sind in der Standardinstallation von Python[17] bereits enthalten. Zunächst folgt eine kurze Schilderung der verwendeten Module.

5.4.1. Das Modul zipfile

In XML-basierten Office-Dokumenten befinden sich die XML-Dateien in ZIP-Containern. Um auf diese zugreifen zu können, wird das Modul `zipfile` [Pyt08d] benutzt. In ihm wird ein Objekt `ZipFile` bereitgestellt, mit dem Zip-Archive geöffnet werden können.

17 Es gibt verschiedene Quellen für das Python-Paket, abhängig vom jeweiligen Betriebssystem. Die Python-Webseite [Pyt08a] enthält im Download-Bereich Links zu verschiedenen Varianten.

```
import zipfile

# Öffnen eines Zip-Archives
archiv = zipfile.ZipFile(filename)

# Extrahieren der Datei content.xml
datei = archiv.read('content.xml')
```

Listing 5.17: *Import des Moduls zipfile und Öffnen einer Datei*

Das Beispiel-Listing 5.17 zeigt, wie ein Archiv geöffnet und eine Datei aus dem Archiv (sinnvollerweise die `content.xml`) extrahiert wird.

5.4.2. Das Modul xml.dom

Das Modul `xml.dom` [Pyt08c] wird bereitgestellt, um eine XML-Datei in das Document Object Model (DOM) zu überführen (Deserialisierung). Das DOM hat als interne Repräsentation eine baumartige Struktur, die mit den Methoden der Klasse sukzessive adressiert werden kann. Ein Teil des Moduls ist `minidom`, das eine leichtgewichtige Implementation des DOM und der Zugriffsmethoden enthält. Für die Beispielimplementation wird deshalb `minidom` verwendet. Mit der Methode `parse` wird aus einer Quelle das DOM eingelesen (siehe Listing 5.18). Die unterschiedliche Syntax beim Import ergibt sich daraus, dass bei der `from`-Variante der komplette Namensraum des Moduls in der eigenen Klasse zur Verfügung steht.

```
from xml.dom import minidom

# Öffnen einer XML-Datei
datei = open(content.xml)

# Einlesen des DOM-Baumes
xml = minidom.parse(file)
```

Listing 5.18: *Import aus dem Modul xml.dom und Aufruf einer Funktion*

Nun kann man zunächst den ersten Knoten ausgeben. Bei einem XML-basierten Office-Dokument ist das immer das Dokument (`document`) Anschließend kann man sich Kindknoten und Geschwisterknoten ausgeben lassen (Listing 5.19). Alternativ kann nach Elementen mit bestimmten Bezeichnern gesucht werden.

```
# Herausgeben der Liste der Kindknoten
kindknoten = xml.childNodes()

# Herausgeben des ersten Kindknotens
kind = xml.firstChild()

# Herausgeben eines Geschwisterknotens
geschwister = kind.nextSibling()

# Suche nach Knoten mit Elementnamen text:p
paragraphs = xml.getElementsByTagName('text:p')
```

Listing 5.19: *Ausgabe von Kindknoten, Geschwisterknoten und Suche nach Knoten*

5.4.3. Das Modul hashlib

Mit `hashlib` [Pyt08b] steht in Python eine Bibliothek für die Verwendung kryptographischer Hash-Funktionen zur Verfügung. Dazu gehören Algorithmen wie MD5 und die SHA-Familie. Diese Hash-Funktionen werden zur Implementation zum Erzeugen der Schlüssel, des Wasserzeichens und der Ermittlung der Rangfolge benötigt. Der Aufruf ist denkbar einfach, siehe Listing 5.20.

```
import hashlib

# Erzeugen eines SHA-1 Hashwertes
hash = hashlib.sha1(object)
```

Listing 5.20: *Import des Moduls hashlib und Aufruf einer Funktion*

5.5. Eigene Klassen

Im folgenden Abschnitt folgt eine Beschreibung der Signaturen der eigenen Klassen, die für die Umsetzung entwickelt wurden. Zur Veranschaulichung ist jede Klasse mit einem UML-Diagramm [Obj07] illustriert.

5.5.1. Die Klassen Key, MasterKey und DocumentKey

Die Klasse ***Key*** ist eine Oberklasse. Sie besitzt das Attribut *keyValue*. Außerdem stellt die Klasse eine Methode namens *generateKey* bereit (siehe Abbildung 5.4). Damit kann ein geheimer Schlüsselwert erzeugt werden, der in *keyValue* gespeichert wird. Die zweite Methode heißt *getValue* und gibt den aktuellen Wert von *keyValue* zurück. Diese Methode wird benötigt, weil *keyValue* ein privates Attribut ist, also nur von innerhalb der Klasse darauf zugegriffen werden kann. Darüber hinaus gibt es die Methode ***loadKey(Source)***. Damit kann ein Schlüssel aus einer Datei[18] geladen werden. Der Dateiname muss als Parameter übergeben werden. Schließlich kann ein Schlüssel mit der Methode *saveKey* in eine Datei gespeichert werden, die als Parameter *Source* übergeben werden muss. Die Klasse ***Key*** wird selber nie verwendet.

Von ***Key*** erbt die Klasse ***MasterKey***. ***MasterKey*** ist eine Spezialisierung für den Hauptschlüssel. Sie enthält alle Methoden der Oberklasse.

Schließlich erweitert auch ***DocumentKey*** die Klasse ***Key***. Sie stellt Dokumentschlüssel bereit. Anstelle zusätzlicher Methoden wird die Methode *generateKey* überladen. Denn ein Dokumentschlüssel wird mit dem Hauptschlüssel und einem Dokument als Eingabe erzeugt. Deshalb müssen beide als Parameter übergeben werden.

Die *__init__()*-Methoden werden bei Python aufgerufen, nachdem ein Objekt erzeugt wurde. Sie entsprechen nicht ganz einem Konstruktor, weil das Objekt zu dem Zeitpunkt bereits existiert. Ein weiterer Punkt zur Nomenklatur: Bei Python ist für gewöhnlich das erste Argument einer Klassenmethode die Klasse selber. Der Konvention nach wird dies durch das Schlüsselwort *self* gekennzeichnet. Weil er auch beim Aufruf der Klasse nicht übergeben werden muss, wird aus Gründen der Übersichtlichkeit in den UML-Diagrammen jedoch darauf verzichtet, diesen obligatorischen Parameter mit abzubilden.

5.5.2. Die Klasse Watermark

Die Klasse ***Watermark*** definiert das Objekt für Wasserzeichen. Der numerische Wert für das Wasserzeichen wird in *WMValue* gespeichert. *Length* gibt die Länge des Wasserzeichens an. Neben der obligatorischen *Init*-Methode gibt es die Methoden *generateWM*, *toBinary* und *getValue*. Mit *generateWM* wird ein Wasserzeichen erzeugt. Dabei muss ein ***DocumentKey*** und die gewünschte Länge in Bit als Parameter übergeben werden. Die private Methode *toBinary* wird nur intern von *getValue* benutzt,

18 Prinzipiell kann auch eine URI (Uniform Ressource Identifier) übergeben werden, so dass ein Master-Schlüssel aus dem Internet geladen werden kann. Aber das ist bei einem Schlüssel, der sicher verwahrt werden soll, nicht sinnvoll. Es gibt keine Mechanismen für eine verschlüsselte Verbindung.

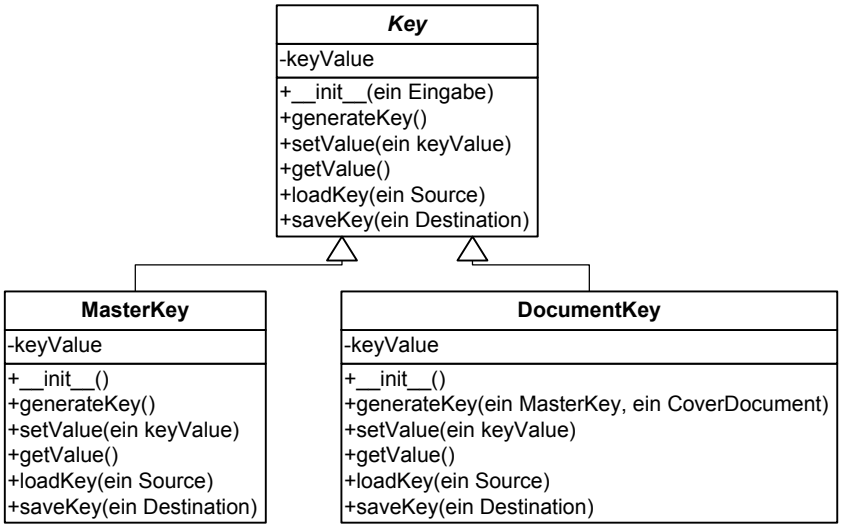
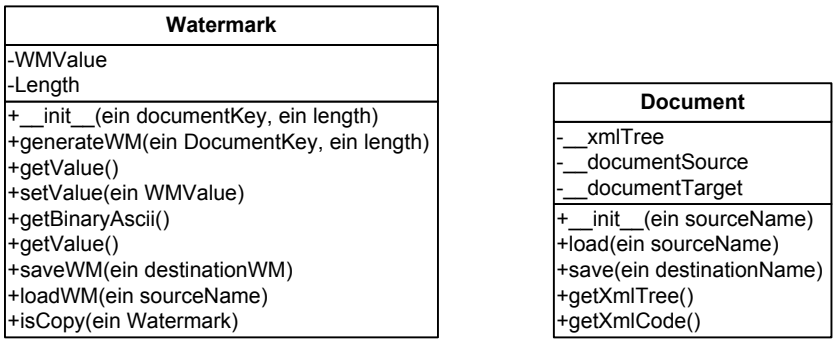


Abbildung 5.4.: *Die Klassen* MasterKey *und* DocumentKey *mit Oberklasse* Key

um das Wasserzeichen in einen Binärwert zu konvertieren. Letztgenannte Methode gibt den (privaten) Wert des Wasserzeichens als Binärwert aus.

