HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

MASTERARBEIT

Herr
Jeffrey Nitschke

Konzeption, prototypische Implementierung und Evaluierung einer intuitiven Benutzerschnittstelle zur Umsetzung glaubhafter Abstreitbarkeit als Mittel der Antiforensik

2022

Fakultät **Angewandte Computer- und Biowissenschaften**

MASTERARBEIT

Konzeption, prototypische Implementierung und Evaluierung einer intuitiven Benutzerschnittstelle zur Umsetzung glaubhafter Abstreitbarkeit als Mittel der Antiforensik

Autor:
Jeffrey Nitschke

Studiengang:
Cybercrime/Cybersecurity

Seminargruppe:
CY18wC-M

Erstprüfer:
Prof. Dipl.-Ing. Ronny Bodach

Zweitprüfer:
M. Sc. Stefan Schildbach

Mittweida, 2022

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

Conception, prototypical implementation and evaluation of an intuitive user interface for realization of credible deniability as tool of anti-forensics

Author:
Jeffrey Nitschke

Course of Study:
Cybercrime/Cybersecurity

Seminar Group:
CY18wC-M

First Examiner:
Prof. Dipl.-Ing. Ronny Bodach

Second Examiner:
M. Sc. Stefan Schildbach

Mittweida, 2022

Bibliografische Angaben

Nitschke, Jeffrey: Konzeption, prototypische Implementierung und Evaluierung einer intuitiven Benutzerschnittstelle zur Umsetzung glaubhafter Abstreitbarkeit als Mittel der Antiforensik, 89 Seiten, 19 Abbildungen, Hochschule Mittweida, University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften

Masterarbeit, 2022

Referat

In dieser Masterarbeit werden sichere steganografische sowie kryptografische Methoden vorgestellt, erläutert, untersucht und innerhalb eines eigens entwickelten Software-Prototypen mit intuitiver Benutzerschnittstelle kombiniert. Noch immer werden Menschenrechtsverteidigende in totalitären Systemen und anderen Krisengebieten systematisch verfolgt, inhaftiert, gefoltert oder sogar exekutiert, weil ihre digital gespeicherten Daten eine antitotalitäre und investigative Tätigkeit beweisen. Die in dieser Arbeit gesammelten Erkenntnisse sowie der darauf basierende Prototyp sollen zu einem besseren Schutz dieser Menschen beitragen.

Abstract

In this master thesis, secure steganographic and cryptographic methods will be introduced, explained, analyzed and combined inside a self-made software prototype with an intuitive UI. Still, human rights defenders are being hunted, arrested, tortured or even killed in totalitarian systems and other conflict areas because their stored digital data proves anti-totalitarian and investigative activity. The accumulated knowledge in this thesis and the resulting prototype shall contribute to a better protection of such people.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

IV. Abkürzungsverzeichnis

AES	Advanced Encryption Standard
ARGB	Alpha Red Green Blue
ASCII	American Standard Code for Information Interchange
BMP	Windows Bitmap
CBC	Cipher Block Chaining
CFB	Cipher Feedback
DES	Data Encryption Standard
ECB	Electronic Code Book
ECDSA	Elliptic Curve Digital Signature Algorithm
GIF	Graphics Interchange Format
HSV	Hue Saturation Value
IDE	Integrated Development Environment
IV	Initialisierungsvektor
JPEG / JPG	Joint Photographic Experts Group
LSB	Least Significant Bit
MDS	Maximum Distance Separable
MIDI	Musical Instrument Digital Interface
MSB	Most Significant Bit
PBKDF2	Password-Based Key Derivation Function 2
PHT	Pseudo-Hadamard Transform
PKCS	Public-Key Cryptography Standards
PNG	Portable Network Graphics
RC4	Rivest Cipher 4 / Ron's Code 4
RGB	Red Green Blue
RGBA	Red Green Blue Alpha
RS	Reed-Solomon
RSA	Rivest-Shamir-Adleman
S-Box	Substitutionsbox
TIFF	Tagged Image File Format
UI	User Interface
XOR	Exklusiv-Oder

1 Einleitung

1.1 Motivation

Die immer weiter voranschreitende Digitalisierung geht auch an den Tätigkeitsfeldern der Menschenrechtsarbeit und des investigativen Journalismus nicht vorbei. So bietet laut [Kathayat, 2020] und [Purnama, 2019] beispielsweise der Tor-Browser Möglichkeiten, um sich in autoritären Staaten möglichst diskret mit Teilen des Internets zu verbinden, die normalerweise in dem jeweiligen Land gesperrt sind. Whistleblower wie Edward Snowden wiederum verwenden starke Kryptografie [Shah, 2020], um ihre Aktivitäten geheim zu halten und damit sensible Daten sowie sich selbst zu schützen. In totalitären Systemen liegt eine umgekehrte Rollenverteilung zwischen den Verfolgten und dem Staat vor. Anders als in freien Teilen der Welt ist hier der Staat im Regelfall der Verbrecher, welcher mit rücksichtsloser Brutalität gegen jede Form der Kritik und des Widerstands vorgeht. Aufgrund derart bedrohlicher Umstände muss die Datensicherheit in diesen Arbeitsbereichen unbedingt ein extrem hohes Niveau haben. Erstens sind diese Daten meist wichtig, um positive Veränderungen zu initiieren und positiven Einfluss auf die Entwicklung in einem Land zu nehmen. Zweitens ist ihre Sicherheit unzertrennlich mit dem Schutz der zu diesem Sachverhalt ermittelnden Menschen verbunden. Die Menschenrechtsorganisation Amnesty International stellt immer wieder fest, dass totalitäre Systeme mit vollkommen unmenschlicher Brutalität gegen potentielle Regimegegner vorgehen, selbst wenn nur ein vager Verdacht vorliegt [Amnesty International, 2021a] [Amnesty International, 2021b]. Aus diesem Grund sind kryptografische Verfahren allein zumeist nicht ausreichend. Ohne den zugehörigen Schlüssel lassen sich die verschlüsselten Daten zwar nicht interpretieren, aber dennoch auffinden. Der Fund verschlüsselter Daten führt dann mit hoher Wahrscheinlichkeit zur Verdächtigung und menschenrechtswidrigen Behandlung des jeweiligen Menschen. Eine Kombination aus Steganografie und Kryptografie hingegen kann Daten nicht nur vor unautorisiertem Zugriff schützen, sondern diese auch gut verstecken und somit zu einem höheren Schutz der Person beitragen.

1.2 Zielstellung

Das Ziel dieser Arbeit besteht in der Entwicklung eines Software-Prototypen, der in erster Linie drei Kriterien erfüllen soll. Zuerst soll die Daten- und Informationssicherheit durch den Einsatz sicherer Kryptografie gewährleistet werden. Zweitens soll durch den Einsatz diskreter Steganografie eine glaubhafte Abstreitbarkeit antitotalitärer Tätigkeiten oder anderer zu unrechtmäßiger Verfolgung führenden Gründe gewährleistet werden. Drittens ist eine einfache Bedienung des Prototypen durch den Einsatz einer intuitiven UI auch für Persönlichkeiten ohne entsprechende Fachkenntnisse zu ermöglichen. Im Vorfeld der Softwareentwicklung erfolgt eine umfangreiche Literaturrecherche.

2 Grundlagen

Das Grundlagenkapitel bildet den Theorie- und Rechercheteil dieser Arbeit und ist damit auch die theoretische Grundlage für alle in der späteren Softwareentwicklung aufgegriffenen Verfahren.

2.1 Kryptographie

In diesem Abschnitt werden die fachlichen Grundlagen auf dem Gebiet der Kryptografie zusammengefasst und erklärt. Zunächst wird ein kurzer Überblick über das Fachgebiet dargeboten. Anschließend erfolgt eine detaillierte Erklärung für zwei bedeutsame Verschlüsselungsalgorithmen, welche später potentiell in den Softwareprototypen implementiert werden.

2.1.1 Allgemeines

Dieser Abschnitt bietet einen groben Überblick über kryaptografische Grundlagen und den aktuellen Stand der Technik auf diesem Gebiet. Alle in Abschnitt 2.1.1 Allgemeines aufgeführten Informationen stammen, wenn nicht anders angegeben, aus [Kerckhoff, 1883], [Paar und Pelzl, 2011] und [Taha et al., 2019].

Definition
Der Begriff Kryptografie geht auf die altgriechischen Wörter κρυπτός/kryptós (deutsch „geheim“, „verborgen“) und γράφειν/gráphein (deutsch „schreiben“) zurück [Liddel et al., 1984, S. 850, 317f]. Unter Kryptographie versteht man in erster Linie das Verschlüsseln von Informationen. Dabei werden in Informationen interpretierbare Daten, meist Klartext genannt, unter Nutzung von einem für die Chiffrierung vorgesehenen Schlüssel in ein nicht mehr ohne Weiteres interpretierbares Format beziehungsweise eine Geheimschrift umgewandelt. Um diesen Vorgang umzukehren und die Chiffre somit zurück in Klartext zu überführen, ist sowohl Kenntnis über das angewandte Verfahren als auch über einen für die Entschlüsselung vorgesehenen Schlüssel notwendig. Neben Verschlüsselungsverfahren gehören aber auch Hashingverfahren zu diesem Gebiet. Fortschrittliche Hashfunktionen ermöglichen die Berechnung von nahezu eindeutigen Hashwerten über im Verhältnis dazu kleine oder auch große Datenmengen. Die Berechnung solcher Werte ist unproblematisch, die Schlussfolgerung der gehashten Informationen aus dem Hashwert wiederum ist praktisch gesehen unmöglich. Hashing-Verfahren werden sehr vielseitig eingesetzt. Ein wichtiges Beispiel ist das Passwort-Hashing, bei dem aus Passwörtern berechnete Hashwerte in Datenbanken gespeichert werden, um diese mit den Hashwerten

beim Login eingegebener Kennwörter abzugleichen. Auf diese Weise kann die Speicherung von Passwörtern in Klartext vermieden werden, was ihre Geheimhaltung vereinfacht. In der IT-Forensik ist es wiederum gängige Praxis, über gesicherte Datenspeicher im Originalzustand Hashwerte zu berechnen, um später nachzuweisen, dass die digitalen Spuren nicht durch das forensische Fachpersonal verfälscht wurden. In Abschnitt 2.2 Schlüsselableitung auf Seite 22 wird später noch näher auf Hashfunktionen eingegangen und darüber hinaus erläutert, wie diese sinnvoll mit symmetrischen Chiffren kombiniert werden können.

Sicherheitskriterien für Verschlüsselungsverfahren

Theoretisch ist es möglich, dass die Sicherheit verschlüsselter Informationen stark von der Geheimhaltung des verwendeten Verfahrens abhängt. Im Verlauf der Geschichte hat sich diese Sicherheitsstrategie jedoch keineswegs bewährt, da sie mit zwei unbestreitbaren Risiken von herausragendem Gewicht verbunden ist. Das erste Risiko besteht darin, dass das genutzte Verfahren und Angaben zu dessen Ablauf durch Eingeweihte verraten werden oder auch mit Gewalt von Unbefugten erlangt werden können. Der zweite große Nachteil dieser Strategie betrifft den Entwicklungsprozess des Verfahrens. Wenn nur ein kleiner Personenkreis mit dem Verfahren und seiner Funktionsweise vertraut ist, so kann auch nur eine sehr überschaubare Menschenmenge die Verschlüsselung effektiv auf Schwachstellen prüfen. Das erhöht logischerweise die Wahrscheinlichkeit dafür, dass bestehende Fehler nicht gefunden und behoben werden. In der modernen Kryptografie wendet man daher das Kerckhoffs'sche Prinzip von Auguste Kerckhoffs an, nach dem die Sicherheit eines Verschlüsselungsverfahrens nicht von dessen Geheimhaltung, sondern lediglich von der Geheimhaltung verwendeter Schlüssel abhängen darf. Die bedeutenden modernen Verschlüsselungsalgorithmen sind daher öffentlich zugänglich. Ein weiteres Sicherheitskriterium besteht in der Verwendung ausreichend langer und möglichst schwer vorhersehbarer Schlüssel, um Brute-Force-Angriffe und Wörterbuchattacken zu verhindern. Darüber hinaus existieren verfahrensspezifische Sicherheitsansprüche. So ist es beispielsweise unabdingbar, dass bei RSA möglichst große Primzahlen für die Berechnung der Schlüssel verwendet werden. Ein weniger offensichtliches, aber dennoch enorm wichtiges Merkmal sicherer Verschlüsselungsmethoden besteht außerdem in einer intuitiv zu bedienenden Umsetzung. Ist die korrekte Benutzung eines technisch sicheren Systems sehr umständlich, so führt dies oft zu Bedienungsfehlern aus Komfort- oder Verständnisgründen. Aus diesem Grund ist es zum Beispiel nicht empfehlenswert, Menschen zur Nutzung dreißigstelliger Passwörter zu zwingen. In diesem Fall würden vermutlich viele Personen sehr lineare Passwörter, wie „1234567…“ erstellen.

Symmetrische und asymmetrische Verschlüsselung

Bei symmetrischen Verfahren wird für den Ver- und Entschlüsselungsprozess der gleiche geheim zu haltende Schlüssel verwendet. Bedeutende Vertreter dieser Kategorie sind unter anderem AES, TwoFish, Blowfish und DES, wobei DES inzwischen nicht mehr als sicher gilt. Die genannten Beispiele sind Bockchiffren, da die Klartextdaten Blockweise verschlüsselt werden. Es existieren jedoch auch symmetrische Verfahren, bei denen es sich um bitweise verschlüsselnde Stromchiffren handelt. Beispiele dafür sind A5 und RC4, wobei beide über deutliche Vulnerabilitäten verfügen. Bei asymmetrischen Chiffren hingegen kommen im Ver- und Entschlüsselungspozess unterschiedliche Schlüssel zum Einsatz. Bedeutende asymmetrische Algorithmen sind unter anderem RSA und ECDSA. Diese zwei Verschlüsselungsarten gehen mit verschiedenen Vor- und Nachteilen einher. Symmetrische Verfahren sind in der Regel deutlich schneller, insbesondere wenn große Datenmengen verarbeitet werden. Darüber hinaus ist natürlich die Menge der zu verwaltenden Schlüssel kleiner. Der wichtige Vorteil asymmetrischer Verfahren hingegen kommt dann zur Geltung, wenn Ver- und Entschlüsselung nicht durch die selbe Instanz, sondern zum Beispiel durch zwei miteinander kommunizierende Computer erfolgen. Bei einem rein symmetrischen Verfahren müssten die zwei Geräte zunächst den Schlüssel austauschen, damit anschließend verschlüsselt kommuniziert werden kann. Wird der Schlüssel in Klartext übertragen, könnte ein Angreifer diesen jedoch abfangen und mitlesen, wodurch er anschließend die verschlüsselte Kommunikation mitlesen kann. Bei einem asymmetrischen Algorithmus hingegen muss der für die Entschlüsselung nötige Schlüssel niemals kommuniziert werden. Computer A kann in diesem Fall aus einem geheimen Schlüssel einen öffentlichen berechnen und diesen an Computer B übermitteln. Computer B leitet aus dem öffentlichen Schlüssel einen eigenen Schlüssel zur Verschlüsselung ab und sendet die damit erzeugte Chiffre an Computer A. Computer A dechiffriert die Nachricht dann mit dem geheimen Schlüssel. Eine Umkehrung der mathematischen Prozesse, um aus dem Schlüssel von B oder dem öffentlichen Schlüssel den geheimen Schlüssel von A abzuleiten, ist bei den aktuell als sicher geltenden Verfahren nicht praktikabel.

CBC und andere Betriebsmodi für Blockchiffren

Für Blockchiffren existieren unterschiedliche Betriebsmodi, z.B der ECB-, CBC- und CFB-Modus. Der ECB-Modus ist dabei der simpelste Modus, da dieser die Klartextblöcke unabhängig voneinander verschlüsselt. Aus diesem Umstand resultiert ein großes Sicherheitsproblem: Gleicher Klartext führt zu gleichem Geheimtext, also kann bei einer ausreichenden Menge von bekannten Klartext-Chiffre-Paaren für weitere Chiffren der zugrunde liegende Klartext abgeleitet werden. Diesen Angriffstyp bezeichnet man als Known-Plaintext-Angriff. Es liegen jedoch einige Betriebsmodi wie der CBC-Modus vor, welche derartige Angriffe durch einige Schutzmaßnahmen verhindern. Im CBC-Modus laufende Blockchiffren erzeugen vor der Verschlüsselung einen zufälligen Initialisierungsvektor und XOR-verknüpfen diesen mit dem ersten Klartextblock. Jeder weitere zu verschlüsselnde Klartextblock wird wiederum mit dem zuletzt resultierenden Chiffreblock (also mit der verschlüsselten Version des vorhergehenden Klartextblockes) XOR-verknüpft.