Watermark braucht prinzipiell keine Methode zum Laden oder Speichern eines Wasserzeichens. Das Wasserzeichen kann immer wieder aus dem Dokumentschlüssel erzeugt werden. Allerdings sind die Funktionen zu Test- und Debuggingzwecken dennoch implementiert.



(a) Die Klasse *Watermark* (b) Die Klasse *Document*

Abbildung 5.5.: *Die Klassen* Watermark *und* Document *definieren weitere Objekte.*

Die Klasse ***Watermark*** hat keine gemeinsame Oberklasse mit der Klasse Key[19], weil Wasserzeichen und Schlüssel semantisch unterschiedlich sind.

5.5.3. Die Klasse Document

Um ein Office-Dokument zu laden, wird die Klasse ***Document*** aufgerufen. Ein Dokument wird über die Methode *load* geladen, der man einen Dateinamen übergeben muss (ein Office-Dokument). Das XML-Hauptdokument wird automatisch aus dem Container entpackt und deserialisiert. Der damit erzeugte DOM-Tree wird als Attribut *__xmlTree* vorgehalten.

Mit *save* wird ein Dokument gespeichert, wobei das XML-Hauptdokument zurück in den Container geschrieben wird.

5.5.4. Die Klasse Embedder

Die eigentliche Einbettung eines Wasserzeichens wird mittels der Klasse ***Embedder*** durchgeführt. Der Methode *findMarkers* wird ein Objekt vom Typ *Document* übergeben. Sie gibt eine Liste mit XML-Nodes zurück, die ein Bit des Wasserzeichens aufnehmen können. Diese Liste kann zusammen mit dem Wasserzeichen an die Methode *embed* übergeben werden. In dieser Methode steckt der Großteil der Anwendungslogik des oben beschriebenen Verfahrens (Abschnitt 5.1.2). Zunächst wird die Liste der Nodes sortiert.

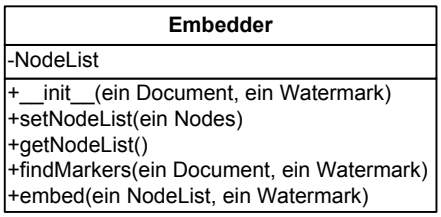


Abbildung 5.6.: *Die Klasse* Embedder

Da das *Document* als Objekt übergeben wird, werden die Änderungen direkt am geladenen Dokument ausgeführt. Anschließend muss nur die *save*-Methode auf dem *Document* aufgerufen werden.

19 Außer dass natürlich beide von *Object* erben, aber das ist ein Mechanismus der Sprache, keine Entwurfsentscheidung.

5.5.5. Die Klasse Verifier

Ein Großteil der Verarbeitungsschritte, um ein Waserzeichen wieder auszulesen und mit einem eingebetteten zu vergleichen, ähnelt denen bei der Einbettung verwendeten. Daher sind die Methoden der Klasse ***Verifier*** denen der Klasse ***Embedder*** auch recht ähnlich.

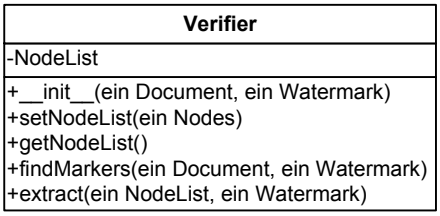


Abbildung 5.7.: *Die Klasse* Verifier

Vor allem unterscheidet sich die Klasse in der Methode *extract*, die eine NodeList und ein Watermark als Parameter übergeben bekommt. Hier werden die Bits des Wasserzeichens aus den Nodes gelesen und zurückgegeben. Der Vergleich der beiden Bit-Strings erfolgt hier nicht, das Wasserzeichen muss übergeben werden, damit die Sortierung der Nodes vorgenommen werden kann und die gleiche Reihenfolge wie bei der Einbettung ergibt.

5.5.6. Startskript und Parameter

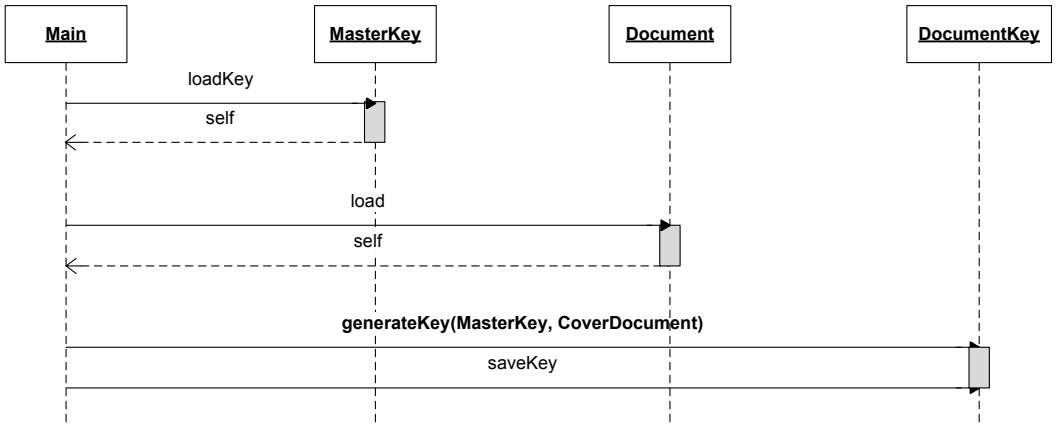


Abbildung 5.8.: *Erzeugen eines Dokumentschlüssels*

Das Python-Skript wird über die Kommandozeile aufgerufen. Der Parameter `-e` oder `-embed-watermark` gefolgt von einem Hauptschlüssel und Dokument führt zur Einbettung eines Wasserzeichens (Listing 5.21). Die *Main*-Methode des Startskripts legt dabei zunächst ein neues Objekt vom Typ ***Document*** an. Mit Aufruf der *load*-Methode von ***Document*** wird ein übergebener Dateiname als geöffnet. Anschließend wird ein neuer Dokumentschlüssel aus dem Dokument und dem Hauptschlüssel erzeugt. Dabei wird dieser aus der als Parameter übergebenen Datei geladen und als Objekt der Methode *generateKey* eines ***DocumentKey***-Objektes übergeben.

```
ihxod.py -e <Hauptschlüssel> <Dokument>
```

Listing 5.21: *Startskript für den Embedder*

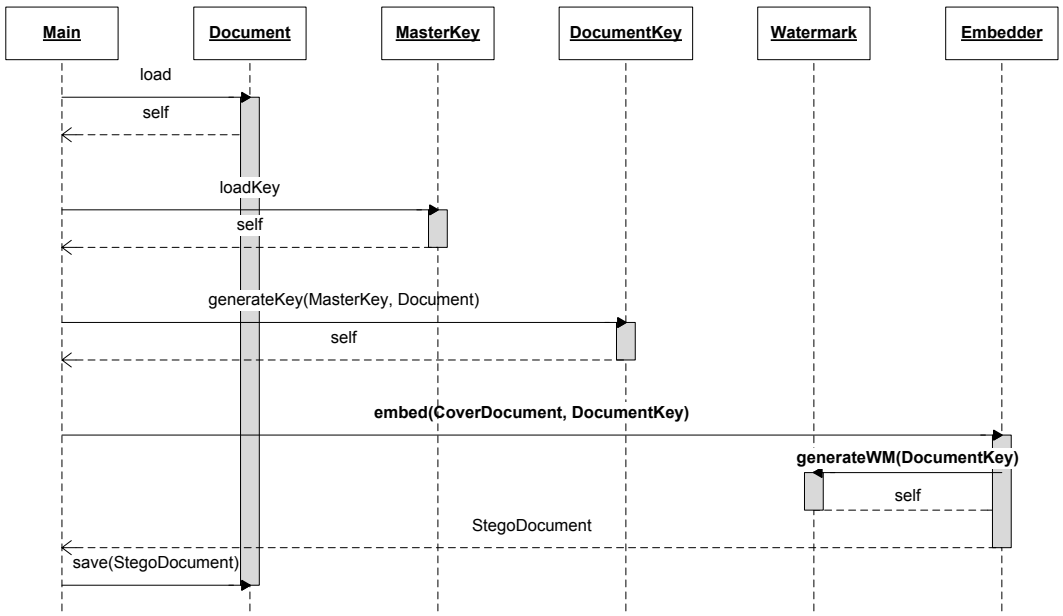


Abbildung 5.9.: *Aufruf des Embedders*

Mit diesem Dokumentschlüssel zusammen wird die Methode *embed* des Embedders aufgerufen. Der Embedder wiederum veranlasst, dass aus dem Dokumentschlüssel ein Wasserzeichen generiert wird. Erst jetzt erfolgt die eigentliche Einbettung. Schließlich löst die *Main*-Methode die *save*-Methode des ***Document***-Objekts aus, die das veränderte Dokument mit den Einbettungen speichert. Das Dokument wird dabei nur verändert, erhält also keinen neuen Dateinamen. Der Dokumentschlüssel wird unter gleichem Namen wie das Dokument mit der Endung `.key` im gleichen Verzeichnis gespeichert (siehe Abbildung 5.9).

Der Aufruf mit dem Parameter `-v` oder `-verify-watermark`, gefolgt von einem Dokumentschlüssel und einem Dokument, führt zur Verifikation. Die Ausgabe ist entweder `True` oder `False` (Listing 5.22).

```
ihxod.py -v <Dokumentschlüssel> <Dokument>
```

Listing 5.22: *Startskript für die Verifikation*

Nach einem Aufruf zur Verifikation ist die Abfolge der bei der Einbettung relativ ähnlich. Sie unterscheidet sich vor allem dadurch, dass die Methode *verify* des Verfiers, die wie der Embedder auch Objekte vom Typ ***Document*** und ***DocumentKey*** übergeben bekommt, kein Dokument zurück gibt, sondern nur einen Boolean-Wert der entweder *True* oder *False* ist. Damit dieser ermittelt werden kann, wird zunächst ein Wasserzeichen aus ***Document*** und ***DocumentKey*** erzeugt (mit der Methode *generateWM* in der Klasse Watermark). Anschließend wird das eingebettete Wasserzeichen ausgelesen. Nun wird das ausgelesene durch die Methode *isCopy* der Klasse ***Watermark*** mit dem Originalwasserzeichen verglichen. Der Rückgabewert, ebenfalls *True* oder *False*, wird zurückgegeben.

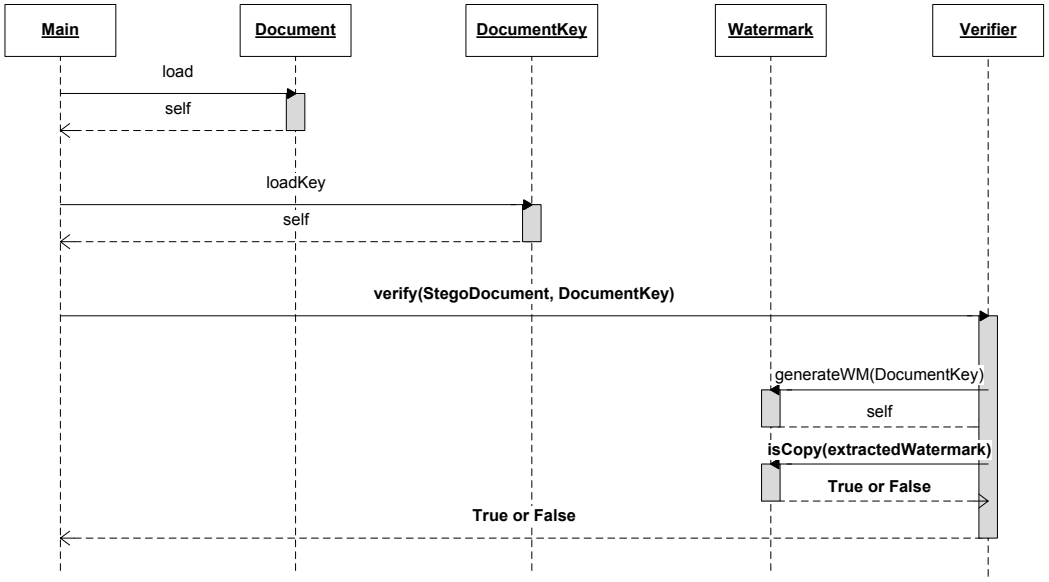


Abbildung 5.10.: *Aufruf des Verifiers*

Einen Hauptschlüssel erzeugt man mit dem Parameter `-k` oder `-generate-key`. Es müssen ein Name für den Schlüssel und ein geheimer Ausdruck angegeben werden. In diesem Prototyp wird daraus nur der Hashwert für den Schlüssel erzeugt (mit

der Methode *generateKey* der Klasse ***MasterKey***), die Datei ist aber nicht dadurch geschützt. Sie muss daher sicher verwahrt werden (Listing 5.23).

```
ihxod.py -k <Hauptschlüssel> "<Geheimnis>"
```

Listing 5.23: *Erzeugen eines Hauptschlüssels*

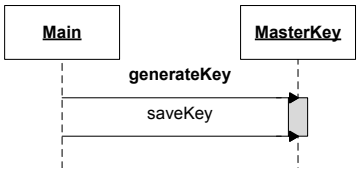


Abbildung 5.11.: *Erzeugen eines Hauptschlüssels*

Fall es beim Aufruf zu Eingabefehlern kommt, wird eine Bedienungsanleitung ausgegeben.

5.6. Erkenntnisse aus der eigenen Herangehensweise

Im diesem Kapitel wurde untersucht, ob ein linguistisches Watermarking-Schema von Wu und Stinson in eine andere Domäne übertragen werden kann. Um das zu erreichen, wurde vor allem das Konzept der Darstellungsrepräsentation von XML-Dokumenten beschrieben. Die Beschreibung der Darstellungsrepräsentation wurde im Laufe des Kapitels noch konkretisiert. Die Algorithmen für die Einbettung und Verifikation konnten entsprechend modifiziert werden.

Anschließend wurden Möglichkeiten zur technischen Informationseinbettung näher betrachtet. Office-Dokumente werden bei der Verarbeitung durch Office-Anwendungen intern nicht im XML-Format repräsentiert, sondern in einer transformierten Repräsentation, die der Spezifikation des Open Document Formats nicht entnommen werden kann. Viele darstellungsäquivalente Veränderungen an XML-Auszeichnungen führen nur zu flüchtigen Änderungen. Erschwerend kommt hinzu, dass zur Ermittlung einer darstellungsgleichen Variante einer Formatierungsanweisung die Bedeutung dieser Formatierungsanweisung interpretiert werden muss. Trotzdem war es möglich, darstellungsgleiche Varianten zu XML-Auszeichnungen zu finden, die diese internen Transformationen überstehen. Insbesondere verbleiben auf diese Weise eingebettete Informationen auch nach Konvertierung in ein

binäres Dateiformat im Dokument. Damit können die naheliegensten Angriffe auf verborgene Informationen abgewehrt werden.

Es ist jetzt also möglich, das vorgeschlagene Watermarking-Schema für XML-basierte Office-Dokumente im Rahmen eines Buyer-Seller-Protokolls einzusetzen. Im nächsten Kapitel werden die Eigenschaften dieses Vorschlags mit den Desiderata verglichen, um herauszufinden, ob das Verfahren die Anforderungen erfüllt.

6. Zusammenfassung und Ausblick

6.1. Zusammenfassung

Die vorliegende Arbeit befasst sich mit den Problemen, die beim Austausch digitaler Dokumente in einem Online-Marktplatz entstehen können (vgl. Kapitel 1). Ziel war es, Möglichkeiten des Information Hiding in XML-basierten Office-Dokumenten zu untersuchen. Es wurde festgestellt, dass kein Lösungsansatz für Information Hiding von XML-basierten Office-Dokumenten existierte. Zudem wurde dargelegt, welche offenen Fragen in diesem Zusammenhang existieren und weshalb eine nähere Analyse sinnvoll ist.

Nachdem in Kapitel 2 grundlegende Definitionen und Begriffe erläutert wurden, stellt Kapitel 3 ein konkretes Anwendungsszenario dar, das die beteiligten Rollen und Vorgänge verdeutlichen und die Anforderungen an ein Verfahren für XML-basiertes Information Hiding aufzeigen soll. Dieses zwar fiktive, gleichwohl aber realistische Szenario zeigt deutlich, dass es sinnvoll ist, technische Lösungen zu entwickeln, die den Missbrauch bzw. die Manipulation von Office-Dokumenten erkennbar machen oder zumindest den Aufwand eines solchen Eingriffs erhöhen.