Anwendung und Ziele von Kryptografie

Aufgrund der Eigenschaften der im vorletzten Absatz betrachteten Verschlüsselungsformen werden in der Praxis häufig hybride Verfahren eingesetzt. So kann man z.B. Kommunikationsinhalte symmetrisch verschlüsseln und den dabei verwendeten Schlüssel über ein asymmetrisches Verfahren an die andere Kommunikationsinstanz übertragen. Generell trägt Kryptografie aktuell häufig zur Realisierung vertraulicher Kommunikation bei, beispielsweise innerhalb von Messenger-Diensten wie Signal [Marlinspike und Perrin, 2017]. Sollen Daten lokal vor unbefugtem Zugriff durch eine Verschlüsselung des Datenträgers geschützt werden, so kommt es für gewöhnlich nur zur Verwendung eines symmetrischen Verfahrens, da die zu verarbeitende Datenmenge oftmals sehr groß ist und die verschlüsselten Daten nicht an Andere übertragen werden müssen. In diesem Fall und auch bei der verschlüsselten Speicherung von Passwörtern wird die Authentizität durch Kryptographie gewährleistet. Es wird sichergestellt, dass nur eine ganz bestimmte Person Zugriff auf Daten oder einen Account hat. Hashfunktionen wiederum werden häufig zur Kontrolle der Datenintegrität eingesetzt. So wird in der IT-Forensik standardgemäß ein Hashwert über Datenträger im Ausgangszustand berechnet, bevor man diese forensischen Untersuchungen unterzieht. Wenn der Hashwert nach der forensischen Analyse bei erneuter Berechnung unverändert ist, dient dies als Nachweis dafür, dass das forensische Fachpersonal keine Beweisdaten beziehungsweise Spuren verändert und damit verfälscht hat. Weiterhin dienen Hashfunktionen auch häufig zur Erzeugung sicherer Schlüssel für Verschlüsselungsverfahren aus Passwörtern, mehr dazu in Abschnitt 2.2 auf Seite 22. Generell kann moderne Kryptographie sehr vielseitig eingesetzt werden und die hier aufgeführten Szenarien sind lediglich Beispiele.

2.1.2 AES-256

In diesem Abschnitt wird ausführlich auf das symmetrische Verschlüsselungsverfahren AES-256 eingegangen. Alle in Abschnitt 2.1.2 AES-256 aufgeführten Informationen stammen, wenn nicht anders angegeben, aus [Daemen und Rijmen, 2002], [Paar und Pelzl, 2011], [Wikipedia Editierende, 2021b] und [Zabala, 2008].

AES steht für „Advanced Encryption Standard“ und die nachfolgende 256 bezieht sich auf die Länge des Schlüssels, der dem Verfahren zu Beginn zusammen mit dem Klartext übergeben wird. So existieren auch noch die Varianten AES-128 und AES-192. Die NSA initiierte einen internationalen Wettbewerb, beim dem Verschlüsselungsalgorithmen eingereicht werden konnten, welche diese Schlüssellängen unterstützen. Als Sieger ging daraus der Algorithmus Rijndael hervor und wurde somit zum AES ernannt. Die ursprüngliche Algorithmusbezeichnung „Rijndael“ ist an die Namen der Entwickler Joan Daemen und Vincent Rijmen angelehnt. Im Folgenden wird der genaue Ablauf des Verfahrens mit all seinen Schritten erläutert. Es handelt sich hierbei um eine Blockchiffre. Das bedeutet, dass die verwendeten Daten blockweise verschlüsselt werden. Der Klartext wird dazu in Eingabeblöcke zu je 128 Bit unterteilt, welche als 4x4-Matrizen mit einem Byte pro Zelle interpretiert werden (siehe Tabelle 2.1). Auch der zu Beginn übergebene Schlüssel, im Folgenden Originalschlüssel genannt, wird wie in Tabelle 2.2 dargestellt als Byte-Matrix aufgefasst.

Tabelle 2.1: Eingabeblock der Größe 128 Bit. Quelle: eigene Arbeit

53	1C	27	9C
66	90	72	37
6D	87	81	B8
9E	0D	07	0D

Tabelle 2.2: Originalschlüssel der Länge 256 Bit. Quelle: eigene Arbeit

A5	7E	95	B4	8B	60	2A	E3
5B	99	59	D0	ED	36	35	0F
67	D5	5A	A6	81	FF	E1	90
E5	CA	AA	24	63	8D	99	9B

Der verwendete Verschlüsselungsalgorithmus Rijndael durchläuft im Fall von AES-256 15 Runden, wobei die erste lediglich aus dem Schritt „AddRoundKey“ besteht. Die anschließenden 13 Runden umfassen wie in Abbildung 2.1 dargestellt die vier Schritte „SubBytes“, „ShiftRows“, „MixColumns“ und „AddRoundKey“. In der letzten Runde hingegen entfällt der Schritt „MixColumns“, weil er an dieser Stelle nicht signifikant zur Steigerung der Sicherheit des Verfahrens beiträgt.

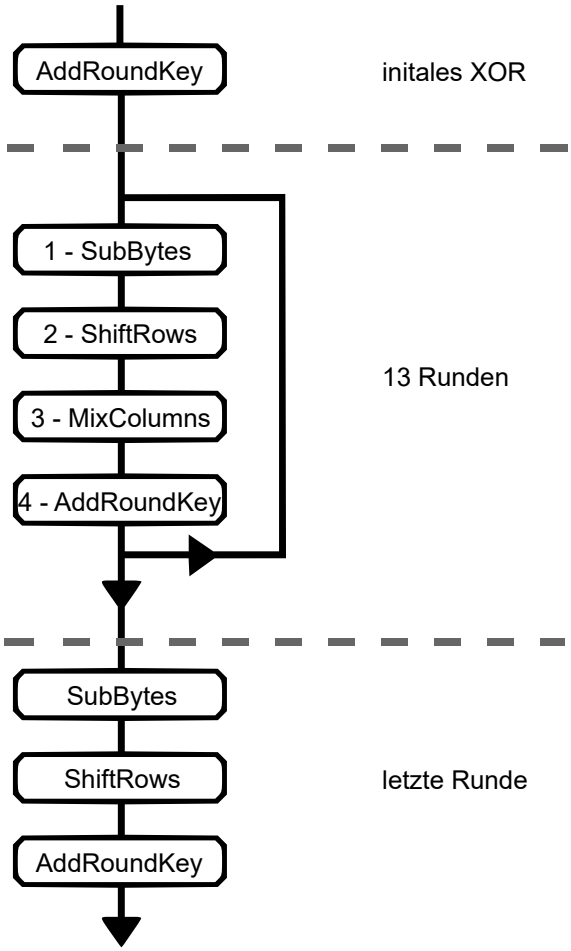


Abbildung 2.1: Übersicht zum allgemeinen Ablauf von AES256.
Quelle: eigene Arbeit, angelehnt an [Kanal AppliedGo, 2017, 0:44 min]

AddRoundKey

Im ersten Schritt des allgemeinen Ablaufs wird der Rundenschlüssel für die aktuelle Runde (in erster Runde R_0) mit dem zu verschlüsselnden Eingabeblock XOR-verknüpft. Auf die Generierung der Rundenschlüssel wird später unter der Überschrift „Schlüsselexpansion“ genauer eingegangen.

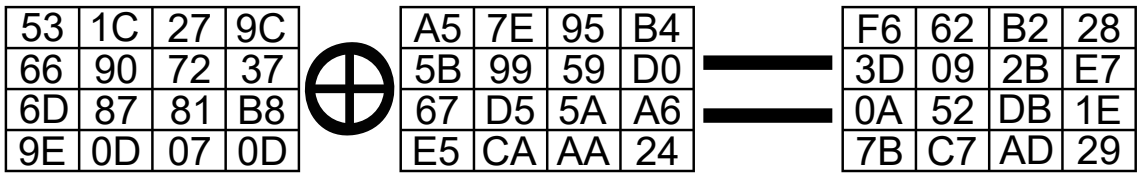


Abbildung 2.2: Erste „AddRoundKey“-Operation im Verlauf des Verfahrens. Quelle: eigene Arbeit

SubBytes

Der nächste in Abbildung 2.1 aufgeführte Schritt nach „AddRoundKey“ lautet „SubBytes“. In dieser Operation werden alle 16 Bytes des als 4x4-Matrix behandelten Eingabeblockes unter Anwendung der in Tabelle .1 dargestellten S-Box durch neue Bytes ersetzt.

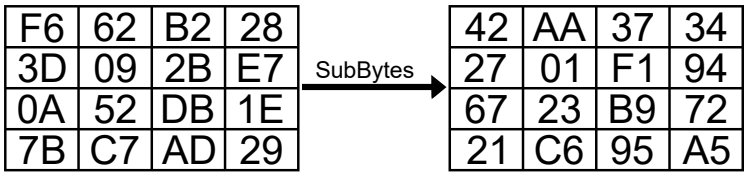


Abbildung 2.3: Erste „SubBytes“-Operation im Verlauf des Verfahrens. Quelle: eigene Arbeit

ShiftRows

Im auf „SubBytes“ folgenden Schritt „ShiftRows“ werden die Bytes der Matrix nach folgendem Schema neu angeordnet:

1. Die erste Zeile bleibt unverändert.
2. Die Elemente der zweiten Zeile werden um ein Byte nach links rotiert.
3. Die Elemente der dritten Zeile werden um zwei Bytes nach links rotiert.
4. Die Elemente der vierten Zeile werden um drei Bytes nach links rotiert.

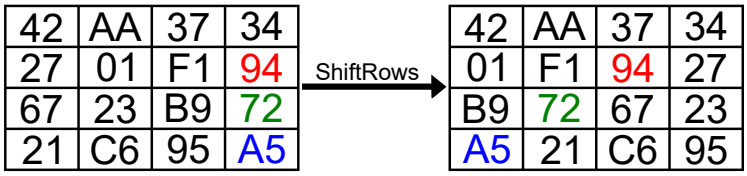


Abbildung 2.4: Erste „ShiftRows“-Operation im Verlauf des Verfahrens. Quelle: eigene Arbeit

MixColumns

Im Schritt „MixColumns“ werden die vier Spalten des Eingabeblockes jeweils als Vektor mit einer im Rijndael-Algorithmus vordefinierten Matrix multipliziert. Das Produkt dieser Operation ist ein neuer Vektor, welcher den alten Spaltenvektor ersetzt. Aufgrund der geltenden Rechenvorschriften ist danach der Wert jedes Bytes im Eingabeblock auch ein Resultat der anderen Bytes, die in der gleichen Spalte des Blockes stehen. Es wird an dieser Stelle ausdrücklich darauf hingewiesen, dass keine klassische Matrixmultiplikation erfolgt, sondern eine wichtige Besonderheit vorliegt. Mehr dazu kann später am Ende dieses Abschnitts in Erfahrung gebracht werden.

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} 42 \\ 01 \\ B9 \\ A5 \end{pmatrix} = \begin{pmatrix} 9B \\ 35 \\ DE \\ 2F \end{pmatrix}$$

Abbildung 2.5: „MixColumns“-Operation für die erste Spalte der Eingabematrix in der zweiten Runde. Quelle: eigene Arbeit

Bei der Entschlüsselung wird das gesamte Prozedere in umgekehrter Reihenfolge ausgeführt. Darüber hinaus wird im Schritt „MixColumns“ entsprechend die inverse Matrix verwendet, um die Multiplikation wieder umzukehren.

Alle bisher geschilderten Operationen in AES-256 sind mathematisch als Operationen auf dem endlichen Körper $GF(2^8)$ definiert und realisiert. In der Praxis bedeutet dies, dass jede im Verfahren durchgeführte Operation jedes Eingangsbyte in genau ein Ergebnisbyte umwandelt. So kann bei der Substitution in „SubBytes“ beispielsweise nicht ein Byte durch zwei Bytes ersetzt werden. Das Einhalten dieser Regel mag im Hinblick auf die meisten Schritte in AES relativ simpel wirken, ist für den Schritt „MixColumns“ jedoch keineswegs trivial. Um die Einhaltung dieser Regel zu gewährleisten, erfolgt nach der Multiplikation eine anschließende Modulo-Division durch ein vordefiniertes irreduzibles Polynom. Da der Fokus dieser Arbeit nicht auf höherer Mathematik liegt, wird bei genauerem Interesse an dieser Stelle auf die Quelle [Paar und Pelzl, 2011] verwiesen. Diese Quelle erwies sich bei der Recherche zum Verfahren und insbesondere der zugrundeliegenden Mathematik nicht nur als inhaltlich überaus hilfreich, sondern auch als gut verständlich.

Schlüsselexpansion

In allen 15 Runden wird jeweils ein individueller Rundenschlüssel im Schritt „AddRoundKey“ verwendet, wobei dieser mit dem Eingabeblock XOR-verknüpft wird. Die Generierung der 15 Rundenschlüssel nennt man Schlüsselexpansion. Die chronologische Auflistung aller 15 erzeugten Schlüssel hintereinander bezeichnet man als den expandierten Schlüssel. Jeder Rundenschlüssel hat eine Länge von 128 Bit beziehungsweise vier 32-Bit-Wörtern, was der Größe jedes zu verschlüsselnden Eingabeblockes entspricht. Der expandierte Schlüssel hat somit eine Länge von $15 \cdot 4 = 60$ 32-Bit-Wörtern. Zu Beginn des Verfahrens wird dem Algorithmus ein vordefinierter Schlüssel einer Länge von 256 Bit übergeben, woraus dann die 15 Rundenschlüssel R_0-R_{14} abgeleitet werden. Diese Ableitung erfolgt nach vier verschiedenen Regeln. Um die Gültigkeitsbereiche dieser im Folgenden erläuterten Regeln besser nachvollziehen zu können, wird ausdrücklich das Hinzuziehen der Tabelle 2.4 auf Seite 13 empfohlen.

1. Die ersten acht 32-Bit-Wörter w_0-w_7, also die ersten 256 Bit, entsprechen dem zu Beginn übermittelten Schlüssel. Die erste Hälfte des Originalschlüssels aus Tabelle 2.2 bildet damit den ersten Rundenschlüssel R_0 und die andere den zweiten, also R_1.
2. Hinter den vom Originalschlüssel beanspruchten 256 Bit verbleiben noch sechseinhalb weitere 256-Bit-Blöcke. Das erste 32-Bit-Wort dieser Blöcke wird stets nach folgendem Schema erzeugt:

   a) Wähle das vorherige Wort w_{i-1}, also die vorherige Spalte in Tabelle 2.4, und rotiere es nach oben.
   b) Substituiere das rotierte Wort mit der S-Box aus Tabelle .1.
   c) XOR-verknüpfe das Wort w_{i-8} aus acht Spalten zuvor mit dem substituierten Wort aus dem letzten Schritt und dem Wort rc_i aus der Rundenkonstanten-Tabelle 2.3, dessen Index dem des Rundenschlüssels R_i gleicht.

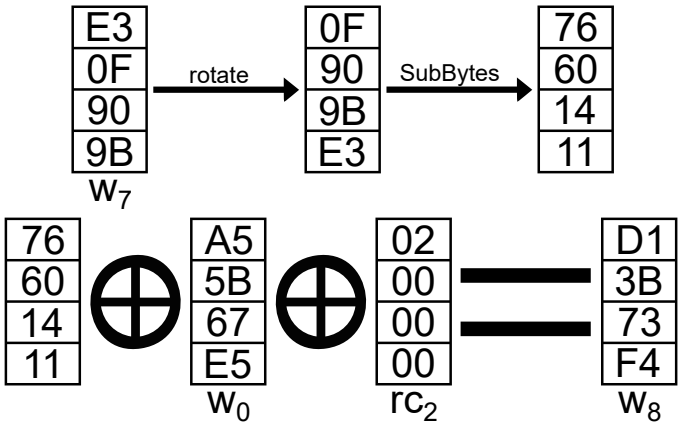


Abbildung 2.6: Demonstration der zweiten Regel für die Bildung von w_8. Quelle: eigene Arbeit

Tabelle 2.3: Rundenkonstantentabelle für AES. Quelle: eigene Arbeit, angelehnt an [Daemen und Rijmen, 2002, S. 214]

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
rc_i	00	01	02	04	08	10	20	40	80	1B	36	6C	D8	AB	4D

3. Das fünfte Wort jedes 256-Bit-Blockes nach dem Originalschlüssel wird nach folgendem Schema erzeugt:

   a) Substituiere das vorhergehende Wort w_{i-1}.
   b) XOR-Verknüpfe das substituierte Wort aus dem letzten Schritt mit dem Wort w_{i-8} aus acht Spalten zuvor.

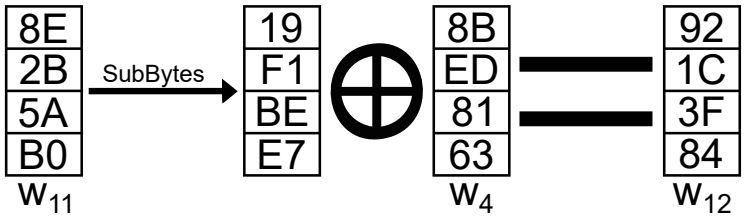


Abbildung 2.7: Demonstration der dritten Regel für die Bildung von w_{12}. Quelle: eigene Arbeit

4. Alle restlichen Wörter werden erzeugt, indem das vorhergehende Wort w_{i-1} mit Wort w_{i-8} aus acht Spalten zuvor XOR-verknüpft wird.

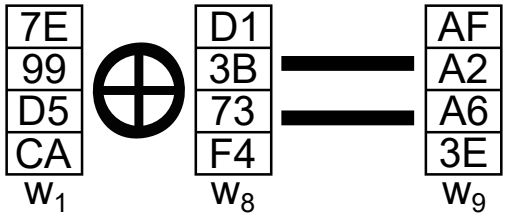


Abbildung 2.8: Demonstration der vierten Regel für die Bildung von w_9. Quelle: eigene Arbeit

Tabelle 2.4: Gültigkeitsbereiche der vier Regeln zur Schlüsselexpansion in AES-256. Quelle: eigene Arbeit

256-Bit-Block B_i*	B_0								B_1								B_2			
Rundenschlüssel R_i	R_0				R_1				R_2				R_3				R_4			
Wie vieltes Wort in B_i?	1.	2.	3.	4.	5.	6.	7.	8.	1.	2.	3.	4.	5.	6.	7.	8.	1.	2.	3.	4.
Wort w_i	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}
Inhalt	A5	7E	95	B4	8B	60	2A	E3	D1	AF	3A	8E	92	F2	D8	3B	...	...	...	...
des	5B	99	59	D0	ED	36	35	0F	3B	A2	FB	2B	1C	2A	1F	10	...	...	...	...
expandierten	67	D5	5A	A6	81	FF	E1	90	73	A6	FC	5A	3F	C0	21	B1	...	...	...	...
Schlüssels	E5	CA	AA	24	63	8D	99	9B	F4	3E	94	B0	84	09	90	0B	...	...	...	...
Greifende Regel	1.								2.		4.		3.		4.		2.		4.	

256-Bit-Block B_i*	B_2				B_3								B_4							
Rundenschlüssel R_i	R_5				R_6				R_7				R_8				R_9			
Wie vieltes Wort in B_i?	5.	6.	7.	8.	1.	2.	3.	4.	5.	6.	7.	8.	1.	2.	3.	4.	5.	6.	7.	8.
Wort w_i	w_{20}	w_{21}	w_{22}	w_{23}	w_{24}	w_{25}	w_{26}	w_{27}	w_{28}	w_{29}	w_{30}	w_{31}	w_{32}	w_{33}	w_{34}	w_{35}	w_{36}	w_{37}	w_{38}	w_{39}
Inhalt	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
des	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
expandierten	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Schlüssels	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Greifende Regel	3.	4.			2.	4.			3.	4.			2.	4.			3.	4.		

256-Bit-Block B_i*	B_5								B_6								B_7			
Rundenschlüssel R_i	R_{10}				R_{11}				R_{12}				R_{13}				R_{14}			
Wie vieltes Wort in B_i?	1.	2.	3.	4.	5.	6.	7.	8.	1.	2.	3.	4.	5.	6.	7.	8.	1.	2.	3.	4.
Wort w_i	w_{40}	w_{41}	w_{42}	w_{43}	w_{44}	w_{45}	w_{46}	w_{47}	w_{48}	w_{49}	w_{50}	w_{51}	w_{52}	w_{53}	w_{54}	w_{55}	w_{56}	w_{57}	w_{58}	w_{59}
Inhalt	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
des	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
expandierten	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Schlüssels	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Greifende Regel	2.	4.			3.	4.			2.	4.			3.	4.			2.	4.		

* von B_7 wird nur die erste Hälfte erzeugt, daher ist B_7 ein 128-Bit-Block

2.1.3 TwoFish

In diesem Abschnitt wird ausführlich auf das symmetrische Verschlüsselungsverfahren TwoFish für eine Schlüssellänge von 256 Bit eingegangen. Alle Quellen in Abschnitt 2.1.3 TwoFish aufgeführten Informationen stammen, wenn nicht anders angegeben, aus [Kanal Abdullah AlQahtani, 2015], [Paar und Pelzl, 2011] und [Schneier et al., 1998].

TwoFish ist wie der zum AES ernannte Rijndael-Algorithmus eine Blockchiffre. TwoFish gewann zwar nicht den AES-Wettbewerb, befand sich jedoch unter den 5 Finalisten. Auch dieser Algorithmus unterstützt aufgrund der Regeln des damals ausgerufenen Wettstreites Schlüssellängen von 128, 192 und 256 Bit. Da die 256-Bit Variante mit der höchsten Sicherheit einhergeht, wird im Folgenden nur für diese Schlüssellänge detailliert das Verfahren erläutert. Zunächst wird der Klartext in 128 Bit große Eingabeblöcke zerlegt. Außerdem wird dem Verfahren zu Beginn ein Schlüssel übergeben, welcher als 256-Bit-Block interpretiert wird. Dieser zu Beginn überlieferte Schlüssel wird in den nachfolgenden Ausführungen zur einfacheren Unterscheidung als Originalschlüssel bezeichnet. Diese Blöcke werden dann wiederum wie in den Tabellen 2.5 und 2.6 dargestellt in vier ($P_0 ... P_3$) beziehungsweise acht ($M_0 ... M_7$) 32-Bit-Wörter unterteilt.

Tabelle 2.5: Eingabeblock der Größe 128 Bit. Quelle: eigene Arbeit

P0	**P1**	**P2**	**P3**
p_0	p_4	p_8	p_{12}
p_1	p_5	p_9	p_{13}
p_2	p_6	p_{10}	p_{14}
p_3	p_7	p_{11}	p_{15}

Tabelle 2.6: Originalschlüssel der Länge 256 Bit. Quelle: eigene Arbeit

M0	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**
m_0	m_4	m_8	m_{12}	m_{16}	m_{20}	m_{24}	m_{28}
m_1	m_5	m_9	m_{13}	m_{17}	m_{21}	m_{25}	m_{29}
m_2	m_6	m_{10}	m_{14}	m_{18}	m_{22}	m_{26}	m_{30}
m_3	m_7	m_{11}	m_{15}	m_{19}	m_{23}	m_{27}	m_{31}

TwoFish ist eine Feistel-Chiffre. Das bedeutet, dass jeder Eingabeblock in zwei Hälften L (P_0, P_1) und R (P_2, P_3) zerlegt wird. Die rechte Hälfte R wird unter Kombination mit Rundenschlüsselbytes durch eine Rundenfunktion F modifiziert. Der resultierende Funktionswert wird dann mit der linken Hälfte L des Eingabeblockes XOR-verknüpft und ersetzt den bisherigen Wert von L. Abschließend werden die Hälften L und R vertauscht. Aufgrund dieses modularen/funktionalen Aufbauschemas könnte man einen modularen Erkläransatz wählen, der die einzelnen Funktionen separat erläutert. Aufgrund einer besseren Verständlichkeit und einer direkteren Verdeutlichung der tatsächlichen Ablaufreihenfolge der einzelnen Schritte, wird in dieser Arbeit ein chronologischer Erkläransatz gewählt.

Der TwoFish-Algorithmus durchläuft wie in Abbilldung 2.9 dargestellt 16 Runden, nachdem der „Input Whitening“-Schritt erfolgt ist. Analog dazu wird nach Abschluss der letzten Runde außerdem ein „Output Whitening“ durchgeführt. Die einzelnen Runden des Verfahrens bestehen aus schlüsselabhängigen S-Boxen, einer MDS-Matrix, einer Pseudo-Hadamard-Transformation, der Schlüsseladdition, Rotations- und XOR-Operationen sowie einer Bytevertauschung. Das Vertauschen der Bytes entfällt für die letzte Runde oder wird für diese umgekehrt.

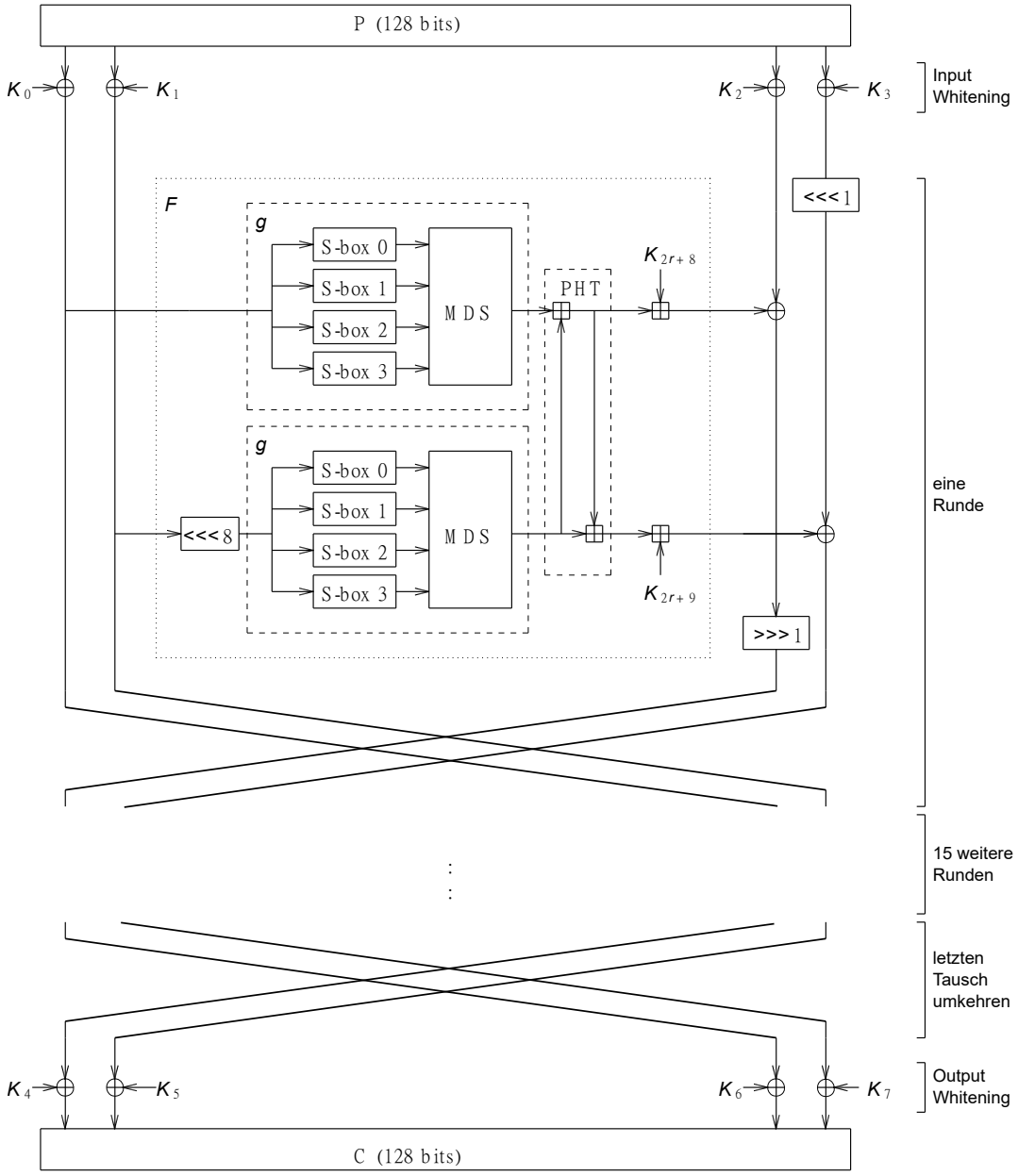


Abbildung 2.9: Übersicht zum allgemeinen Ablauf von TwoFish.
Quelle: eigene Arbeit, angelehnt an [Schneier et al., 1998, S. 6]

Input Whitening

Beim sogenannten „Input Whitening“ wird der Eingabeblock mit den Schlüsselwörtern $K_0 ... K_3$ XOR-verknüpft. Insgesamt existeren 40 verschiedene K-Schlüssel von K_0 bis K_{39}. Auf deren Erzeugung wird später unter der Überschrift „Schlüsselexpansion“ genauer eingegangen. Das Ergebnis der genannten XOR-Operation sind die vier 32-Bit-Wörter $R_0 ... R_3$, siehe Abbildung 2.10. Diese Wörter ersetzen den ursprünglichen Eingabeblock.

$$(P_0, P_1, P_2, P_3) \oplus (K_0.K_1, K_2, K_3) = (R_0, R_1, R_2, R_3)$$

Abbildung 2.10: „Input Whitening“.
Quelle: eigene Arbeit, angelehnt an [Schneier et al., 1998, S. 7]

S-Boxen

Anschließend beginnt die erste TwoFish-Runde. Der erste Schritt jeder Runde besteht darin, dass die ersten zwei Wörter R_0 und R_1 des aktuellen Eingabeblockes jeweils die schlüsselabhängigen S-Boxen durchlaufen. Der Aufbau dieser S-Boxen ist in Abbildung 2.11 dargestellt.

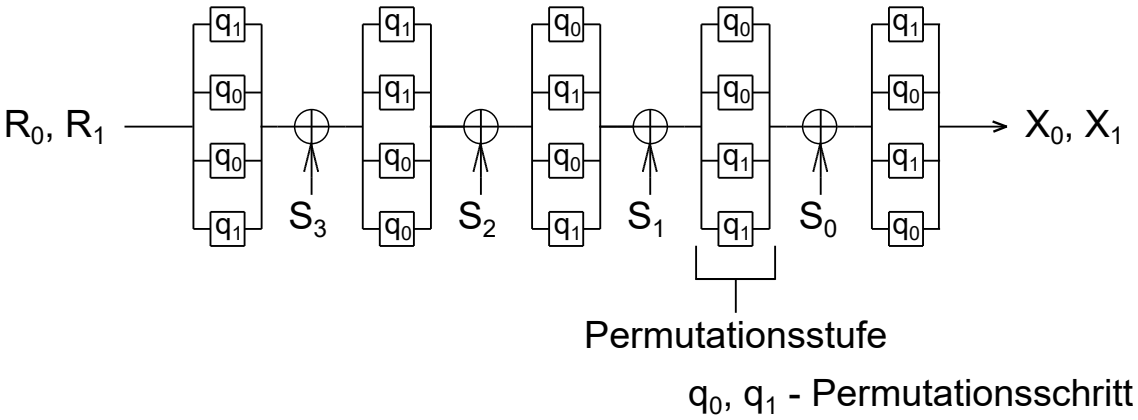


Abbildung 2.11: Der S-Box-Schritt in TwoFish.
Quelle: eigene Arbeit, angelehnt an [Schneier et al., 1998, S. 9]

Wie in der Grafik zu erkennen, wird R_0 zum Wort X_0 und R_1 zum Wort X_1 gewandelt. Insgesamt durchläuft jedes der beiden Wörter fünf Permutationsstufen mit je vier Permutationsschritten q_0 und q_1, welche jeweils ein Byte des Wortes modifizieren. So wird beispielsweise auf das erste Byte von R_0 und R_1 in der ersten Stufe der Permutationsschritt q_1 angewandt, auf das zweite Byte q_0 und so weiter. Nach jeder der ersten vier Permutationsstufen wird das Wort darüber hinaus mit einem der vier S-Schlüssel XOR-verknüpft. Wie diese Schlüssel zustande kommen, wird später unter der Überschrift „Schlüsselexpansion“ detailliert erklärt. Auch bei den S-Schlüsseln handelt es sich um 32-Bit-Wörter.