Ausgehend von dem entwickelten Szenario wurden in Kapitel 4 zunächst die Anforderungen dargestellt, die entsprechende technische Lösungen innerhalb des Systems erfüllen müssen. Diskutiert wurde, inwieweit Verfahren wie Digital Rights Management oder Digitale Signaturen diese Anforderungen erfüllen können. Im zweiten Teil des Kapitels wurden anschließend bereits bestehende Verfahren des Information Hiding untersucht und daraufhin überprüft, ob sie auf XML-basierte Office-Dokumente anwendbar sind. Das Ergebnis: Es existieren keine Methoden, die unmittelbar für Office-Dokumente eingesetzt werden können, um diese auf mögliches Information Hiding hin zu überprüfen. In Abschnitt 6.1.1 erfolgt eine detailliertere Zusammenfassung dieser Analyse.

Im ersten Teil von Kapitel 5 wurde ein linguistisches Watermarking-Schema von Wu und Stinson an die Erfordernisse von XML-Dokumenten angepasst. Im zweiten Teil des Kapitels erfolgte dessen technische Umsetzung. Zum einen wurden dabei Vorschläge für konkrete Methoden der technischen Einbettung beschrieben; dabei handelt es sich um die Übertragung von Absatzformatierungen auf Textformatierungen. Zum anderen wurden die Komponenten statisch mit Klassendiagrammen dargestellt.

Ebenfalls wurde das dynamische Systemverhalten des Entwurfs unter Zuhilfenahme von UML-Sequenzdiagrammen erläutert.

6.1.1. Ergebnisse der Analyse

Vor der Diskussion existierender Methoden wurde zunächst eine kategorische Einteilung der Verfahren anhand der Cover-Medien vorgenommen. Die Kategorien waren Methoden für gerenderter Text (Abschnitt 4.3.1), linguistische Methoden (Abschnitt 4.3.2), XML (Abschnitt 4.3.3), nicht-textuelle Dokumentteile (Abschnitt 4.3.4), ZIP-Archive (Abschnitt 4.3.5) und sonstige Methoden (Abschnitt 4.3.6). Methoden für gerenderten Text sind im wesentlichen für XML-basierte Dokumente nicht geeignet oder können das eigentliche Potential – die Bandbreite, die durch die Kodierung im XML-Format entsteht – nicht nutzen.

Die vergleichende Darstellung der linguistischen Methoden hat zunächst Gemeinsamkeiten mit XML-Information Hiding gezeigt, vor allem die geringe Bandbreite und eine niedrige Anzahl der möglichen Transformationen. Es wurde festgehalten, dass es grundsätzlich möglich ist, linguistische Methoden auf den Text von Office-Dokumenten anzuwenden. Der konkrete Fall zeigt jedoch auch, dass es mithilfe der diskutierten Methoden nicht möglich ist, die Anforderungen an Korrektheit und Ausdruck der Sprache durch die diskutierten Methoden zu erfüllen. Aufgrund der Gemeinsamkeiten zeigte sich allerdings, dass Teile übernommen werden können, wie ein Einbettungsschema mit Anpassungen. Damit können die Vorteile des Schemas (im konkreten Fall das Verfahren von Wu und Stinson [WS08]) genutzt werden: Probleme wie Synchronisation und Robustheit sind damit bereits zu einem wesentlichen Teil gelöst.

Die bisher existierenden Methoden für Information Hiding in XML wurden für XML-Datenbanken entwickelt. Die Einsatzbedingungen sind nicht identisch mit denen für XML-Office-Dokumente. In den bislang vorliegenden Veröffentlichungen befassen sich die Autoren vorwiegend mit der Markierung der Daten und nicht der XML-Struktur. Damit sind die existierenden Verfahren nicht für die Anwendung im beschriebenen Szenario geeignet.

Damit ein Wasserzeichen auch Teile eines Dokuments einschließt, die nicht aus Text bestehen, können bestehende Verfahren für Bilder oder Videos benutzt werden. Prinzipiell könnte zwar auch der ZIP-Container des Dokuments mit Markierungen versehen werden. Da der Container aber beim Speichern komplett neu erstellt wird, wären diese Markierungen dann nicht ausreichend robust. Aus dem gleichen Grund ist auch das Verfahren von Inoue für XML-basierte Office-Dokumente nur zur flüchtigen Einbettung zu gebrauchen. Als Schlussfolgerung wurde vorgeschlagen, für künf-

tige Anwendungen flüchtige Wasserzeichen als Schutz vor Manipulationen in XML-Dokumenten zu benutzen.

6.1.2. Bewertung des Entwurfs

Der vorliegende Prototyp zeigt für ausgewählte Methoden, dass das entwickelte Verfahren eingesetzt werden kann. Es ist nun einfach, auf dem bestehenden Entwurf aufbauend weitere Transformationen in die Komponente des Embedders (und des Verifiers) zu integrieren. Die übrigen Komponenten des Prototyps brauchen dabei nicht verändert zu werden. Dann können in weiteren XML-Elementen des Open Document Format Bits eingebettet werden. So kann die Länge der in einem Dokument eingebetteten Wasserzeichen noch deutlich gesteigert werden. In praktischen Anwendungen des Verfahrens sollte die Anzahl der markierbaren Teilbäume die Länge des Wasserzeichens (in Bit) übersteigen. Je mehr ungenutzte Stellen im Dokument zusätzlich vorhanden sind, desto schwieriger ist ein Angriff auf das Verfahren.

Das eigene Verfahren wird anhand der in Abschnitt 4.2 eingeführten Desiderata bewertet.

Eindeutigkeit: Es ist möglich, den Ersteller eines Dokuments eindeutig zu bestimmen. Wenn der Master-Schlüssel nicht kompromittiert wird, kann nur der Ersteller der Erzeuger des gültigen Wasserzeichens sein.

Nicht-Abstreitbarkeit: Diese Anforderung muss durch das Protokoll sichergestellt werden. In Abschnitt 5.1.3 wurde beschrieben, wie das Schema angepasst werden muss, um die Funktion der Nicht-Abstreitbarkeit zu erreichen.

Fähigkeit, Manipulationen zu erkennen: Durch den Editierabstand zwischen dem Original-Wasserzeichen und dem eingebetteten Wasserzeichen kann gemessen werden, wie viele Bits des Wasserzeichens verändert wurden. Damit werden nur Angriffe bemerkt, die zumindest einige Bits des Wasserzeichens erfolgreich verändern. Um generell Angriffe zu erkennen, muss zusätzlich eine der zuvor beschriebenen Methoden für flüchtige Wasserzeichen in die Lösung integriert werden. Es konnte gezeigt werden, dass dies leicht umzusetzen ist.

Unversehrtheit der Informationen: Die Informationen in den Dokumenten bleiben nach Einbettung des Wasserzeichens erhalten. Der Text wird, im Gegensatz zu linguistischen Methoden für Information Hiding, nicht verändert. Die gerenderte Darstellungsweise wird ebenfalls nicht beeinträchtigt.

Ebenso werden die Anforderungen erfüllt, die sich aus der Umsetzung ergeben.

Nicht-Wahrnehmbarkeit: Dokumente sind nach der Einbettung eines Wasserzeichens darstellungsgleich mit dem Originaldokument. Die Datei wird natürlich verändert. Der Prototyp erschwert den Vergleich des Cover-Dokuments mit dem Stego-Dokument, indem die Ausgabe die Originaldatei überschreibt.

Verifizierbarkeit: Der Dokumentschlüssel kann jederzeit vom Ersteller produziert werden, ohne dabei den Master-Schlüssel zu kompromittieren. Dadurch kann sich prinzipiell jeder Teilnehmer des Systems, ganz gleich ob Betreiber oder Mitbewerber, von der Gültigkeit der Dokumente überzeugen.

Robustheit: Das Schema erreicht Robustheit durch die Sortierung der Teilbäume anhand des Dokumentschlüssels und die Berechnung des Editierabstands beim Vergleichen des Wasserzeichens.

Kollisionsfreiheit: Für die Schlüssel werden gut untersuchte und dokumentierte Hash-Funktionen verwendet, für die diese Eigenschaft hinreichend belegt ist. Für das Wasserzeichen gilt, dass eine ausreichende Länge des Wasserzeichens notwendig ist, um Kollisionsfreiheit zu erfüllen. Das ist abhängig von der Anzahl der markierbaren Elemente im Dokument. Vorschläge zur Steigerung dieses Wertes wurden zuvor diskutiert.

Ressourcenschonend: Durch die Einbettung von Wasserzeichen in jede Nachricht im System werden zusätzliche Rechenschritte erforderlich. Der entstehende rechnerische Overhead ist bei dem erzielten Mehrwert jedoch vertretbar. Der Mehraufwand, der durch den Präprozessor entsteht, kann etwa mit dem Zeitaufwand verglichen werden, den ein Virenscanner benötigt, um die Dokumente vor dem Öffnen zu prüfen.