Es folgt eine genaue Beschreibung der Permutationsschritte. q_0 und q_1 verfügen über eine identische Struktur, was hier am Beispiel von R_0 in der ersten Stufe demonstriert wird:

1. Das Eingabebyte N des Wortes R_0 mit den Bits $n_0...n_7$ wird in die zwei 4-Bit-Nibbles a_0 (linke Bytehälfte) und b_0 (rechte Bytehälfte) zerlegt.
   $N = (n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7)$
   $a_0 = (n_0, n_1, n_2, n_3)$
   $b_0 = (n_4, n_5, n_6, n_7)$
2. a_1 wird als XOR-Verknüpfung der beiden 4-Bit-Nibbles berechnet.
   $a_1 = a_0 \oplus b_0$
3. Anschließend wird b_1 ermittelt. Dazu wird das um 4 Bits nach rechts rotierte b_0 erst mit a_0 und dann mit dem Achtfachen von a_0 modulo 16 XOR-verknüpft.
   $b_1 = a_0 \oplus ROR_4(b_0, 1) \oplus 8a_0 \bmod 16$
4. Im Anschluss werden a_2 und b_2 durch die zum Permutationsschritt passende Substitutionstabelle erzeugt, siehe Tabelle 2.7 und Tabelle 2.8 auf Seite 18.
   $a_2 = t_0[a_1]$
   $b_2 = t_1[b_1]$
5. Nun wird a_3 als XOR-Verknüpfung der beiden 4-Bit-Nibbles a_2 und b_2 berechnet.
   $a_3 = a_2 \oplus b_2$
6. Anschließend wird b_3 ermittelt. Dazu wird das um 4 Bits nach rechts rotierte b_2 erst mit a_2 und dann mit dem Achtfachen von a_2 modulo 16 XOR-verknüpft.
   $b_3 = a_2 \oplus ROR_4(b_2, 1) \oplus 8a_2 \bmod 16$
7. Im Anschluss werden a_4 und b_4 durch die zum Permutationsschritt passende Substitutionstabelle erzeugt, siehe Tabelle 2.7 und Tabelle 2.8 auf Seite 18.
   $a_4 = t_2[a_3]$
   $b_4 = t_3[b_3]$
8. Abschließend werden die 4-Bit-Nibbles a_4 und b_4 wieder zu einem gemeinsamen Byte O kombiniert.
   $a_4 = (o_0, o_1, o_2, o_3)$
   $b_4 = (o_4, o_5, o_6, o_7)$
   $O = (o_0, o_1, o_2, o_3, o_4, o_5, o_6, o_7)$

Der einzige Unterschied zwischen den Permutationsschritten q_0 und q_1 liegt also in der Verwendung verschiedener Substitutionstabellen.

Tabelle 2.7: Substitutionstabelle für q_0.
Quellen: [Schneier et al., 1998, S. 10] und [Kanal Abdullah AlQahtani, 2015, 3:59min]

q_0	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
t_0	8	1	7	D	6	F	3	2	0	B	5	9	E	C	A	4
t_1	E	C	B	8	1	2	3	5	F	4	A	6	7	0	9	D
t_2	B	A	5	E	6	D	9	0	C	8	F	3	2	4	7	1
t_3	D	7	F	4	1	2	6	E	9	B	3	0	8	5	C	A

Tabelle 2.8: Substitutionstabelle für q_1.
Quellen: [Schneier et al., 1998, S. 10] und [Kanal Abdullah AlQahtani, 2015, 3:59min]

q_1	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
t_0	2	8	B	D	F	7	6	E	3	1	9	4	0	A	C	5
t_1	1	E	2	B	4	C	3	7	6	D	A	5	F	9	0	8
t_2	4	C	7	5	1	6	9	A	0	E	D	8	2	B	3	F
t_3	B	9	5	1	C	3	D	E	6	4	7	F	2	0	8	A

MDS-Matrix

Die durch die S-Boxen in Abbildung 2.11 erzeugten Wörter X_0 und X_1 werden dann jeweils mit einer für das Verfahren vordefinierten MDS-Matrix multipliziert. Die entstehenden Produkte heißen T_0 und T_1.

$$\begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \times \begin{pmatrix} X_{0,0} \\ X_{0,1} \\ X_{0,2} \\ X_{0,3} \end{pmatrix} = \begin{pmatrix} T_{0,0} \\ T_{0,1} \\ T_{0,2} \\ T_{0,3} \end{pmatrix} = T_0$$

$$\begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \times \begin{pmatrix} X_{1,0} \\ X_{1,1} \\ X_{1,2} \\ X_{1,3} \end{pmatrix} = \begin{pmatrix} T_{1,0} \\ T_{1,1} \\ T_{1,2} \\ T_{1,3} \end{pmatrix} = T_1$$

Abbildung 2.12: Berechnung von T_0 und T_1 durch Multiplikation von X_0 und X_1 mit der MDS-Matrix. Quelle: eigene Arbeit, angelehnt an [Schneier et al., 1998, S. 8]

Hierbei ist zu beachten, dass es sich um eine Multiplikation auf dem endlichen Körper $GF = (2^8)$ handelt. In der Praxis bedeutet dies, dass jedes Eingangsbyte in genau ein Ergebnisbyte überführt wird. Um die Einhaltung dieser Restriktion zu gewährleisten, erfolgt nach der Multiplikation eine Modulo-Division durch ein vom Verfahren vorgegebenes irreduzibles Polynom. Da der Fokus dieser Arbeit jedoch nicht auf höherer Mathematik liegt, wird hier auf die Originalquelle [Schneier et al., 1998] verwiesen, falls ein tieferes Interesse daran besteht.

Pseudo-Hadamard-Transformation
Die aus der Matrixmultiplikation hervorgehenden Produkte T_0 und T_1 werden dann durch eine PHT kombiniert, wodurch man die Werte a' und b' erhält.
$a' = a + b \bmod 2^{32}$
$b' = a + 2b \bmod 2^{32}$

Addition der Rundenschlüssel und Bildung des finalen Rundenwortes
Zu den Werten a' und b' aus der PHT werden nun die Rundenschlüssel K_{2r+8} und K_{2r+9} modulo 2^{32} addiert. Die Summen heißen F_0 und F_1. r ist der Index der aktuellen Runde. Für die erste Runde gilt also:
$F_0 = a' + K_8 \bmod 2^{32}$
$F_1 = b' + K_9 \bmod 2^{32}$

F_0 wird nun mit dem Wort R_2 XOR-verknüpft. R_2 ist das dritte Wort des zu Rundenbeginn vorliegenden Eingabeblockes. Das Ergebnis wird um ein Bit nach rechts rotiert und bildet damit das Wort C_2. F_1 wiederum wird mit dem um ein Bit nach links rotierten Wort R_3 XOR-verknüpft und bildet damit das Wort C_3. R_3 ist das vierte Wort des zu Rundenbeginn vorliegenden Eingabeblockes.
$C_2 = ROR(R_2 \oplus F_0, 1)$
$C_3 = ROL(R_3, 1) \oplus F_1$

C_2 und C_3 ersetzen R_2 und R_3. Es entsteht also der Block R_0, R_1, C_2, C_3, dessen Hälften am Ende der Runde zu C_2, C_3, R_0, R_1 vertauscht werden. Diese Vertauschung entfällt für die letzte Runde, wie schon zuvor auf Seite 15 angemerkt.

Output Whitening
Dieser Schritt erfolgt analog zum „Input Whitening“, jedoch werden stattdessen die Schlüssel $K_4 ... K_7$ verwendet.

Schlüsselexpansion

Für die Durchführung von TwoFish mit einem 256 Bit langen Originalschlüssel werden insgesamt 44 Schlüssel benötigt, dabei handelt es sich um vier S-Schlüssel und 40 K-Schlüssel. Zuerst wird die Erzeugung der in den S-Boxen genutzten S-Schlüssel erläutert. Der 256 Bit lange Schlüsselblock aus Tabelle 2.6 wird dazu in vier Vektoren zu je acht Bytes zerlegt. Daraufhin wird jeder dieser Vektoren mit einer vordefinierten RS-Matrix multipliziert. Die vier entstehenden Produkte sind die 32-Bit-Wörter S_0, S_1, S_2 und S_3.

$$\begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & A5 & 58 & DB & 9E & 03 \end{pmatrix} \times \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \end{pmatrix} = \begin{pmatrix} S_{0,0} \\ S_{0,1} \\ S_{0,2} \\ S_{0,3} \end{pmatrix} = S_0$$

$$\begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & A5 & 58 & DB & 9E & 03 \end{pmatrix} \times \begin{pmatrix} m_8 \\ m_9 \\ m_{10} \\ m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{15} \end{pmatrix} = \begin{pmatrix} S_{1,0} \\ S_{1,1} \\ S_{1,2} \\ S_{1,3} \end{pmatrix} = S_1$$

$$\begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & A5 & 58 & DB & 9E & 03 \end{pmatrix} \times \begin{pmatrix} m_{16} \\ m_{17} \\ m_{18} \\ m_{19} \\ m_{20} \\ m_{21} \\ m_{22} \\ m_{23} \end{pmatrix} = \begin{pmatrix} S_{2,0} \\ S_{2,1} \\ S_{2,2} \\ S_{2,3} \end{pmatrix} = S_2$$

$$\begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & A5 & 58 & DB & 9E & 03 \end{pmatrix} \times \begin{pmatrix} m_{24} \\ m_{25} \\ m_{26} \\ m_{27} \\ m_{28} \\ m_{29} \\ m_{30} \\ m_{31} \end{pmatrix} = \begin{pmatrix} S_{3,0} \\ S_{3,1} \\ S_{3,2} \\ S_{3,3} \end{pmatrix} = S_3$$

Abbildung 2.13: Erzeugung der S-Schlüssel, Quelle: eigene Arbeit, angelehnt an [Schneier et al., 1998, S. 8]

Für die Erzeugung der 40 K-Schlüssel wird der Originalschlüssel aus Tabelle 2.6 in die Vektoren M_e und M_o zerlegt. M_e beinhaltet die Schlüsselwörter mit geradem Index und M_o die mit ungeradem.
$M_e = (M_0, M_2, M_4, M_6)$
$M_o = (M_1, M_3, M_5, M_7)$

Die K-Schlüssel werden dann nach dem Schema in Abbildung 2.14 generiert.

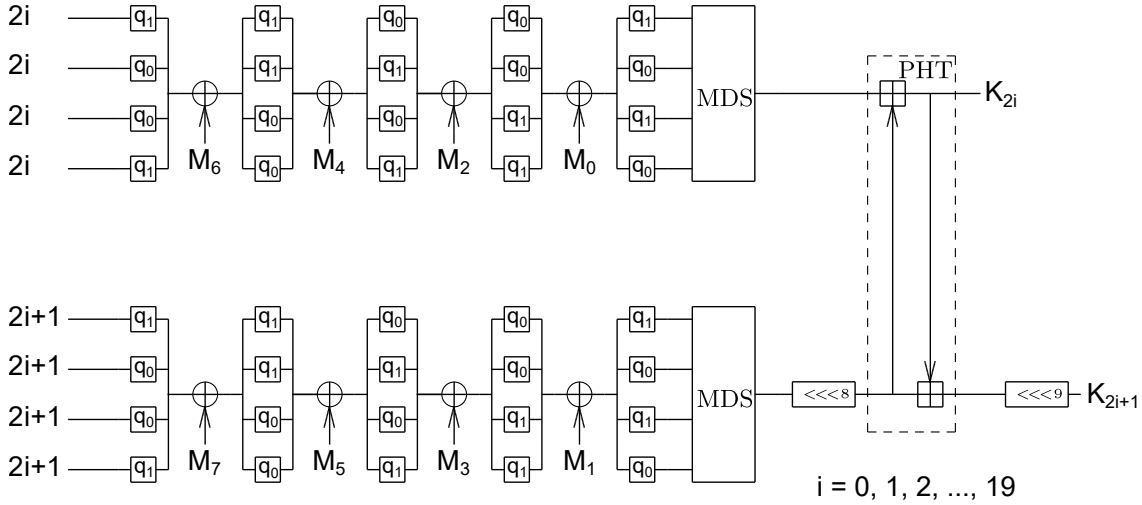


Abbildung 2.14: Erzeugung der K-Schlüssel. Quelle: eigene Arbeit, angelehnt an [Kanal Abdullah AlQahtani, 2015, 5:38min] und [Schneier et al., 1998, S. 9, 11]

Wie in der Grafik zu erkennen ist, nutzt TwoFish ein sehr komplexes Verfahren, um die K-Schlüssel zu generieren. So kommen dabei die bereits behandelten S-Box-Strukturen, die MDS-Matrix und die PHT zum Einsatz. Darüber hinaus fällt auf, dass die Berechnung der Schlüssel paarweise erfolgt.

2.2 Schlüsselableitung

Dieser Abschnitt geht auf die Ableitung kryptografischer Schlüssel aus Passwörtern ein. Zuerst werden allgemeine theoretische Grundlagen zur Schlüsselableitung geschildert. Im Anschluss wird mit PBKDF2 eine konkrete Schlüsselableitungsfunktion genauer vorgestellt.

2.2.1 Allgemeines

In diesem Abschnitt wird ein grober Überblick über die theoretischen Grundlagen der Schlüsselableitung geboten. Außerdem wird auf die Gründe eingegangen, aus denen kryptografische Schlüssel über bestimmte Funktionen erzeugt und nicht direkt vom Menschen eingegeben werden. Alle in Abschnitt 2.2.1 Allgemeines aufgeführten Informationen stammen, wenn nicht anders angegeben, aus [Moriarty et al., 2017], [Paar und Pelzl, 2011], [Turan et al., 2010], [FIP, 2001] und [Wikipedia Editierende, 2021c].

Aus den Abschnitten 2.1.2 sowie 2.1.3 zu AES-256 und TwoFish geht hervor, dass für die Durchführung dieser Verfahren zuerst ein Schlüssel an den jeweiligen Algorithmus übergeben werden muss. Es handelt sich in beiden Fällen um ein symmetrisches Verfahren. Der Schlüssel muss also geheim gehalten werden, da er gleichermaßen zur Ver- und Entschlüsselung dient. Eine direkte Vorgabe des zu nutzenden Schlüssels durch einen Menschen geht mit zwei fundamentalen Risiken für die Sicherheit des Verfahrens einher:

1. Die erste große Gefahr besteht darin, dass unbewusst Schlüssel mit signifikanter Linearität, also einer vorhersehbaren Struktur verwendet werden. Die erforderlichen 256-Bit Schlüssel sollten aber schwer zu erraten und daher möglichst nichtlinear sein, um erfolgreiche Brute-Force-Angriffe sowie Wörterbuchattacken zu verhindern. Selbst wenn sich die anwendende Person dieses Sicherheitskriteriums bewusst ist, können sich unbewusst Muster in die verwendeten Schlüssel einschleichen, welche die Sicherheit der Chiffre gefährden.
2. Darüber hinaus ist auch eine bewusste Verwendung von Schlüsseln mit signifikanter Linearität nicht unwahrscheinlich, da sich die manuelle Eingabe sowie das Einprägen von 256-Bit Schlüsseln als äußerst unpraktisch und umständlich erweisen dürften. Um eine komfortablere Nutzung der Verschlüsselungsverfahren zu ermöglichen, würden die meisten Menschen früher oder später wahrscheinlich zur Verwendung einfach einzuprägender Schlüssel übergehen. Diese Art von Schlüsseln wäre jedoch auch deutlich leichter für angreifende Instanzen vorhersehbar. Einige Personen würden die Schlüssel vielleicht sogar für den Fall des Vergessens notieren. Im schlimmsten Fall erfolgen diese Notizen digital, wodurch der Schlüssel durch einfache Angriffe ausgespäht werden könnte. Wie bereits in Abschnitt 2.1.1 unter „Sicherheitskriterien für Verschlüsselungsverfahren“ dargelegt, sollte eine sichere Verwendung von Chiffren daher möglichst wenige Umstände bereiten.

Um die geschilderten Szenarien beziehungsweise Risiken zu eliminieren, bietet sich die Verwendung von Hashfunktionen zur Schlüsselableitung aus Passwörtern an. Hashfunktionen bilden Eingabewerte unterschiedlicher Länge auf Ausgabewerte fester Länge ab, sogenannte Hashwerte. Im Hinblick auf die Schlüsselableitung können somit 256-Bit Schlüssel aus unterschiedlich langen Passwörtern berechnet werden. Diese Passwörter sind normalerweise deutlich kürzer als der Schlüssel, beispielsweise entsprechen 14 ASCII-Zeichen lediglich 98 Bits. Im Regelfall wird dabei neben dem Passwort auch noch ein Salt als Eingabewert verwendet, dazu später mehr. Durch diese Automatisierung der Schlüsselerzeugung wird der für die sichere Anwendung des Verfahrens zu betreibende Aufwand erheblich auf der menschlichen Seite reduziert, was einen sicheren Umgang mit den Chiffren praktikabler und damit wahrscheinlicher macht. Damit dieses Prozedere aber auch technisch sicher ist, muss die zur Schlüsselableitung eingesetzte Hashfunktion außerdem folgende Kriterien erfüllen:

1. Die Menge der möglichen Ergebnisse muss so groß wie möglich sein, sollte also am besten alle möglichen Bitkombinationen aus 256 Bit umfassen.
2. Die möglichen Ergebnisse sollten mit möglichst gleichverteilten Wahrscheinlichkeiten auftreten. Einfach ausgedrückt bedeutet dies, dass möglichst unterschiedliche Schlüssel generiert werden und es nur extrem selten zur Erzeugung ähnlicher oder bereits verwendeter Schlüssel kommt.
3. Kleine Passwortveränderungen sollten bereits zu großen Hashwert- beziehungsweise Schlüsselveränderungen führen.
4. Die Hashfunktion sollte annähernd kollisionsfrei sein. Die Wahrscheinlichkeit dafür, dass unterschiedliche Eingabewerte auf den gleichen Funktionswert abgebildet werden, sollte also gegen 0 gehen. Aufgrund der Abbildung von Eingabewerten variabler Länge auf Funktionswerte fester Länge ist die Ergebnismenge oft naturgemäß kleiner als die Menge der möglichen Eingabewerte. In so einem Fall müssen also zwangsläufig Kollisionen existieren. Die Wahrscheinlichkeit dafür sollte jedoch möglichst minimal sein. Im Fall der Schlüsselableitung sind die Passwörter jedoch in der Regel wesentlich kürzer als 256-Bit Schlüssel, was die Vermeidung von Kollisionen stark vereinfacht.

Hashfunktionen können wie bereits in Abschnitt 2.1.1 unter „Anwendung und Ziele von Kryptographie“ erwähnt auch zu anderen Zwecken eingesetzt werden. In diesem Masterprojekt spielen sie jedoch ausschließlich für die soeben beschriebene Schlüsselableitung eine Rolle. Beispiele für bedeutende Hashfunktionen sind Argon2, bcrypt, PBKDF2 und Scrypt.

2.2.2 PBKDF2

Dieser Abschnitt stellt mit PBKDF2 eine konkrete Schlüsselableitungsfunktion vor, die möglicherweise später in den aus dieser Arbeit hervorgehenden Softwareprototypen implementiert wird. Alle in Abschnitt 2.2.2 PBKDF2 aufgeführten Informationen stammen, wenn nicht anders angegeben, aus [Grassi et al., 2017], [Moriarty et al., 2017], [Paar und Pelzl, 2011], [Turan et al., 2010], [FIP, 2001] und [Wikipedia Editierende, 2021c].

PBKDF2 steht für „Password-Based Key Derivation Function 2“ und ist eine populäre, standardisierte Hashfunktion zur Ableitung von Schlüsseln aus Passwörtern unter Nutzung eines sogenannten Salts. Der Salt ist dabei ein pseudozufälliger Wert, der nicht geheim gehalten werden muss. Schlussfolgerungen vom Salt auf den Schlüssel oder das ebenfalls zu dessen Berechnung eingesetzte Passwort sind nicht praktikabel. Als weitere Eingabewerte sind der Hashfunktion die gewünschte Länge des resultierenden Hashwertes sowie die Anzahl der Iterationen zu übermitteln. Die Anzahl der Iterationen gibt an, wie oft die Ableitungsfunktion insgesamt angewandt wird. Mit einer höheren Anzahl von Iterationen geht aufgrund des höheren Rechenaufwandes eine größere Sicherheit einher, aber auch eine längere Berechnungszeit.

Abbildung 2.15: Schema für die Ableitung eines 256-Bit-Schlüssels aus einem Passwort mittels PBKDF2. Quelle: eigene Arbeit

PBKDF2 erfüllt die am Ende von Abschnitt 2.2.1 aufgeführten Kriterien für sichere Schlüsselableitungsfunktionen. PBKDF2 verfügt jedoch über Vulnerabilitäten, welche die Ermittlung der für die Hashwerterzeugung genutzten Eingabewerte ermöglichen, insofern einem die Hashwerte vorliegen und die Anzahl der Iterationen zu niedrig ist. Im Anwendungsfall der Schlüsselableitung ist das jedoch unproblematisch, da der erzeugte Hashwert beziehungsweise der Schlüssel genau wie das Passwort nicht persistent im Speicher abgelegt wird. Der einzige nach Abschluss der Verschlüsselung auslesbare Wert ist der Salt. Für bestimmte andere Anwendungsformen, zum Beispiel das Passworthashing, ist PBKDF2 bei unsauberer Implementierung aufgrund dieses Umstandes allerdings ungeeignet.

2.3 Steganographie

In diesem Abschnitt werden die fachlichen Grundlagen auf dem Gebiet der Steganografie zusammengefasst und erklärt. Zunächst wird ein kurzer Überblick über das Fachgebiet dargeboten. Anschließend erfolgt eine detaillierte Erklärung für zwei bedeutsame steganografische Verfahren, welche später potentiell in den Softwareprototypen implementiert werden.

2.3.1 Allgemeines

Dieser Abschnitt bietet einen groben Überblick über kryptografische Grundlagen und den aktuellen Stand der Technik auf diesem Gebiet. Alle in Abschnitt 2.3.1 Allgemeines aufgeführten Informationen stammen, wenn nicht anders angegeben, aus [Taha et al., 2019].

Definition
Der Begriff Steganografie geht auf die altgriechischen Wörter στεγανός/steganós (deutsch „bedeckt“) und γράφειν/gráphein (deutsch „schreiben“) zurück [Liddel et al., 1984, S. 1423, 317f]. Steganografie bezeichnet das Verstecken von Informationen und/oder Kommunikationsprozessen. Die Daten werden also nicht wie bei der Kryptografie (siehe Abschnitt 2.1.1) in eine Geheimschrift umgewandelt, deren Informationen nur Eingeweihten zugänglich sind, sondern das Ziel besteht bereits in der Geheimhaltung der bloßen Existenz dieser Daten. Zur Umsetzung dieser Zielstellung wird in der Regel ein zweiter, im Hintergrund verwendeter Kommunikationskanal benutzt, über dessen Existenz und Funktionsweise nur Eingeweihte Bescheid wissen. Im Vordergrund kann über einen gängigen Kommunikationskanal, beispielsweise gewöhnliche Urlaubsfotos, ein thematisch völlig anderer Inhalt platziert werden.

Sicherheit und Anwendung, Abgrenzung zur Kryptografie
Während die Kryptographie eine sehr mathematische Disziplin ist, was insbesondere in den Ausführungen zu AES-256 in Abschnitt 2.1.2 und TwoFish in Abschnitt 2.1.3 deutlich wird, handelt es sich bei der Steganographie vor allem um eine kreative Disziplin. Aus diesem Grund gibt es auf letzterem Gebiet auch keine wirklichen Standard-Verfahren. Weiterhin hängt die Sicherheit von Steganographie in den meisten Fällen von der Geheimhaltung des Kommunikationsprozesses ab. Wenn man Daten und kommunikative Vorgänge nicht konkret auf die Anwendung einer bestimmten steganographischen Methode untersucht, so sollte diese völlig unentdeckt bleiben. Weiß man jedoch vom sekundären Kommunikationskanal und dessen Regeln, kann die Kommunikation zumeist leicht festgestellt werden. Eine gute Metapher wäre der Eingang zu einem Haus: Die Kryptographie entspricht in diesem Bild einer auffälligen Hochsicherheitstür, die man nur mit dem nötigen Schlüssel öffnen kann. Die Steganographie hingegen entspricht in dieser Metapher

einem Geheimzugang, beispielsweise einem unterirdischen Tunnel, dessen Eingang durch Sträucher verdeckt ist. Aufgrund dieser Unterschiede eignet sich die Kryptografie besonders für den Schutz der Sache, also für den Schutz der zu verschlüsselnden Daten oder im Hinblick auf die Metapher für den Schutz des Areals hinter der verschlossenen Tür. Die Steganografie hingegen ist wegen ihrer Unauffälligkeit besonders gut für den Schutz der Person geeignet, die wegen des Besitzes oder der Übermittlung bestimmter Daten gefährdet ist. Im Hinblick auf die Metapher ist eine Hochsicherheitstür schwerer zu durchqueren als ein versteckter Tunnel, aber letzterer kann dafür ungesehen passiert werden.

2.3.2 RGB-Steganographie

In diesem Abschnitt wird demonstriert, wie als Bitstrom vorliegende Daten steganografisch in Pixelgrafiken versteckt werden können, welche das RGB-Farbraummodell verwenden. Um dieses Verfahren zu verstehen, ist grundlegendes Wissen zur Beschaffenheit von Pixelgrafiken und den darin festgehaltenen Daten erforderlich. Aus diesem Grund werden zu Beginn einige Grundlagen erläutert. Alle in Abschnitt 2.3.2 RGB-Steganografie aufgeführten Informationen stammen, wenn nicht anders angegeben, aus [Boughen, 2003], [Kanal D3NCE, 2020], [MDN Web Docs Editierende, 2021] und [Smith, 1995].

Beschaffenheit von Pixelgrafiken
Eine Pixelgrafik entspricht einem zweidimensionalen Gitter aus einer endlichen Anzahl von Bildpunkten, sogenannten Pixeln. Diese einzelnen Bildelemente lassen sich über entsprechende X- und Y-Koordinaten adressieren, siehe Tabelle 2.9.

Tabelle 2.9: Struktur einer 4×6-Pixelgrafik. Quelle: eigene Arbeit

X\Y	**0**	**1**	**2**	**3**	**4**	**5**
0	P_0	P_4	P_8	P_{12}	P_{16}	P_{20}
1	P_1	P_5	P_9	P_{13}	P_{17}	P_{21}
2	P_2	P_6	P_{10}	P_{14}	P_{18}	P_{22}
3	P_3	P_7	P_{11}	P_{15}	P_{19}	P_{23}

Insofern es sich nicht um eine schwarz-weiße Grafik handelt, müssen für diese einzelnen Pixel jeweils Farbinformationen hinterlegt werden. Bei schwarz-weißen Bildern können statt Farbinformationen sogenannte Grauwerte für jedes Pixel hinterlegt werden, deren Wertebereich 256 verschiedene Helligkeitsstufen eines Pixels beschreibt. Null ist dabei der minimale Wert und entspricht Schwarz, während 255 der Maximalwert ist und Weiß entspricht. Farbinformationen hingegen können durch unterschiedliche Modelle beschrieben werden, beispielsweise durch das HSV- oder das RGB-Modell. Auf Letzteres wird im folgenden Absatz näher eingegangen.

RGB-Raum

RGB steht für Rot-Grün-Blau und bezeichnet ein weit verbreitetes Farbraummodell für die Darstellung unterschiedlicher Farben in Pixelgrafiken. Es handelt sich um einen dreidimensionalen Raum, dessen drei senkrecht aufeinander stehende Achsen den Farben Rot, Grün und Blau entsprechen. Für jede dieser drei Achsen existiert ein Wertebereich von Null bis 255. Der Wertebereich des Farbraumes reicht dem zur Folge von $(0;0;0)$ (Schwarz) bis $(255;255;255)$ (Weiß). Der Aufbau dieses Farbmodells sowie die damit einhergehende Verteilung unterschiedlicher Farben auf den Wertebereich werden in Abbildung 2.16 optisch verdeutlicht.

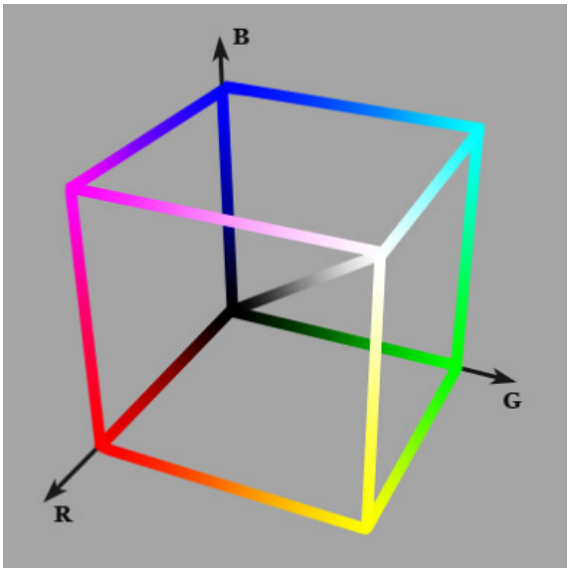


Abbildung 2.16: Darstellung des RGB-Raumes als Würfel. Quelle: [Oscar de Lama, 2014]

Alpha-Kanal

Ein nicht gerade seltenes Phänomen ist die Erweiterung des im letzten Absatz erläuterten RGB-Farbraum-Modells um den sogenannten Alpha-Kanal. Der Alphawert dient zur Angabe der Transparenz eines Pixels und sein Wertebereich reicht von Null (komplett transparent) bis Eins (keine Transparenz, Pixel wird vollständig gerendert). Weiterhin ist zu beachten, dass die Nutzung des Alphakanals an die Verwendung bestimmter Bild-Dateiformate gebunden ist, welche diesen technisch unterstützen. Zwei Beispiele für derartige Dateitypen sind TIFF und PNG. Außerdem muss die zur Verarbeitung der Daten genutzte Software ebenfalls den Einsatz des Alphakanals unterstützen. Auf den Alphakanal wird in diesem Abschnitt nur der Vollständigkeit halber eingegangen, da es nicht unwahrscheinlich ist, dass man bei der Umsetzung von RGB-Steganografie auf die Alphaerweiterung trifft. Für die Realiserung der steganografischen Methode an sich spielt sie jedoch keine Rolle.

Steganografie in Pixelgrafiken unter Nutzung des RGB-Raumes
Die steganografisch zu versteckenden Daten können als Bitstrom betrachtet werden. Bei diesen Eingabedaten kann es sich auch um Chiffren handeln, welche aus den Verfahren in Abschnitt 2.1.2 und 2.1.3 hervorgegangen sind. Wie bereits unter der Überschrift „RGB-Raum“ in diesem Abschnitt erläutert, besitzt jedes Pixel drei verschiedene Farbwerte mit einem jeweiligen Wertebereich von Null bis 255. Diese 256 verschiedenen Abstufungen eines Farbwertes können mit acht Bit kodiert werden, da $2^8 = 256$. Die Basis Zwei ergibt sich bei dieser Rechnung aus den zwei verschiedenen Zuständen Null und Eins, welche die Bits jeweils annehmen können. Der Exponent Acht wiederum ergibt sich aus der Anzahl der Bits. Der zu versteckende Bitstrom wird daher in Subströme zu je 8 Bits zerlegt, aus denen dann Dezimalzahlen von Null bis 255 berechnet werden können. Jeder Substrom kann auf diese Weise als Rot, Grün- oder Blau-Wert eines Bildpixels interpretiert werden. Da die Anzahl der Pixel in den meisten Bilddateien extrem groß ist, sind die daraus resultierenden optischen Veränderungen für die menschliche Wahrnehmung weitgehend unsichtbar, weil sich eben nur ein sehr kleiner Bruchteil alle Farbwerte verändert. Um das beschriebene Prozedere umzukehren, ist lediglich Wissen über die Position, Anzahl und Reihenfolge der in die Steganografie involvierten Pixel erforderlich. Die von der Methode verwendeten Pixel können dabei sehr unterschiedlich selektiert werden. Es folgen zwei Beispiele für die Bestimmung des Pixelpfades.

1. Eine Implementierung kann einen festen Pixelpfad für die vom Verfahren zu verarbeitenden Bilder bestimmen. Ein möglicher Beispielpfad beginnt beim obersten Pixel in der Spalte ganz links und geht dann für jeden 8-Bitstrom ein Pixel nach unten. Wird schließlich das Ende er Spalte erreicht, erfolgt eine analoge Fortsetzung des Pfades in der Spalte rechts daneben.
2. Analog zum ersten Beispiel können mehrere mögliche Pfade von einer Implementierung vordefiniert und dann für jede Umsetzung neu nach einem pseudozufälligen Prinzip ausgewählt werden. Die unterschiedlichen Pfade könnten dann über Indizes eindeutig identifiziert und somit an einem standardisierten Ort im Bild, zum Beispiel die untere rechte Bildecke, durch den Index vermerkt werden.