Offenheit: Alle verwendeten Verfahren können beschrieben und die Dokumentationen veröffentlicht werden. Trotzdem wird das Verfahren nicht gefährdet, da die Sicherheit von den Schlüsseln (vor allem dem Master-Schlüssel) und nicht dem Verfahren abhängt. Kerkhoffs' Prinzip ist damit gewährleistet.

Ende-Zu-Ende-Prinzip: Das System stellt die Bedingung, dass alle im Umlauf befindlichen Dokumente mitWasserzeichen versehen werden müsssen. Die Nachrichten unterliegen damit über den gesamten Weg vom Sender zum Empfänger dem Schutz des Wasserzeichens.

Synchronisation: Das Problem der Synchronisation ist durch das vorliegende Schema gelöst. Durch die Sortierung von Teilbäumen anhand des Dokumentschlüssels ergibt sich die gleiche Sortierfolge auch dann, wenn die Teilbäume innerhalb des Dokuments umgestellt werden. Da für die Einbettung Teilbäume gewählt werden, die innerhalb eines Textdokuments generell als Einheit betrachtet werden (wie etwa Absätze), führen Änderungen ihrer Reihenfolge nicht automatisch zur Zerstörung des Wasserzeichens.

Mit dem gezeigten Verfahren können nunmehr die Bedrohungen aus dem fiktiven Szenario erfolgreich durch ein Buyer-Seller-Protokoll abgewendet werden.

Kopie von Dokumenten durch einen Reseller: Durch die Einbettung digitaler Wasserzeichen in XML-basierte Office-Dokumente kann der Ersteller eines Dokuments bestimmt werden. Es kann nicht verhindert werden, dass Dokumente ausgedruckt und neu eingescannt werden. Hierfür stellen die Anwendungen andere Mechanismen bereit.

Abstreiten des Angebots: Dies wird vor allem durch das gewählte Protokoll verhindert. Allerdings steht das dafür benötigte Wasserzeichen für XML-basierte Office-Dokumente erst durch diese Arbeit zur Verfügung.

Abstreiten der Ausschreibung: Das gleiche gilt in diesem Fall.

Fälschen eines Angebots: Weil Nicht-Abstreitbarkeit erreicht werden kann, ist es umgekehrt auch nicht möglich, Angebotsunterlagen im Namen eines anderen herauszugeben.

Fälschen einer Ausschreibung: Auch Dokumente einer Ausschreibung können nicht gefälscht werden, ohne dass der Master-Schlüssel des Opfers erraten (was rechnerisch nicht effizient zu lösen ist) oder gestohlen wird, was bei guter Verwahrung physikalischen Zugriff auf den Master-Schlüssel voraussetzt.

Teilweise werden die Bedrohungen durch ein ordentliches Buyer-Seller-Protokoll (wie das von Memon) abgewehrt. Allerdings werden für ein solches Protokoll Wasserzeichen benötigt, die vor der beschriebenen Methode für XML-basierte Office-Dokumente noch nicht existierten.

Eine Umsetzung lässt sich auch für Office Open XML erzielen. Zwar sind die Formatierungstags beider Standards unterschiedlich – für die wesentlichen Formatierungen stellen aber beide Standards entsprechende Auszeichnungen bereit, so dass sich die Funktionalität analog erzielen lässt.

Darüber hinaus ist das Verfahren grundsätzlich inhaltsunabhängig. Das bedeutet, dass es nicht an eine Sprache oder den Wortschatz einer Anwendungsdomäne angepasst werden muss, wie es bei linguistischem Information Hiding der Fall ist.

Die Umsetzung eines vollständigen Verfahrens für ODF erfordert einen kompletten Parser über den kompletten Namensraum, der für jedes Zeichen die Darstellungsaspekte abbildet. Nur so kann eine exakte darstellungsgleiche Variante erzeugt werden. Aufgrund des modularen Charakters des Prototyps können diese Komponenten sowohl in bestehende Anwendungen integriert werden als auch als Postprozessor für einen beliebigen Editor eingesetzt werden.

6.1.3. Beitrag dieser Arbeit

Im Rahmen dieser Arbeit wurde begrifflich festgelegt, was Information Hiding in XML-basierten Office-Dokumenten ist und wie es sich von anderen Domänen im Information Hiding unterscheidet. Es wurde auf den Unterschied von Informationen und Daten hingewiesen sowie die Darstellungsgleichheit für XML-basierte Office-Dokumente definiert. Existierende Verfahren wurden kategorisiert. Dabei wurde die Wirkung der Verfahren auf die unterschiedlichen Ebenen des Dokuments berücksichtigt. Existierende Arbeiten wurden analysiert und im Hinblick auf ihre Eignung für XML-basierte Office-Dokumente bewertet.

Das Verfahren von Wu und Stinson wurde als geeignet identifiziert. Das Schema kann nicht direkt verwendet werden, weil es sich um eine völlig unterschiedliche Anwendungsdomäne handelt. Teile des Verfahrens wurden durch eigene ersetzt, um es auf XML-Strukturen anzupassen. Unter anderem wurde eine Tie-Break-Regel bei der dokumentschlüsselabhängigen Sortierfunktion definiert. Zudem wurde eine Analogie zu einer Bedeutungsrepräsentation von Text gebildet. Es konnte nicht auf die bestehenden technischen Einbettungsmethoden für XML zurückgegriffen werden. Diese haben aufgrund der komplexen Struktur der Dokumentformate und des Umstandes, dass es sich um ein reines Speicherformat handelt, in das die Dokumente ausschließlich zu diesem Zweck transformiert werden, Probleme eine robuste Einbettung zu gewährleisten. Es wurde eine neue, zusätzliche Methode gezeigt, um Wasserzeichen in XML einzubetten. Diese Methode basiert etwa auf der Transformation von Absatzformatierungen zu Textformatierungen. Sie ist ausreichend robust, um im Dokument auch nach der Verarbeitung durch die Anwendung erhalten zu bleiben. Zudem ist das Verfahren somit unabhängig vom Dokumentinhalt.

Mittels einer Proof-of-Concept-Implementation wurde der Beweis erbracht, dass das System an sich funktionsfähig ist und implementiert werden kann. Es wurde ein vereinfachtes Modell von Formatierungsauszeichnungen entwickelt, um eine prototypische Umsetzung im Rahmen dieser Arbeit zu ermöglichen.

6.2. Ausblick

Die Arbeiten an einem Verfahren für Information Hiding in XML-basierten Office-Dokumenten haben nicht nur offene Fragen gelöst, sondern zugleich auch neue aufgeworfen. Im Folgenden wird zu zeigen versucht, wie die vorgenommenen Untersuchungen fortgeführt werden können.

Zunächst ist es offensichtlich, dass das in dieser Arbeit entwickelte Verfahren an andere Dokumenttypen (etwa Präsentationen oder Tabellen) angepasst werden kann.

Auch in diesen Dokumenten werden Styles zur Formatierung eingesetzt. Allerdings unterscheiden sich die Formatierungen von denen eines Textdokuments und müssen angepasst werden. Dadurch entstehen aber auch zusätzliche Möglichkeiten, wie etwa die umfangreichen leeren Tabellenfelder oder bedingte Formatierungen bei Tabellenkalkulationen. Grundsätzlich sind XML-basierte Office-Dokumente komplexe Formate. Die Anwendungen sind zwar bei der Darstellung innerhalb der durch XML definierten Rahmenbedingungen tolerant gegenüber XML-Auszeichnungen, die von den Anwendungen selber nicht produziert werden, erzeugen diese jedoch nicht. Dadurch kann die Bandbreite, die durch diese Varianz in XML entsteht, in Office-Dokumenten nicht für robuste Einbettungen genutzt werden. Anders sieht es jedoch aus, wenn der Ansatz generalisiert werden soll. Reine XML-Dokumente lassen sehr wohl weitere Einbettungen zu, als in dieser Arbeit für XML-basierte Office-Dokumente beschrieben wurden. Einige davon wurden im Kapitel 4 bereits beschrieben. Zu den XML-Formaten, die dadurch mit Wasserzeichen versehen werden können, gehören mit einigen Anpassungen dann XHTML und mit XML verwandte Auszeichnungssprachen wie HTML. Mit HTML oder XML als Auszeichnungssprache ließe sich ein Online-Marktplatz – wie der im Szenario beschriebene – ebenfalls implementieren. Aber für Wasserzeichen in HTML-Dokumenten gibt es auch weiterreichende Anwendungsfelder, etwa die Einbettung in beliebigen Webseiten, um die Struktur der Seite und das Layout anstelle des Inhalts zu schützen. Damit lassen sich die Strukturen von Webseiten oder Teile von Webseiten identifizieren. Zeichnungen im XML-basierten Vektorgraphikformat Scalable Vector Graphics (SVG) können ebenfalls durch Wasserzeichen in XML geschützt werden.

Wie in der Einleitung formuliert, überschreitet der Wert von Informationen in einigen Bereichen den der Waren. Diese Arbeit hat einen Betrag dazu geleistet, diesen Wert für ein Format, in dem solche Informationen gespeichert werden, zu bewahren. In Zukunft sollten weitere Formate untersucht werden, um einen sicheren Online-Handel für beliebige Dokumente und Medien zu ermöglichen.

A. XML-Auszüge

A.1. XML-Code zu Karawane.odt im Original

```
<?xml version="1.0" encoding="UTF-8"?>
<office:document-content xmlns:office="
   urn:oasis:names:tc:opendocument:xmlns:office:1.0" [...]
   office:version="1.1">
  [...]
      <text:h text:style-name="Heading_20_1" text:outline-
         level="1">
        <text:span text:style-name="T1">KARAWANE</text:span>
      </text:h>
      <text:p text:style-name="P7">
        <text:span text:style-name="T2">jolifanto bambla ô
           falli bambla</text:span>
      </text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T4">grossiga m´pfa habla
           horem</text:span>
      </text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T7">égiga goramen</
           text:span>
      </text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T2">higo bloiko russula
           huju</text:span>
      </text:p>
      <text:p text:style-name="P1">
        <text:span text:style-name="T2">hollaka hollala</
           text:span>
      </text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T6">anlogo bung</
           text:span>
      </text:p>
```

```
      <text:p text:style-name="P7">
        <text:span text:style-name="T2">blago bung</text:span
           >
      </text:p>
      <text:p text:style-name="P7">blago bung</text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T2">bosso fataka</
           text:span>
      </text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T2">ü üü ü</text:span>
      </text:p>
      <text:p text:style-name="P3">
        <text:span text:style-name="T2">schampa wulla wussa
           ólobo</text:span>
      </text:p>
      <text:p text:style-name="P2">
        <text:span text:style-name="T2">hej tatta gôrem</
           text:span>
      </text:p>
      <text:p text:style-name="P4">eschige zunbada</text:p>
      <text:p text:style-name="P6">
        <text:span text:style-name="T2">wulubu ssubudu uluw
           ssubudu</text:span>
      </text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T2">tumba ba- umf</
           text:span>
      </text:p>
      <text:p text:style-name="P8">kusagauma</text:p>
      <text:p text:style-name="P5">
        <text:span text:style-name="T2">ba - umf</text:span>
      </text:p>
    </office:text>
  </office:body>
</office:document-content>
```

A.2. XML-Code zu Karawane.odt nach der Einbettung

```
<?xml version="1.0" encoding="UTF-8"?>
<office:document-content xmlns:office="
   urn:oasis:names:tc:opendocument:xmlns:office:1.0" [...]
   office:version="1.1">
  [...]
      <text:h text:style-name="Heading_20_1" text:outline-
         level="1">
        <text:span text:style-name="T1">KARAWANE</text:span>
      </text:h>
      <text:p text:style-name="P7">
        <text:span text:style-name="T2">jolifanto bambla ô
           falli bambla</text:span>
      </text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T4">grossiga m´pfa habla
           horem</text:span>
      </text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T7">égiga goramen</
           text:span>
      </text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T2">higo bloiko russula
           huju</text:span>
      </text:p>
      <text:p text:style-name="P1">
        <text:span text:style-name="T2">hollaka</text:span> <
           text:span text:style-name="T2">hollala</text:span>
      </text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T6">anlogo bung</
           text:span>
      </text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T2">blago bung</text:span
           >
      </text:p>
      <text:p text:style-name="P7">blago bung</text:p>
      <text:p text:style-name="P7">
```

```
        <text:span text:style-name="T2">bosso</text:span> <
           text:span text:style-name="T2">fataka</text:span>
      </text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T2">ü üü ü</text:span>
      </text:p>
      <text:p text:style-name="P3">
        <text:span text:style-name="T2">schampa wulla wussa
           ólobo</text:span>
      </text:p>
      <text:p text:style-name="P2">
        <text:span text:style-name="T2">hej tatta gôrem</
           text:span>
      </text:p>
      <text:p text:style-name="P4">eschige zunbada</text:p>
      <text:p text:style-name="P6">
        <text:span text:style-name="T2">wulubu ssubudu uluw</
           text:span> <text:span text:style-name="T2">ssubudu
           </text:span>
      </text:p>
      <text:p text:style-name="P7">
        <text:span text:style-name="T2">tumba ba- umf</
           text:span>
      </text:p>
      <text:p text:style-name="P8">kusagauma</text:p>
      <text:p text:style-name="P5">
        <text:span text:style-name="T2">ba - umf</text:span>
      </text:p>
    </office:text>
  </office:body>
</office:document-content>
```

Abkürzungsverzeichnis

$I(s)$	Information (Variante) von s
K	(Haupt-) Schlüssel
K_D	Dokumentschlüssel
$M(s)$	Bedeutung (Meaning)
W	Wasserzeichen
APS	Authorship Proof Scheme
CDATA	XML Character Data
CSS	Cascading Style Sheet
DCT	Discrete Cosinus Transformation
DOC	Mircosoft Word Dokument (Binärformat)
DOCX	Microsoft Word Office Open XML Dokument
DOM	Document Object Model
DRM	Digital Rights Management
DSSSL	Document Style Semantics and Specification Language
EXIF	Exchangeable Image File Format
GML	Generalized Markup Language
IPTC	International Press Telecommunications Council
JAR	Java Archive
MT	Machine Translation
NLP	Natural Language Processing
NLP	Natural Language Processing
NLS	Natural Language Steganography
NLW	Natural Language Watermarking
OCR	Optical Character Recognition
ODF	Open Document Format for Office Applications
ODT	Open Document Text
OOXML	Office Open XML
PDF	Portable Document Format
PKI	Public Key Infrastructure
SGML	Standard Generalized Markup Language

SHA ..... Secure Hash Algorithm 66
SVG ..... Scalable Vector Graphics 109
TMR ..... Text-Meaning-Representation 10
TPP ..... Trusted Third Party 92
URI ..... Uniform Ressource Identifier 95
WYSIWYG ..... What You See Is What You Get 10
WYSIWYS ..... What You See Is What You Sign 41
XML ..... eXtensible Markup Language 22
XML-C14N ..... Canonical XML 40
XML-DSig ..... XML-Digital Signature 40
XPath ..... XML Path Language 23
XSL ..... Extensible Stylesheet Language 23
XSL-FO ..... XSL Formatting Objects 23
XSLT ..... Extensible Stylesheet Language Transformations 23

Literaturverzeichnis

[AA04] Adnan M. Alattar and Osama M. Alattar. Watermarking electronic text documents containing justified paragraphs and irregular line spacing. In Edward J. Delp III. and Ping Wah Wong, editors, *Security, Steganography, and Watermarking of Multimedia Contents VI. Proceedings of the SPIE*, volume 5306 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 685–695, Jun. 2004.

[AHK03] Rakesh Agrawal, Peter J. Haas, and Jerry Kiernan. Watermarking relational data: framework, algorithms and analysis. *The VLDB Journal*, 12(2):157–169, Jul. 2003.

[AKS03] André Adelsbach, Stefan Katzenbeisser, and Ahmad-Reza Sadeghi. Watermark detection with zero-knowledge disclosure. *Multimedia Syst.*, 9(3):266–278, 2003.

[AL03] Mikhail J. Atallah and Stefano Lonardi. Authentication of lz-77 compressed data. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 282–287, New York, NY, USA, 2003. ACM Press.

[Alw08] Alwin Höfert Dichtungstechnik. Webseite. `http://www.hoefert.de/`, Jan. 2008. [Online; besucht 27. Mai 2008].

[AP98] R.J. Anderson and F.A.P. Petitcolas. On the limits of steganography. *Selected Areas in Communications, IEEE Journal on*, 16(4):474–481, 1998.

[AR01] Mikhail J. Atallah and Victor Raskin. Natural language watermarking: Design, analysis, and a proof-of-concept implementation. In *Proc. 4th International Information Hiding Workshop*, Apr. 2001. CERIAS TR 2001-13 Accepted for publication in Lecture Notes in Computer Science, Springer Verlag, Proc. 4th International Information Hiding Workshop, Pittsburgh, Pennsylvania.

[ARH+02] M. Atallah, V. Raskin, C. Hempelmann, M. Karahan, R. Sion, and K. Triezenberg. Natural language watermarking and tamperproofing. In *Lecture Notes in Computer Sciences*, volume 2578/2003, pages 196–212, 2002.

[ASP01] Mikhail Atallah, Radu Sion, and Sunil Prabhakar. Watermarking non-media content. In the the CERIAS Security Symposium, 2001.

[Bal17] Hugo Ball. Karawane. `http://de.wikisource.org/w/index.php?title=Karawane&oldid=325904`, 1917. [Online; besucht 06. August 2008].

[BeHL07] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. Cascading style sheets level 2 revision 1 (css 2.1) specification. `http://www.w3.org/TR/CSS21/`, Jul. 2007.