Grenzen der RGB-Steganografie
Zum Schluss wird hier ausdrücklich darauf hingewiesen, dass für eine korrekte Funktionsweise des Verfahrens die Verwendung kompressionsfreier Dateitypen eine absolute Notwendigkeit ist. Werden Bilddaten zur Reduzierung des Speicherbedarfs komprimiert, kommt es dabei zu einer Veränderung der den Pixeln zugeordneten Farbwerte. Die Kompression verhindert dem zur Folge eine korrekte Interpretation der mittels RGB-Steganografie eingebetteten Daten, weil diese anschließend beim Speichern der Datei modifiziert werden. Ein Beispiel für einen kompressionsfreien Datentyp ist PNG, während JPEG wiederum ein sehr populäres Dateiformat mit Kompression ist. Außerdem ist zu bedenken, dass ein zu großer Anteil der manipulierten Pixel am Gesamtbild die Unauffälligkeit des Verfahren gefährden kann. Je mehr Daten versteckt werden müssen, desto größer ist auch die Menge der durch das Verfahren manipulierten Pixel und damit

die optische Verzerrung. Weiterhin ist denkbar, dass eine stärkere Verteilung der in die Steganografie involvierten Bildpunkte über das gesamte Bild weniger auffällt als eine Konzentration dieser Pixel auf einen bestimmten Bereich in der Grafik. Darüber hinaus ist eine Beeinflussung der Verfahrensdiskretion durch die Farbverläufe in der Trägergrafik denkbar. Durch die Steganografie entstehende Farben könnten je nach Trägerbild besser oder schlechter zur restlichen Grafik passen. Entstehen besonders viele vorwiegend blau gefärbte Pixel durch die Anwendung dieses Verfahrens, wäre beispielsweise ein Bild von wolkenlosem Himmel tagsüber oder auch ein Foto von blauem Meer besonders gut geeignet.

2.3.3 MIDI-Steganographie

In diesem Abschnitt wird demonstriert, wie als Bitstrom vorliegende Daten steganografisch in MIDI-Dateien versteckt werden können. Um diesbezügliche Verfahren zu verstehen, ist grundlegendes Wissen zur Beschaffenheit von MIDI-Dateien erforderlich. Aus diesem Grund werden zunächst entsprechende Grundlagen erläutert. Alle in Abschnitt 2.3.3 MIDI-Steganografie aufgeführten Informationen stammen, wenn nicht anders angegeben, aus [Birch, 2021a], [Hass, 2020], [John, 2004], [MIDI Association, 2021] und [Wu et al., 2019].

Bedeutung und Anwendung von MIDI
Das MIDI-Format wurde entwickelt, um elektronische Instrumente mittels dafür vorgesehener Dateien und Kommunikationsprotokolle automatisiert beliebige Stücke spielen zu lassen. Die Besonderheit besteht darin, dass dies wörtlich zu nehmen ist. Es handelt sich nämlich nicht um Aufnahmen, die unter anderem von Instrumenten abgespielt werden können. Stattdessen entspricht eine MIDI-Datei einer detaillierten Beschreibung, wie ein Stück aufgebaut ist und welche Aktionen am Instrument ausgeführt werden müssen, um dieses zu spielen. Vereinfacht könnte man also sagen, dass MIDI-Dateien für elektronische Instrumente die gleiche Funktion erfüllen wie Notenblätter für den Menschen. Es ist sozusagen eine Anleitung, nach der ein Stück gespielt werden kann.

Struktur von MIDI-Dateien
Am Beginn jeder MIDI-Datei steht zunächst ein „File Header“, welcher sich aus den Informationen in Tabelle 2.10 zusammensetzt. Die Zeichenfolge „MThd“ dient dabei lediglich zur Unterscheidung von der zweiten Headerart, dem „Track Header“. Dieser wird nach den Ausführungen zum „File Header“ genauer erläutert. Das Längenfeld im „File Header“ gibt die Länge des sich anschließenden Datenfeldes in Bytes an, welches aus den Feldern „Format“, „Spuren“ und „Division“ zu jeweils zwei Bytes besteht. Im Längenfeld muss also immer der Wert Sechs stehen. Das Formatfeld gibt an, um welchen MIDI-Typ es es sich handelt, da aktuell drei verschiedene MIDI-Typen existieren: MIDI 0, MIDI 1.0 sowie MIDI 2.0. Auf die genauen Unterschiede zwischen diesen Varianten wird später im Detail eingegangen. Im Spurenfeld ist die Anzahl der in der Datei enthaltenen Spuren und damit auch die Anzahl der „Track Header“ codiert. Das Feld „Division“ gibt die Standard-Deltazeit zwischen den MIDI-Ereignissen vor. Bei der Deltazeit handelt es sich um den zeitlichen Zwischenraum zwischen einem aktuellen und dem jeweils vorherigen Ereignis, mehr dazu später.

Tabelle 2.10: Aufbau eines MIDI File Headers. Quelle: [Birch, 2021a]

Headertyp	**Länge**	**Daten**		
Vier Bytes (ASCII)	32 Bit	16 Bit	16 Bit	16 Bit
MThd	Länge	Format	Spuren	Division

Auf den „File Header“ am Anfang der Datei folgt für jede verwendete Spur genau ein „Track Header“. In jeder gültigen MIDI-Datei muss mindestens ein „Track Header“ vorliegen. Die konkrete Headeranzahl ist wie im letzten Absatz erwähnt dem „File Header“, genauer gesagt dessen Spurenfeld zu entnehmen. Ein „Track Header“ ist wie in Tabelle 2.11 dargestellt strukturiert. „MTrk“ gibt dabei lediglich an, dass es sich um einen Track Header handelt. Aus dem Längenfeld kann man die Länge des sich daran anschließenden Datenfeldes ablesen. Dieses Datenfeld setzt sich dann wiederum aus Paaren von Deltazeiten und diesen zugeordneten Ereignissen (siehe dafür Tabelle .2) zusammen.

Tabelle 2.11: Aufbau eines MIDI Track Headers. Quelle: [Birch, 2021a]

Headertyp	**Länge**	**Daten**	
Vier Bytes (ASCII)	32 Bit	für Länge siehe Inhalt des Längenfeldes	
MTrk	Länge	Deltazeit	Ereignis

Zentraler Bestandteil des MIDI-Formates sind die unterschiedlichen Ereignisse, aus denen sich die einzelnen Spuren einer MIDI-Datei zusammensetzen. Die Tabelle .2 im Anhang gibt einen Überblick über konkrete Steuerereignisse, welche die Spielweise eines Stückes unmittelbar beschreiben und damit beeinflussen. Man bezeichnet die Events in dieser Tabelle als „Channel Voice Messages“. Darüber hinaus existieren auch noch weitere Ereignisgruppen, wie beispielsweise Meta-Ereignisse, die man unter anderem zur Übertragung von Songtexten verwenden kann. Für das unter der Überschrift „Ausnutzung der MIDI-Struktur für die steganografische Einbettung von Bitströmen“ erklärte Verfahren sind jedoch keine ausführlichen Kenntnisse über diese anderen Ereigniskategorien erforderlich.

MIDI-Formate

Wie bereits unter der Überschrift „Struktur von MIDI-Dateien“ angemerkt, existieren die drei verschiedenen MIDI-Versionen „MIDI 0“, „MIDI 1.0“ und „MIDI 2.0“. Alle MIDI-Dateien des Typs Null verfügen über genau eine Spur und damit einen Track Header. Die beiden anderen Formate ermöglichen die Verwendung mehrerer Spuren. Bei MIDI 1.0 werden diese Spuren jedoch immer alle gleichzeitig wiedergegeben, während MIDI 2.0 eine voneinander unabhängige Spurwiedergabe erlaubt.

Steganografische Ausnutzung von „Program Change“-Ereignissen
Ereignisse des Typs „Program Change“ führen wie bereits in Tabelle .2 im Anhang aufgeführt zum Wechsel des Instrumentes. Diese Ereignisse allein beinhalten jedoch keinerlei Anweisungen zum Spielen bestimmter Töne. Solange keine derartigen Instruktionen durch „Note-On“-Ereignisse folgen, bleibt ein „Program Change“ also unhörbar. Aus diesem Umstand leitet sich ein steganografisches Verfahren für MIDI-Dateien ab. Werden mehrere „Program Change“-Ereignisse direkt aufeinander folgend mit einer Deltazeit von Null innerhalb einer MIDI-Datei platziert, so ist nur der letzte Instrumentwechsel hörbar. Letzteres gilt natürlich auch nur dann, wenn anschließend wieder Klänge durch die Verarbeitung anderer Ereignisse erzeugt werden. Man kann Daten daher auf eine für das menschliche Ohr nicht wahrnehmbare Art und Weise in MIDI-Dateien verstecken, indem man sie als Programmnummer in den unhörbaren „Progam Change“-Ereignissen platziert. Zur Veranschaulichung wird der Aufbau derartiger Ereignisse in Abbildung 2.17 dargestellt.

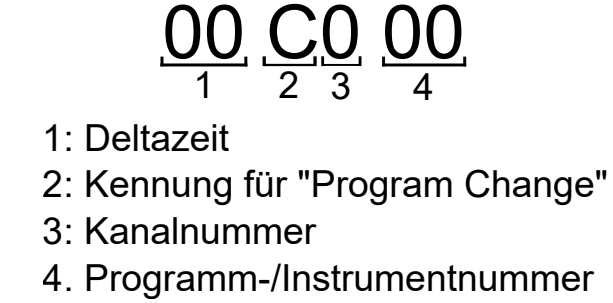


Abbildung 2.17: Aufbau eines „Program Change“-Ereignisses. Quelle: eigene Arbeit

Für eine ordnungsgemäß funktionierende Implementierung müssen jedoch drei besondere Herausforderungen bei der Umsetzung dieser Strategie beachtet werden:

1. Damit der Vorgang rückführbar ist, muss in irgendeiner Form hinterlegt werden, welche und damit auch wie viele Ereignisse in das Verstecken geheimer Informationen eingebunden sind. So könnte man zum Beispiel die Ereignisdaten im ersten „Program Change“ für das Hinterlegen sowie Ablesen der Anzahl von in die Steganografie involvierten Ereignissen nutzen und dementsprechend viele Folgeereignisse für das Schreiben oder Auslesen der Nachricht verwenden.
2. MIDI-Daten sind in „Big Endian“ formatiert, typischerweise werden zu versteckende Daten aber in „Little Endian“ kodiert. Es ist daher nötig, die Daten bei der Verwendung der geschilderten Steganografie fehlerfrei ins jeweils andere Format zu überführen. „Little Endian“ ordnet Bits nach ihrer Wertigkeit absteigend von links nach rechts an, „Big Endian“ aufsteigend.
3. Es stehen nur sieben Bits und damit kein vollständiges Byte im Datenfeld eines „Program Change“-Ereignisses für das Verstecken von Daten zur Verfügung, weil der Wertebereich für die das Instrument angebende Programmnummer nur von 0x00 bis 0x7F reicht. Es ist daher sinnvoll, wie in [John, 2004] Bytes in zwei Hälften aufzuteilen, um diese dann einzeln zu verstecken.

3 Methoden

In diesem Kapitel geht es um die praktische Umsetzung der im Grundlagenteil ab Seite 3 veranschaulichten Verfahren und Technologien in einem Softwareprojekt namens „StegaCrypto“. Zuerst wird das Entwicklungskonzept mit den an die Anwendung gestellten Anforderungen vorgestellt. Darauf folgend wird auf die Gestaltung der Bedienoberfläche sowie die sich daraus ableitende Struktur der Software eingegangen. Abschließend werden dann die konkreten Implementierungen der einzelnen Verfahren anhand der Konfiguration der Steuerelemente erläutert. Die gesamte Anwendung wurde in der Programmiersprache C# geschrieben. Als Entwicklungsumgebung wurde dabei die Community-Edition von Visual Studio verwendet. Des Weiteren wird hier darauf hingewiesen, dass dieser Teil der Masterarbeit den Entwicklungsprozess sowie den allgemeinen Ablauf und die grobe Struktur der einzelnen Softwarebestandteile beschreibt. Für wirklich ausführliche Detailinformationen zum konkreten Programmcode ist dieser unbedingt auf der am Ende dieser Arbeit beiliegenden CD zu inspizieren. Der Quellcode wurde großzügig mit Kommentaren versehen, um seine Verständlichkeit für Interessierte zu gewährleisten. Eine ausführliche Behandlung des Codes im Methodenteil hätte aufgrund seines gigantischen Umfangs leider den Rahmen gesprengt.

3.1 Entwicklungskonzept und Softwareanforderungen

Durch die Realisierung von „StegaCrypto“ sollen die auf Seite 1 erläuterten Ziele erfüllt werden. Um dies zu gewährleisten, wird die dort noch sehr abstrakt formulierte Zielstellung nun auf die Implementierung der folgenden konkreten Anwendungsbestandteile heruntergebrochen:

1. Ver- und Entschlüsselung von Texten durch AES-256
2. Ver- und Entschlüsselung von Dateien mittels AES-256
3. Verstecken und Extrahieren von Texten mittels RGB-Steganografie
4. Verstecken und Extrahieren von Dateien mittels RGB-Steganografie
5. Verstecken und Extrahieren von Texten mittels MIDI-Steganografie auf der Basis von „Program Change“-Ereignissen
6. Verstecken und Extrahieren von Dateien mittels MIDI-Steganografie auf der Basis von „Program Change“-Ereignissen
7. Kombination der in 1. und 2. aufgelisteten Verfahren mit einem Passwortschutz auf Basis der Schlüsselableitungsfunktion PBKDF2
8. Verknüpfung der in 1.-7. aufgelisteten Verfahren und einfache Steuerung der Anwendung durch eine intuitive Benutzeroberfläche

3.2 Design der Bedienoberfläche

Da die einfache Bedienung von „StegaCrypto“ ein zentraler Bestandteil der Zielstellung ist, wird zuerst die grafische Oberfläche erstellt. Diese UI soll dann das Fundament für den Aufbau des gesamten Projektes und die Implementierung der unterschiedlichen Verfahren bilden. Damit das Design am Ende nicht das Ziel der Intuitivität verfehlt, werden vorab Intuitivitätskriterien formuliert. Jedes Kriterium wird zur besseren Verständlichkeit mit einem passenden Gegenbeispiel zu „StegaCrypto“ versehen.

1. Keine Redundanz:
   Das Design enthält keine Redundanz, also nur für die Bedienung und das Verständnis absolut notwendige Steuerelemente und Informationen. Gegenbeispiel: Es ist unnötig, die Länge der verschlüsselten Daten an den Menschen zu übermitteln, da er dafür keine Verwendung hat.
2. Vollständigkeit:
   Das Design ist vollständig, weil alle für den Menschen wichtigen Informationen darin übermittelt werden. Gegenbeispiel: Eine extrahierte und anschließend entschlüsselte Datei wird an einem dem Menschen unbekannten Ort abgelegt.
3. Logische Anordnung:
   Die Anordnung der Steuerelemente (Buttons, Textboxen etc.) passt zu deren Beziehung zueinander und ist damit logisch. Gegenbeispiel: Ein Button für die Dateiauswahl befindet sich ganz oben im Interface, aber der Pfad der ausgewählten Datei wird in einer Textbox ganz unten angezeigt.
4. Übersichtlichkeit:
   Das Design ist übersichtlich, weil alle integrierten Funktionen schnell und leicht gefunden werden können. Gegenbeispiel: Die ansteuerbaren Funktionen können nicht aus einer Übersicht ausgewählt werden, sondern es muss stets ihr kompletter Name in eine Textbox eingegeben werden. Der Mensch muss so mühsam den Funktionsumfang austesten.
5. Pragmatismus:
   Das Design ist pragmatisch, weil alle integrierten Funktionen mit nur wenigen Steuerbefehlen abgerufen werden können. Gegenbeispiel: Für eine Kombination von Krypto- mit Steganografie muss sich der Mensch durch zwei separate Fenster mit Buttons durchklicken, statt beide aus einer Übersicht in einem Fenster wählen zu können.
6. Prinzipientreue:
   Die Anordnung, Optik und Funktionalität der Elemente steht nicht im Widerspruch zu als Allgemeinwissen geltenden Standards und Prinzipien. Gegenbeispiel: Die Schaltflächen für Minimieren, Maximieren und Schließen werden nicht in genau dieser Reihenfolge angeordnet.