[Ber04] Richard Bergmair. Towards linguistic staganography: A systematic investigation of approaches, systems and issues. `http://richard.bergmair.eu/pub/towlingsteg-rep-inoff-a4.pdf`, Apr. 2004.

[Ber06] Anders Berglund. Extensible stylesheet language (xsl) version 1.1. `http://www.w3.org/TR/xsl/`, Dec. 2006.

[BK07] Arati Baliga and Joe Kilian. On covert collaboration. In *MM&Sec '07: Proceedings of the 9th workshop on Multimedia & security*, pages 25–34, New York, NY, USA, 2007. ACM Press.

[Bor01] Reinhard Bork. *Allgemeiner Teil des Bürgerlichen Gesetzbuches*. J. C. B. Mohr (Paul Siebeck), 2001.

[Boy00] John Boyer. Canonical xml. `http://www.w3.org/TR/xml-c14n`, Oct. 2000. [Online; besucht 26. November 2007].

[BPSM+06] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan. Extensible markup language (xml) 1.1 (second edition). `http://www.w3.org/TR/2006/REC-xml11-20060816/`, Aug. 2006.

[Bry91] Bill Bryson. *Mother Tongue: The English Language*. Penguin Books Ltd, 1991.

[CCHD05] Gang Chen, Ke Chen, Tianlei Hu, and Jinxiang Dong. *Watermarking Abstract Tree-Structured Data*, pages 221–232. Springer Berlin / Heidelberg, 2005.

[CCI91] CCITT. Recommandation x.800: Security architecture for open systems interconnection for ccitt applications, 1991.

[CD97] Mark Chapman and George I. Davida. Hiding the hidden: A software system for concealing ciphertext as innocuous text. In *ICICS '97: Proceedings of the First International Conference on Information and*

Communication Security, pages 335–345, London, UK, 1997. Springer-Verlag.

[CD99] James Clark and Steve DeRose. Xml path language (xpath) version 1.0. `http://www.w3.org/TR/xpath`, Nov. 1999.

[CD04] Gary Cantrell and David D. Dampier. Experiments in hiding data inside the file structure of common office documents: a stegonography application. In *ISICT '04: Proceedings of the 2004 international symposium on Information and communication technologies*, pages 146–151. Trinity College Dublin, 2004.

[CDF06] Ingemar Cox, Gwenaël Doërr, and Teddy Furon. *Watermarking Is Not Cryptography*, pages 1–15. Springer Berlin / Heidelberg, 2006.

[CK01] Scott Craver and Stefan Katzenbeisser. Security analysis of public-key watermarking schemes. In *Proceedings of the SPIE, Mathematics of Data/Image Coding, Compression, and Encryption IV*, volume 4475, pages 172–182, Jul. 2001.

[Cla99] James Clark. Xsl transformations (xslt) version 1.0. `http://www.w3.org/TR/xslt`, Nov. 1999.

[CM01] Ingemar J. Cox and Matt L. Miller. Electronic watermarking: The first 50 years. In *IEEE Fourth Workshop on Multimedia Signal Processing*, pages 225–230, 2001.

[Con07] The Unicode Consortium. *The Unicode Standard, Version 5.0.0*. Addison-Wesley, Boston, MA, 2007.

[CSP03] Jae-Gwi Choi, Kouichi Sakurai, and Ji-Hwan Park. Does it need trusted third party? design of buyer-seller watermarking protocol without trusted third party. *Applied Cryptography and Network Security*, pages 265–279, 2003.

[CT98] Christian Collberg and Clark Thomborson. The limits of software watermarking, Aug. 1998.

[DOD85] DOD. *Trusted Computer System Evaluation Criteria (TCSEC)*. Department of Defense, 1985. (Orange Book).

[DRM] DRM Info. Digital Restriction Management. `http://drm.info/de`. [Online; besucht 18. Juni 2008].

[ECM06] ECMA. *Standard ECMA-376 Office Open XML File Formats*. ECMA International, Dec. 2006. [Online; besucht 11. Oktober 2007].

[Ehr06] Erika Ehrli. Walkthrough: Word 2007 XML Format. `msdn2.microsoft.com/en-us/library/ms771890.aspx`, Jun. 2006.

[Eis07] J. Eisenberg. *OpenOffice.org XML Essentials*. O'Reilly & Asscociates, 2007. Work in progress.

[ERS02] Donald Eastlake, Joseph Reagle, and David Solo. Xml-signature syntax and processing. `http://www.w3.org/TR/xmldsig-core/`, Feb. 2002. W3C-Recommandation.

[ESG00] J. Eggers, J. Su, and B. Girod. Public key watermarking by eigenvectors of linear transforms. In *Proceedings of the European Signal Processing Conference*, Apr. 2000.

[FD03] Teddy Furon and Pierre Duhamel. An asymmetric watermarking method. *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on]*, 51(4):981–995, Apr. 2003.

[FG00] Jiri Fridrich and Miroslav Goljan. Robust hash functions for digital watermarking. In *ITCC '00: Proceedings of the The International Conference on Information Technology: Coding and Computing (ITCC'00)*, pages 178–183, Washington, DC, USA, 2000. IEEE Computer Society.

[GGA+05] Christian Grothoff, Krista Grothoff, Ludmila Alkhutova, Ryan Stutsman, and Mikhail Atallah. Translation-based steganography. In *Information Hiding*, pages 219–233, 2005.

[Gut04] Peter Gutman. Why XML Security is Broken. `www.cs.auckland.ac.nz/~pgut001/pubs/xmlsec.txt`, Oct. 2004. [Online; besucht 9. Dezember 2007].

[GWZL06] Fei Guo, Jianmin Wang, Zhihao Zhang, and Deyi Li. *A New Scheme to Fingerprint XML Data*. Springer Berlin / Heidelberg, 2006.

[Hil08] Gijs Hillenius. De: Foreign ministry: 'cost of open source desktop maintenance is by far the lowest'. `http://www.osor.eu/news/de-foreign-ministry-cost-of-open-source-desktop-maintenance-is-by-far-the-lowest`, Oct. 2008. [Online; besucht 29.Oktober 08].

[Hir75] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, 1975.

[IMM+01] Shingo Inoue, Kyoko Makino, Ichiro Murase, Osamu Takizawa, Tsutomu Matsumoto, and Hiroshi Nakagawa. A proposal on information hiding methods using xml. In *The 1st Workshop on NLP and XML*, Nov. 2001.

[IPT99] IPTC. Iptc-naa information interchange model version 4. `http://www.iptc.org/std/IIM/4.1/specification/IIMV4.1.pdf`, Jul. 1999.

[ISO06] ISO. Information technology – open document format for office applications (opendocument) v1.0. `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43485`, Nov. 2006. ISO/IEC 26300:2006.

[JEI02] JEITA. Exchangeable image file format for digital still cameras: Exif version 2.2. `http://www.exif.org/Exif2-2.PDF`, Apr. 2002.

[JM00] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice-Hall, 1 edition, 2000.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, 9:5–38, Jan. 1883.

[KP00] Stefan Katzenbeisser and Fabien A. Petitcolas, editors. *Information Hiding Techniques for Steganography and Digital Watermarking*. Artech House, Inc., Norwood, MA, USA, 2000.

[Lag07] Philippe Lagadec. Opendocument and open xml security (openoffice.org and ms office 2007). *Journal in Computer Virology*, 2007.

[Lam73] Butler W. Lampson. A note on the confinement problem. In *Communications of the ACM*, volume 16, pages 613–615, New York, NY, USA, 1973. ACM Press.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966. (Russisch).

[LGJ04] Yingjiu Li, Huiping Guo, and Sushil Jajodia. Tamper detection and localization for categorical data using fragile watermarks. In *DRM '04: Proceedings of the 4th ACM workshop on Digital rights management*, pages 73–82, New York, NY, USA, 2004. ACM.

[LR97] Benoit Lavoie and Owen Rambow. A fast and portable realizer for text generation systems. In *Proceedings of the fifth conference on Applied natural language processing*, pages 265–268, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[LRAP05] Antonio Lioy, Gianluca Ramunno, Marco Domenico Aime, and Massimiliano Pala. Motivations for a theoretical approach to wysiwys. In *Communications and Multimedia Security*, pages 289–290, 2005.

[MCS+06] A. K. Mikkilineni, P.-J. Chiang, S. Suh, G. T.-C. Chiu, J. P. Allebach, and E. J. Delp. Information embedding and extraction for electrophotographic printing processes. In *Proceedings of the SPIE International Conference on Security, Steganography, and Watermarking of Multimedia Contents VIII*, 2006.

[Mic07] Microsoft. Openxml developer. `http://openxmldeveloper.org/default.aspx`, 2007. [Online; besucht 11. Oktober 2007].

[Mil87] Jonathan K. Millen. Covert channel capacity. *IEEE Symposium on Security and Privacy*, page 60, 1987.

[Mil95] George A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.