Zunächst werden die essentiellen Ein- und Ausgabedaten zusammen mit den dafür erforderlichen Steuerelementen in Bezug auf die ersten sechs Punkte im Entwicklungskonzept ermittelt. Dabei fällt auf, dass sich die darin aufgeführten Verfahren in zwei Gruppen aufteilen lassen:

Tabelle 3.1: Gruppierung der zu implementierenden Verfahren nach benötigten Steuerelementen

Verfahren	**Steuerelemente**
Textbasierte Verfahren: 1. AES-256 3. RGB-Steganografie 5. MIDI-Steganografie	Textbox mit Button für Auswahl der Trägerdatei Textbox für Nachrichteneingabe oder -anzeige Button für Verschlüsseln und Verstecken Button für Extrahieren und Entschlüsseln Textbox für das zu verwendende Passwort
Dateibasierte Verfahren: 2. AES-256 4. RGB-Steganografie 6. MIDI-Steganografie	Textbox mit Button für Auswahl der Zieldatei Textbox mit Button für Auswahl der Zieldatei Button für Verschlüsseln und Verstecken Button für Extrahieren und Entschlüsseln Textbox für das zu verwendende Passwort

Anhand Tabelle 3.1 wird deutlich, dass sich alle text- und dateibasierten Verfahren jeweils eine eigene Oberfläche teilen könnten. Darüber hinaus muss natürlich noch eine Auswahl der gewünschten Kombination aus Krypto- und Steganografie durch den Menschen erfolgen. Dazu könnte man die verfügbaren krypto- und steganografischen Funktionalitäten jeweils in einer sogenannten ListBox aufführen. Die so präsentierten Technologien können dann einfach per Mausklick selektiert werden. Auf Grundlage der bis hierhin geschilderten Überlegungen sowie der genannten Intuitivitätskriterien wurde schließlich eine Bedienoberfläche mit zwei unterschiedlichen Reitern entwickelt, wobei der erste Reiter eine Schnittstelle für textbasierte und der zweite eine für dateibasierte Verfahren bereitstellt. In den Abbildungen 3.1 auf Seite 36 und 3.2 auf Seite 37 ist dieses finale UI-Design dargestellt. Da vorerst nur für einen Verschlüsselungsalgorithmus die Implementierung geplant ist, mag eine Kryptografieauswahl zunächst redundant erscheinen. Der Prototyp soll jedoch in Zukunft einfach um weitere Verfahren erweitert werden können und deswegen wurde bereits eine ListBox für die Kryptografie angelegt, welche später mit weiteren Optionen befüllt werden kann.

Bedienung und Aufbau des Reiters für textbasierte Verfahren
Beim Klick auf „Encrypt“ wird die eingegebene Nachricht mit der ausgewählten Kryptografie unter Nutzung des angegebenen Passwortes verschlüsselt. Die so entstandene Chiffre wird anschließend in der ausgewählten Trägerdatei mit der selektierten steganografischen Methode versteckt.

Beim Klick auf „Decrypt“ wird eine in der Trägerdatei versteckte Chiffre mit dem angegebenen steganografischen Verfahren extrahiert und im Anschluss mit der ausgewählten Kryptografie unter Verwendung des eingetippten Passwortes zu einer lesbaren Nachricht entschlüsselt, welche im Nachrichtenfeld angezeigt wird.

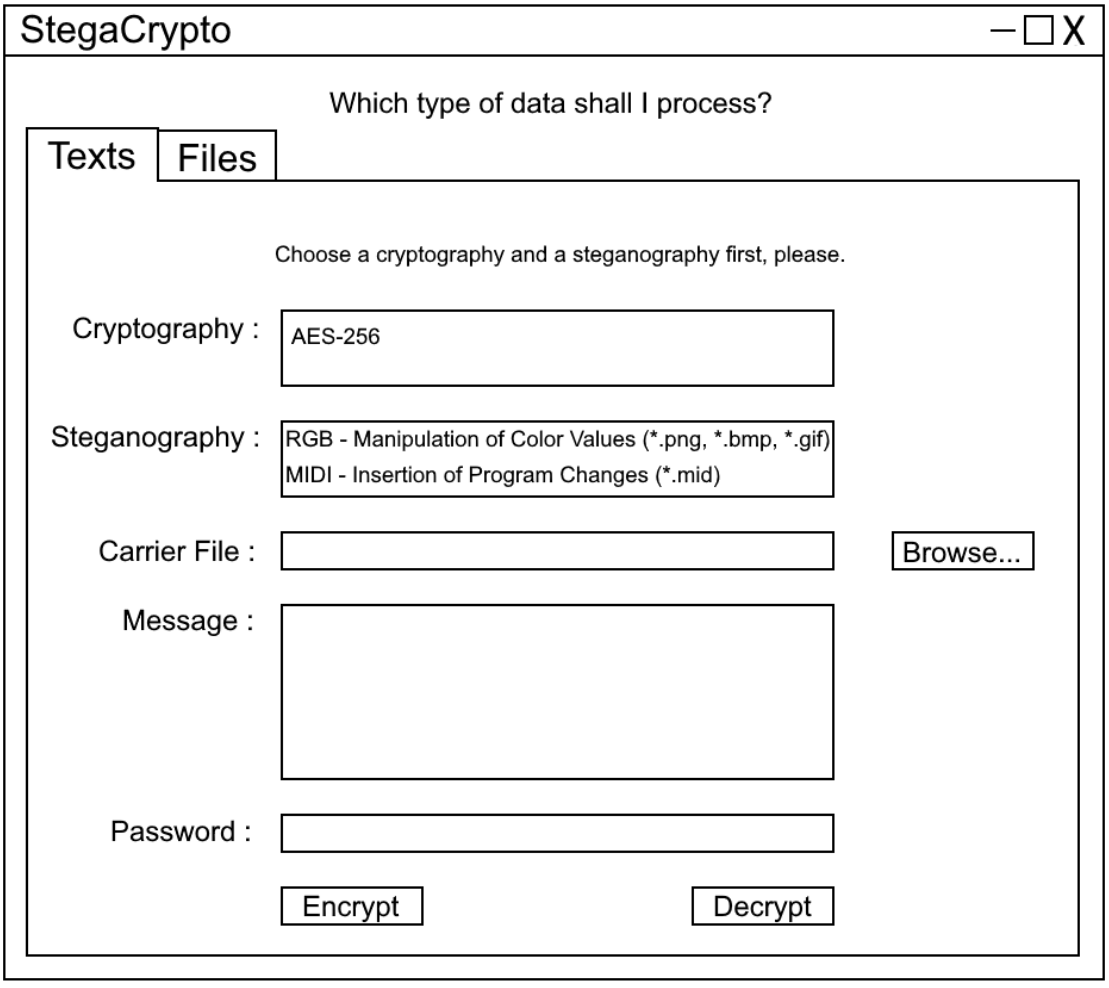


Abbildung 3.1: UI-Reiter für textbasierte Krypto- und Steganografie. Quelle: eigene Arbeit

Bedienung und Aufbau des Reiters für dateibasierte Verfahren
Beim Klick auf „Encrypt“ wird die angegebene Quelldatei mit der ausgewählten Kryptografie unter Nutzung des eingetragenen Passwortes verschlüsselt. Die so entstandene Chiffre wird dann durch die selektierte steganografische Methode in der Zieldatei versteckt.

Beim Klick auf „Decrypt“ wird eine in der Quelldatei versteckte Chiffre mit der ausgewählten steganografischen Methode extrahiert und im Anschluss durch die selektierte Kryptografie unter Nutzung des eingetragenen Passwortes zu einer lesbaren Datei entschlüsselt, welche dann die angegebene Zieldatei überschreibt.

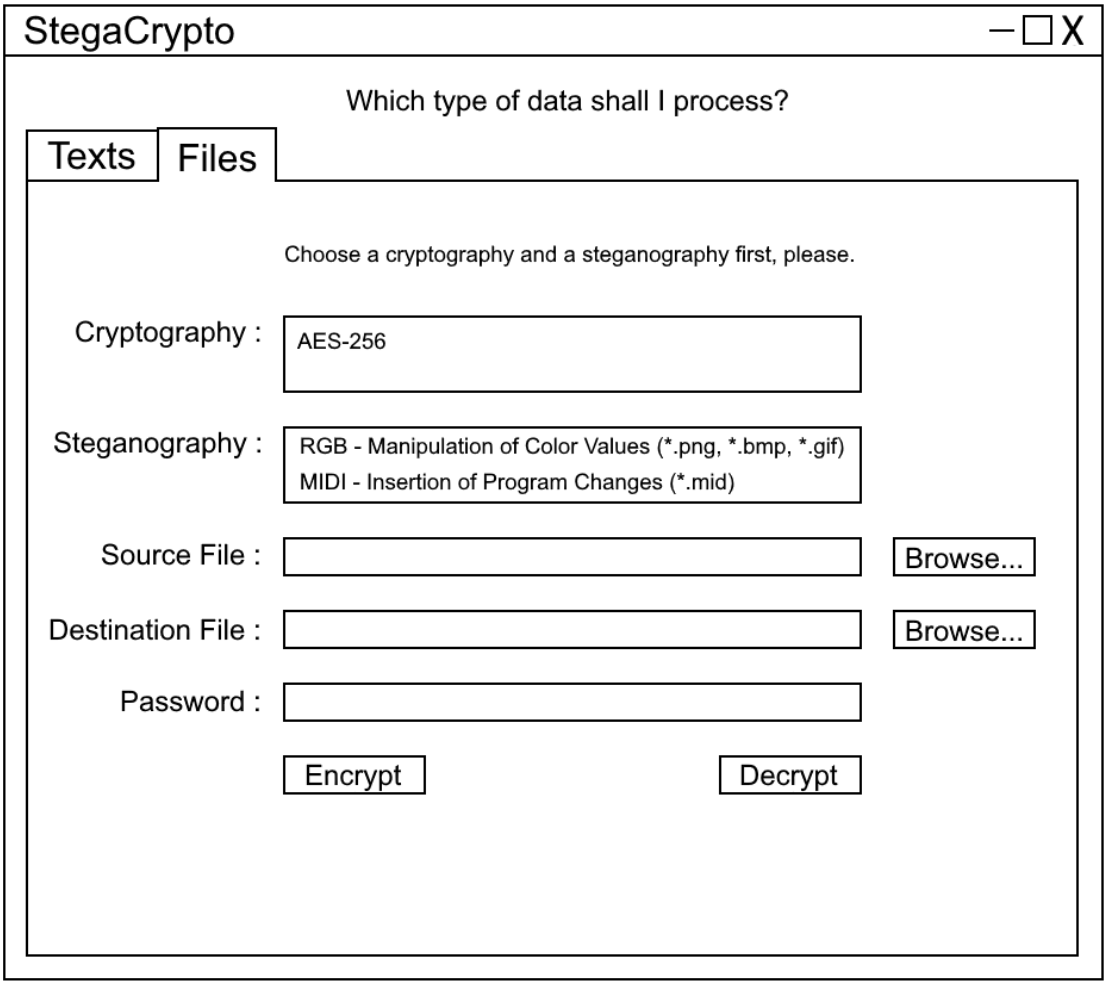


Abbildung 3.2: UI-Reiter für dateibasierte Krypto- und Steganografie. Quelle: eigene Arbeit

3.3 Konfiguration des „Encrypt“-Buttons im Reiter „Texts“

Dieser Abschnitt bietet einen groben Überblick über die Funktionsweise des genannten Buttons und die Implementierung textbasierter Verschlüsselung in Kombination mit Steganografie. Der „Encrypt“-Button prüft zunächst durch If-Anweisungen, welche Kryptografie im Reiter „Texts“ ausgewählt ist und führt deren Programmcode aus. Anschließend ermittelt er mit weiteren If-Anweisungen die selektierte Steganografie und übergibt dem dazugehörigen Code das Ergebnis der Verschlüsselung.

3.3.1 AES-256 kombiniert mit PBKDF2

Wurde AES-256 als textbasierte Verschlüsselung ausgewählt, so wird zunächst ein 16 Byte umfassender Salt mit einer kryptografischen Zufallsfunktion erzeugt. Dieser wird dann zusammen mit dem im Reiter „Texts“ hinterlegten Passwort an eine Klasse übergeben, die mittels PBKDF2 unter Nutzung der übergebenen Daten über 10.000 Iterationen einen Schlüssel berechnet. Anschließend wird dieser Schlüssel mit einer Länge von 256 Bits neben weiteren Parametern, wie der Blockgröße von 128 Bits, als eine Eigenschaft der bevorstehenden Verschlüsselung vereinbart. Auch der für den verwendeten CBC-Modus erforderliche Initialisierungsvektor mit einer Länge von 16 Bytes wird durch die bereits eingesetzte Zufallsfunktion generiert. Nach all diesen Vorbereitungen wird die eingegebene Nachricht schließlich zu einem Byte-Array beziehungsweise einem Bitstrom verschlüsselt. An das Ende dieser Binärdaten werden abschließend der Salt und der Initialisierungsvektor als weitere Byte-Arrays in genau dieser Reihenfolge angehängt. Das Ergebnis dieses Codes sind also Binärdaten, welche in den letzten 16 Bytes den IV, in den vorletzten 16 Bytes den Salt und in allen Bytes davor die verschlüsselte Nachricht beinhalten.

3.3.2 RGB-Steganografie

Wurde „RGB - Manipulation of Color Values“ innerhalb der Steganografieauswahl im Reiter „Texts“ selektiert, so wird zunächst die Pixelgrafik aus der angegebenen Trägerdatei geladen. Dies geschieht über den Aufruf einer externen Funktion, siehe Abschnitt 3.7.2 auf Seite 51. Jedes Pixel kann dabei wie in Abschnitt 2.3.2 erläutert über X- und Y-Koordinaten adressiert werden und verfügt über einen ihm zugeordneten ARGB-Wert. Für die Erweiterung von RGB-Werten um einen Alphakanal mag die Bezeichnung „RGBA“ vielleicht geläufiger sein. Die zum Laden der Pixelgrafik verwendete Klasse „Bitmap“ stellt den Alpha-Wert jedoch stets vor den RGB-Wert, weswegen die Bezeichnung „ARGB“ für dieses Beispiel deutlich schlüssiger und sinnvoller ist. Nachdem die Pixelgrafik als Bitmap-Objekt erfasst wurde, wird durch eine Zählschleife das gesamte aus der Verschlüsselung resultierende Byte-Array durchlaufen. Außerdem werden die Pixelkoordinaten „x“ und

„y“ als $(0;0)$ vereinbart. Diese Koordinaten adressieren das Pixel in der oberen linken Bildecke. Für das aktuelle Byte werden dabei stets die RGB-Informationen des Pixels mit den aktuellen Koordinaten ausgelesen. Anschließend wird der Blauwert des aktuell adressierten Pixels auf den Wert des aktuellen Bytes aus dem Byte-Array geändert. Bevor das Prozedere für das nächste Byte wiederholt wird, werden die Pixelkoordinaten aktualisiert. Solange noch nicht die fünfte Bildzeile von unten erreicht ist, wird die Y-Koordinate um 5 inkrementiert. Andernfalls wird der Y-Wert auf Null gesetzt und die X-Koordinate um Fünf erhöht, um das bisherige Verfahren fünf Spalten später am Beginn einer neuen Bildspalte fortzusetzen. Zusammengefasst wird also das Byte-Array aus der Verschlüsselung byteweise in den Blauwerten der Bildpixel versteckt, wobei die Pixelgrafik von oben links bis unten rechts spaltenweise durchlaufen wird. Das Koordinatenintervall von Fünf wurde gewählt, um durch eine stärkere Datenverteilung eine möglichst unauffällige optische Verzerrung zu erreichen. Damit die so versteckten Daten später auch wieder erfolgreich extrahiert werden können, wird außerdem die Länge der versteckten Daten steganografisch im Bild hinterlegt. Dazu wird die Länge des versteckten Arrays in Bytes zunächst als „Int32“ erfasst. Dabei handelt es sich um eine durch vier Bytes kodierte Ganzzahl. Diese wird nun als vierstelliges Byte-Array interpretiert, dessen Bytes von links nach rechts als Blauwerte der letzten vier Bildpixel in der untersten Zeile versteckt werden. Auf Basis der so entstandenen Pixelgrafik mit versteckten verschlüsselten Informationen wird nun eine neue Bilddatei erzeugt, welche die angegebene Trägerdatei überschreibt.