[MJF07] Frank J. Mabry, John R. James, and Aaron J. Ferguson. Unicode steganographic exploits: Maintaining enterprise border security. *Security & Privacy Magazine, IEEE*, 5(5):32–39, 2007.

[MM06] Matteo Magnani and Danilo Montesi. A unified approach to structured and xml data modeling and manipulation. *Data & Knowledge Engineering*, 59(1):25–62, Oct. 2006.

[MW01] N. Memon and Ping Wah Wong. A buyer-seller watermarking protocol. *Image Processing, IEEE Transactions on*, 10(4):643–649, Apr. 2001.

[Nat02] National Institute of Standards and Technology. Secure hash standard. `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf`, Aug. 2002.

[Ngo06] Tom Ngo. Office open xml overview. `http://www.ecma-international.org/news/TC45_current_work/OpenXML%20White%20Paper.pdf`, Oct. 2006.

[NL05] Wilfred Ng and Ho Lam Lau. Effective approaches for watermarking xml data. In *DASFAA*, pages 68–80, 2005.

[OAS07] OASIS. Open document format for office applications (opendocument) specification v1.1. `http://docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1-html/OpenDocument-v1.1.html`, Feb. 2007.

[Obj07] Object Management Group. Omg unified modeling language (omg uml), infrastructure, v2.1.2. `http://www.omg.org/docs/formal/07-11-04.pdf`, Nov. 2007.

[Odl07] Andrew Odlyzko. Digital rights management: desirable, inevitable, and almost irrelevant. In *DRM '07: Proceedings of the 2007 ACM workshop on Digital Rights Management*, pages 39–40, New York, NY, USA, 2007. ACM.

[ON92] Boyan A. Onyshkevych and Sergei Nirenburg. Lexicon, ontology, and text meaning. In *Proceedings of the First SIGLEX Workshop on Lexical Semantics and Knowledge Representation*, pages 289–303, London, UK, 1992. Springer-Verlag.

[Owe02] Mark Owens. A discussion of covert channels and steganography, Mar. 2002.

[Pfi96] Birgit Pfitzmann. Information hiding terminology - results of an informal plenary meeting and additional proposals. In *Proceedings of the First International Workshop on Information Hiding*, pages 347–350, London, UK, 1996. Springer-Verlag.

[PKW07] PKWARE. .zip file format specification. `http://www.pkware.com/documents/casestudies/APPNOTE.TXT`, Sep. 2007.

[PP06] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing (4th Edition)*. Prentice Hall PTR, 2006.

[Pro07] The XTAG Project. Xtag webseite. `http://www.cis.upenn.edu/~xtag/`, 12 2007. [Online; besucht 30. September 2008].

[pte07] pte (Autorenkürzel). Musik mit ident-code. `http://www.sueddeutsche.de/computer/artikel/716/116600/`, Jun. 2007. Sueddeutsche.de.

[PW08] Henrich C. Pöhls and Lars Westphal. Die untiefen der neuen xml-basierten dokumentenformate. In *15. Workshop Sicherheit in vernetzten Systemen*. DFN-CERT, Christian Paulsen, Feb. 2008.

[Pyt08a] Python. Python programming language – official website. `http://www.python.org/`, 2008. [Online; besucht 19. September 2008].

[Pyt08b] Python Library Reference. hashlib – secure hashes and message digests. `http://docs.python.org/lib/module-hashlib.html`, Feb. 2008. [Online; besucht 19. September 2008].

[Pyt08c] Python Library Reference. xml.dom – the document object model api. `http://docs.python.org/lib/module-xml.dom.html`, Feb. 2008. [Online; besucht 19. September 2008].

[Pyt08d] Python Library Reference. zipfile – work with zip archives. `http://docs.python.org/lib/module-zipfile.html`, Feb. 2008. [Online; besucht 19. September 2008].

[Pyt08e] Python Software Foundation. Python documentation. `http://www.python.org/doc/`, 2008. [Online; besucht 06. September 2008].

[RSA83] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1):96–99, 1983.

[SA04] Radu Sion and Mikhail Atallah. Attacking digital watermarks. In Edward J. Delp III. and Ping Wah Wong, editors, *Security, Steganography, and Watermarking of Multimedia Contents VI. Proceedings of the SPIE*, volume 5306 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 848–858, Jun. 2004.

[SAP03] Radu Sion, Mikhail Atallah, and Sunil Prabhakar. Resilient information hiding for abstract semi-structures. In *Digital Watermarking, Second International Workshop, IWDW*, pages 141–153, 2003.

[Sch94] Burkhard Schröder. Pretty good envelope. `http://www.burks.de/stegano/pge.html`, Mar. 1994. [Online; besucht 11. Oktober 2007].

[Shi00] R. Shirey. Internet security glossary. `http://www.ietf.org/rfc/rfc2828.txt`, May 2000. (RFC 2828).

[Sim84] Gustavus J. Simmons. The prisoners' problem and the subliminal channel. In *Advances in Cryptology, Proceedings of CRYPTO '83*, pages 51–67, 1984.

[Sio04] Radu Sion. Proving ownership over categorical data. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 584–595, 2004.

[Smi08] Rich Smith. Doppelgänger: An approach to protecting against unknown vulnerabilities. HP Labs, Systems Security Lab, 2008.

[SRC84] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, 1984.

[Tra07] Gina Trapani. Geek to live: Hide data in files with easy steganography tools. `http://lifehacker.com/software/privacy/geek-to-live--hide-data-in-files-with-easy-steganography-tools-230915.php`, Jan. 2007. [Online; besucht 11. Oktober 2007].

[TTA06a] Mercan Topkara, Umut Topkara, and Mikhail J. Atallah. Words are not enough: sentence level natural language watermarking. In *MCPS '06: Proceedings of the 4th ACM international workshop on Contents protection and security*, pages 37–46, New York, NY, USA, 2006. ACM Press.

[TTA06b] Umut Topkara, Mercan Topkara, and Mikhail J. Atallah. The hiding virtues of ambiguity: quantifiably resilient watermarking of natural language text through synonym substitutions. In *MM&Sec '06: Proceeding of the 8th workshop on Multimedia and security*, pages 164–174, New York, NY, USA, 2006. ACM Press.

[TTA07] Mercan Topkara, Umut Topkara, and Mikhail J. Atallah. Information hiding through errors: a confusing approach. In Edward J. Delp III. and Ping Wah Wong, editors, *Proceedings of SPIE*, volume 6505, page 65050V. SPIE, Feb. 2007.

[TTD05] Mercan Topkara, Cüneyt M. Taskiran, and Edward J. Delp. Natural language watermarking. In *Security, Steganography, and Watermarking of Multimedia Contents*, pages 441–452, 2005.

[vAH04] Luis von Ahn and Nicholas J. Hopper. Public-key steganography. In *EUROCRYPT*, pages 323–341, 2004.

[VPP+01] S. Voloshynovskiy, S. Pereira, T. Pun, J.J. Eggers, and J.K. Su. Attacks on digital watermarks: classification, estimation based attacks, and benchmarks. *Communications Magazine, IEEE*, 39(8):118–126, 2001.

[Wes99] Andreas Westfeld. *Angriffe auf steganographische Systeme*, pages 263–286. Vieweg Braunschweig Wiesbaden, 1999.

[WF74] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.

[Wik08] Wikipedia. Look-up-table — wikipedia, die freie enzyklopädie. `http://de.wikipedia.org/w/index.php?title=Look-Up-Table&oldid=45362425`, 2008. [Online; besucht 21. August 2008].

[Wor97] World Wide Web Consortium. Comparison of SGML and XML. http://www.w3.org/TR/NOTE-sgml-xml-971215.html, Dec. 1997.

[Wor06] World Wide Web Consortium. HTML Working Group Charter. http://www.w3.org/2007/03/HTML-WG-charter, 2006.

[WP00] Andreas Westfeld and Andreas Pfitzmann. Attacks on steganographic systems. In *Information Hiding*, pages 61–76, 2000.

[WS08] J. Wu and D. R. Stinson. Authorship proof for textual document. In *Information Hiding 2008*, 2008. Workshop not yet held.

[YSTM06] K. Yoshioka, K. Sonoda, O. Takizawa, and T. Matsumoto. Information hiding on lossless data compression. In *Intelligent Information Hiding and Multimedia Signal Processing, 2006. IIH-MSP '06. International Conference on*, pages 15–18, 2006.

[Zim80] Hubert Zimmermann. Osi reference model–the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on [legacy, pre - 1988]*, 28(4):425–432, Apr. 1980.

[ZPTM05] Xuan Zhou, HweeHwa Pang, Kian-Lee Tan, and Dhruv Mangla. Wmxml: A system for watermarking xml data, 2005.

Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Alle Quellen, die dem World Wide Web entnommen oder in einer sonstigen digitalen Form verwendet wurden, sind der Arbeit beigefügt.

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Departments Informatik sowie der Webseiten der Universität Hamburg, HAW Hamburg und Universität Passau einverstanden.

Hamburg, den_____ Unterschrift: _____