3.3.3 MIDI-Steganografie

Wurde „MIDI - Insertion of Program Changes“ innerhalb der Steganografieauswahl im Reiter „Texts“ selektiert, so wird zunächst die Trägerdatei in ein Byte-Array umgewandelt, für Details siehe Abschnitt 3.7.3 auf Seite 51. Darauf folgend werden die Bytes des aus der Verschlüsselung stammenden Byte-Arrays jeweils in zwei Hälften aufgespalten. Die daraus resultierenden 4-Bit-Sequenzen werden dann in einem neuen Byte-Array hintereinander als separate Bytes abgelegt, von denen die ersten vier Bits logischerweise stets auf Null gesetzt sind. So ergibt sich aus den Bytes „*AB CD*“ beispielsweise „0*A* 0*B* 0*C* 0*D*“. Anschließend prüft „StegaCrypto“ durch eine If-Anweisung, ob an den dafür vorgesehenen Stellen der „MIDI File Header“ mit dem Standardlängenwert von sechs Bytes und der erste „MIDI Track Header“ detektiert werden können. Nur falls dieser Test positiv ausfällt, wird der weitere Code der MIDI-Steganografie ausgeführt. In diesem Fall besteht der nächste Schritt darin, dass die Position des ersten „Program Change“-Ereignisses ermittelt wird. Danach wird die Trägerdatei vor dessen Start in zwei Hälften geteilt, welche jeweils als Byte-Array zwischengespeichert werden. Anschließend wird das Byte-Array für die einzufügenden „Program Change“-Ereignisse erstellt, welche jeweils ein halbes Byte beziehungsweise vier Bit verstecken können. Es werden also zwei solcher Ereignisse pro zu versteckendem Byte benötigt. Weiterhin umfasst jeder „Program Change“ insgesamt drei Bytes. Um die Länge des zwischen Dateianfang und -ende einzuschiebenden Byte-Arrays zu ermitteln, wird daher die Länge des aus der

Verschlüsselung stammenden Byte-Arrays erst verdoppelt und anschließend verdreifacht, was insgesamt einer Versechsfachung entspricht. Als nächstes wird das Byte-Array für die einzufügenden Daten mit entsprechenden Ereignissen befüllt. Dazu werden die Werte des Arrays für jede 3-Byte-Sequenz bzw. jeden „Program Change“ nach folgendem Schema festgelegt:

1. Setze das erste Byte, also die Deltazeit, auf den Wert 0x00. Dies hat wie am Ende von Abschnitt 2.3.3 erläutert den Sinn, dass zwischen den Ereignissen keine Töne erzeugt werden und die eingefügten Daten somit keinen Einfluss auf das Klangmuster bei Wiedergabe der Datei haben.
2. Setze das zweite Byte auf den Wert 0xC0. „C“ signalisiert, dass es sich um ein „Program Change“-Ereignis handelt. Die zweite Bytehälfte „0“ gibt an, dass dieses Ereignis auf Kanal Null stattfindet. Hier könnten genauso andere Kanäle gewählt werden, es spielt für die Funktionalität des Verfahrens keine Rolle.
3. Verstecke eine der zuvor durch die Byteteilung generierten 4-Bit-Sequenzen als Wert für das dritte Byte, um so Daten als Programmnummer im aktuellen Ereignis zu verstecken.

Damit die Daten im 3. Schritt auch vollständig und in der richtigen Reihenfolge versteckt werden, wird vor all diesen Wertzuweisungen eine Indexvariable mit dem Wert Null angelegt, welche nach jeder Ausführung des dritten Schrittes um Eins inkrementiert wird. Diese Indexvariable ist der Index des zu versteckenden halbierten Bytes im entsprechenden Array. Damit die nun versteckten Daten später auch wieder erfolgreich extrahiert werden können, wird außerdem die Anzahl der nach dem obigen Schema eingefügten Ereignisse steganografisch in der Trägerdatei festgehalten. Dazu wird zunächst die Länge des Byte-Arrays mit den einzufügenden Daten durch Drei dividiert, da wie bereits angemerkt jeder „Program Change“ drei Bytes umfasst. Die so berechnete Ereignisanzahl wird als „Int32“ gespeichert, also als eine durch vier Bytes kodierte Ganzzahl. Diese vier Bytes werden dann ebenfalls nach dem obigen Schema in insgesamt acht „Program Changes“ versteckt. Damit die MIDI-Datei gültig bzw. ausführbar bleibt, wird noch die Länge des ersten „Track Headers“ um die Länge aller eingefügten Ereignisse erhöht. Da diese Längenkodierung in MIDI-Dateien in „Big Endian“ erfolgt, also vom MSB zum LSB, muss die Byte-Reihenfolge des als vierstelliges Byte-Array interpretierten Längenwertes zunächst umgekehrt werden. Nach dieser Umwandlung werden die längenkodierenden Bytes in das Längenfeld des „Track Headers“ geschrieben. Abschließend werden die Byte-Arrays für Dateianfang, Anzahl eingefügter „Program Changes“, eingefügte „Program Changes“ und Dateiende in genau dieser Reihenfolge zu einem gemeinsamen Byte-Array verknüpft. Die Bytes des neuen Arrays werden dann in eine neue Datei geschrieben, welche die ausgewählte Trägerdatei ersetzt.

3.4 Konfiguration des „Decrypt“-Buttons im Reiter „Texts“

Dieser Abschnitt bietet einen groben Überblick über die Funktionsweise des genannten Buttons und die Implementierung von steganografischer Extraktion in Kombination mit textbasierter Entschlüsselung. Der „Decrypt“-Button prüft zunächst durch If-Anweisungen, welche Steganografie im Reiter „Texts“ ausgewählt ist und führt deren zugehörigen Programmcode aus. Anschließend ermittelt er mit weiteren If-Anweisungen die ausgewählte Kryptografie und übergibt dem dazugehörigen Code das Ergebnis der Extraktion.

3.4.1 RGB-Steganografie

Wurde im Reiter „Texts“ als Steganografie „RGB - Manipulation of Color Values“ ausgewählt, so wird zunächst die Pixelgrafik aus der angegebenen Trägerdatei in ein dafür angelegtes Objekt der „Bitmap“-Klasse geladen. Dieser Ladevorgang wird durch den Aufruf einer externen Funktion realisiert, siehe Abschnitt 3.7.2 auf Seite 51. Für eine erfolgreiche Extraktion muss zunächst die Länge des zuvor versteckten Byte-Arrays ermittelt werden. Dazu wird ein vierstelliges Byte-Array mit den Blauwerten der vier letzten Bildpixel in der untersten Zeile von links nach rechts befüllt. Anschließend wird dieses Byte-Array als „Int32“ interpretiert, also als eine durch die vier extrahierten Bytes kodierte Ganzzahl. Diese Zahl entspricht der Länge des versteckten verschlüsselten Byte-Arrays und damit der Anzahl der Bildpixel, die Inhalte des Arrays verstecken und daher bei der kommenden Extraktion einbezogen werden müssen. Die Länge eines leeren Byte-Arrays, das bereits vor dem Aufruf des Codes für die RGB-steganografische Extraktion initialisiert wurde, wird nun auf die soeben ermittelte Länge der versteckten Daten festgelegt. Außerdem werden die Pixelkoordinaten „x“ und „y“ als $(0;0)$ vereinbart. Diese Koordinaten adressieren das Pixel in der oberen linken Bildecke. Durch eine Zählschleife wird nun das gesamte leere Byte-Array mit der zuvor bestimmten Länge durchlaufen. Für das aktuelle Byte werden dabei stets die RGB-Informationen des Pixels mit den aktuellen Koordinaten ausgelesen. Im Anschluss wird der Blauwert dieses Pixels in das aktuelle Byte des Arrays geladen. Solange noch nicht die fünfte Bildzeile von unten erreicht ist, wird am Ende jedes Schleifendurchlaufs die Y-Koordinate um Fünf inkrementiert. Andernfalls wird die Y-Koordinate auf Null gesetzt, während der X-Wert um Fünf inkrementiert wird. Das Koordinatenintervall von Fünf wurde gewählt, um durch eine stärkere Datenverteilung eine möglichst unauffällige optische Verzerrung zu erreichen. Zusammengefasst wird also das Byte-Array mit der aus den letzten vier Bildpixeln der untersten Spalte ausgelesenen Länge byteweise mit den Blauwerten der Bildpixel befüllt, wobei die Pixelgrafik von oben links bis unten rechts spaltenweise durchlaufen wird. Das auf diese Weise mit Daten befüllte Byte-Array ist das finale Ergebnis der RGB-steganografischen Extraktion.

3.4.2 MIDI-Steganografie

Wurde „MIDI - Insertion of Program Changes“ innerhalb der Steganografieauswahl im Reiter „Texts“ selektiert, so wird zunächst die Trägerdatei in ein Byte-Array umgewandelt, für Details siehe Abschnitt 3.7.3 auf Seite 51. „StegaCrypto“ prüft durch eine If-Anweisung, ob an den dafür vorgesehenen Stellen der „MIDI File Header“ mit dem Standardlängenwert von sechs Bytes und der erste „MIDI Track Header“ detektiert werden können. Alle im Folgenden beschriebenen Aktionen erfolgen nur, wenn das Ergebnis dieser Prüfung positiv ist. Im nächsten Schritt werden dann die Startposition sowie die Länge der für die Steganografie eingeschobenen „Program Change“-Ereigniskette ermittelt. Dazu wird die als Byte-Array repräsentierte Trägerdatei mit einer Zählschleife nach dem ersten Wert 0xC0 durchsucht, um den ersten „Program Change“ zu finden. Zu dieser Position werden 23 Bytes addiert, da die ersten acht „Program Changes“ (24 Bytes) lediglich die Länge der Ereigniskette kodieren und der erste 0xC0-Wert an der zweiten Stelle des ersten „Program Changes“ steht (im Byte davor steht die Deltazeit). Auf diese Weise wird also die Startposition der zu extrahierenden Daten ermittelt. Um anschließend noch die Länge dieser Ereigniskette zu bestimmen, werden die Programmnummern der ersten acht „Program Changes“ in ein separates Array geschrieben. Darauf folgend werden diese Programmnummern abwechselnd als linke und rechte Bytehälften interpretiert und so paarweise zu ganzen Bytes eines neuen Arrays zusammengesetzt. Das Resultat ist ein vierstelliges Byte-Array, welches dann als „Int32“, also eine durch vier Bytes kodierte Ganzzahl interpretiert wird. Diese Ganzzahl entspricht der Anzahl der zu extrahierenden „Program Changes“. Um nun die Länge der auszulesenden Ereigniskette in Bytes zu bestimmen, wird die soeben festgestellte Anzahl mit Drei multipliziert, da jedes „Program Change“-Ereignis drei Bytes umfasst. Nachdem Startposition und Länge der Ereigniskette bekannt sind, wird diese Reihe von „Program Changes“ vom Rest der Datei isoliert und in ein eigenes Byte-Array geschrieben. Darauf folgend werden unter Nutzung einer Schleife alle Programmnummern dieser Ereignisse in ein weiteres Byte-Array geschrieben. Diese Daten sind die halbierten Bytes der versteckten Inhalte, weswegen sie im Anschluss abwechselnd als linke und rechte Bytehälften interpretiert und so in einem neuen Array zu ganzen Bytes paarweise zusammengesetzt werden. Das Array mit den zusammengesetzten Bytes ist das finale Ergebnis der MIDI-steganografischen Extraktion.

3.4.3 AES-256 kombiniert mit PBKDF2

Wurde AES-256 als textbasierte Entschlüsselung ausgewählt, so werden die letzten 16 Bytes des aus der Steganografie stammenden Byte-Arrays als Initialisierungsvektor für die bevorstehende Entschlüsselung im CBC-Modus bestimmt. Der Salt für die Dechiffrierung entspricht wiederum den vorletzten 16 Bytes. Nachdem die letzten 32 Bytes für den Salt und den IV ausgelesen wurden, wird das Array um 32 Bytes gekürzt. Nach der Kürzung steht nur noch die Chiffre im Array. Mit Hilfe des im Reiter „Texts“ eingegebenen Passwortes und des ausgelesenen Salts wird mittels PBKDF2 über 10.000 Iterationen der Schlüssel für die Entschlüsselung berechnet. Anschließend wird dieser Schlüssel mit einer Länge von 256 Bits neben weiteren Parametern, wie der Blockgröße von 128 Bits, als eine Eigenschaft der bevorstehenden Verschlüsselung vereinbart. Nach all diesen Vorbereitungen wird die Chiffre schließlich zu einem „String“, also einer Zeichenkette, entschlüsselt. Das finale Ergebnis dieses Entschlüsselungsprozesses ist also ein Text, welcher dann in der Textbox hinter „Message :“ als Resultat angezeigt wird.

3.5 Konfiguration des „Encrypt-Buttons“ im Reiter „Files“

Dieser Abschnitt bietet einen groben Überblick über die Funktionsweise des genannten Buttons und die Implementierung dateibasierter Verschlüsselung in Kombination mit Steganografie. Der „Encrypt“-Button prüft zunächst durch If-Anweisungen, welche Kryptografie im Reiter „Files“ ausgewählt ist und führt deren Programmcode aus. Anschließend ermittelt er mit weiteren If-Anweisungen die ausgewählte Steganografie und übergibt dem dazugehörigen Code das Ergebnis der Verschlüsselung.

3.5.1 AES-256 kombiniert mit PBKDF2

Wurde AES-256 als dateibasierte Verschlüsselung ausgewählt, so wird zunächst ein 16 Byte umfassender Salt mit einer kryptografischen Zufallsfunktion erzeugt. Dieser wird dann zusammen mit dem im Reiter „Files“ hinterlegten Passwort an eine Klasse übergeben, die mittels PBKDF2 unter Nutzung der übergebenen Daten über 10.000 Iterationen einen Schlüssel berechnet. Anschließend wird dieser Schlüssel mit einer Länge von 256 Bits neben weiteren Parametern, wie der Blockgröße von 128 Bits, als eine Eigenschaft der bevorstehenden Verschlüsselung vereinbart. Auch der für den verwendeten CBC-Modus erforderliche Initialisierungsvektor mit einer Länge von 16 Bytes wird durch die bereits eingesetzte Zufallsfunktion generiert. Nach all diesen Vorbereitungen wird die ausgewählte Quelldatei schließlich zu einem Byte-Array beziehungsweise einem Bitstrom verschlüsselt. Dafür wird die Quelldatei zunächst ebenfalls als Byte-Array interpretiert, siehe dazu Abschnitt 3.7.3 auf Seite 51. An das Ende der aus der Verschlüsselung resultierenden Binärdaten werden abschließend der Salt und der Initialisierungsvektor als weitere Byte-Arrays in genau dieser Reihenfolge angehängt. Das Ergebnis dieses Codes sind also Binärdaten, welche in den letzten 16 Bytes den IV, in den vorletzten 16 Bytes den Salt und in allen Bytes davor die verschlüsselte Quelldatei beinhalten.

3.5.2 RGB-Steganografie

Wurde „RGB - Manipulation of Color Values“ innerhalb der Steganografieauswahl im Reiter „Files“ selektiert, so wird zunächst die Pixelgrafik aus der angegebenen Zieldatei geladen. Dies geschieht über den Aufruf einer externen Funktion, siehe Abschnitt 3.7.2 auf Seite 51. Jedes Pixel kann dabei wie in Abschnitt 2.3.2 erläutert über X- und Y-Koordinaten adressiert werden und verfügt über einen ihm zugeordneten ARGB-Wert. Für die Erweiterung von RGB-Werten um einen Alphakanal mag die Bezeichnung „RGBA“ vielleicht geläufiger sein. Die zum Laden der Pixelgrafik verwendete Klasse „Bitmap“ stellt den Alpha-Wert jedoch stets vor den RGB-Wert, weswegen die Bezeichnung „ARGB“ für dieses Beispiel deutlich schlüssiger und sinnvoller ist. Nachdem die Pixelgrafik als Bitmap-Objekt erfasst wurde, wird durch eine Zählschleife das gesamte aus der Verschlüsselung

resultierende Byte-Array durchlaufen. Außerdem werden die Pixelkoordinaten „x“ und „y“ als $(0;0)$ vereinbart. Diese Koordinaten adressieren das Pixel in der oberen linken Bildecke. Für das aktuelle Byte werden dabei stets die RGB-Informationen des Pixels mit den aktuellen Koordinaten ausgelesen. Anschließend wird der Blauwert des aktuell adressierten Pixels auf den Wert des aktuellen Bytes aus dem Byte-Array geändert. Bevor das Prozedere für das nächste Byte wiederholt wird, werden die Pixelkoordinaten aktualisiert. Solange noch nicht die fünfte Bildzeile von unten erreicht ist, wird die Y-Koordinate um Fünf inkrementiert. Andernfalls wird der Y-Wert auf Null gesetzt und die X-Koordinate um Fünf erhöht, um das bisherige Verfahren am Beginn einer anderen Bildspalte fortzusetzen. Das Koordinatenintervall von Fünf wurde gewählt, um durch eine stärkere Datenverteilung eine möglichst unauffällige optische Verzerrung zu erreichen. Zusammengefasst wird also das Byte-Array aus der Verschlüsselung byteweise in den Blauwerten der Bildpixel versteckt, wobei die Pixelgrafik von oben links bis unten rechts spaltenweise durchlaufen wird. Damit die so versteckten Daten später auch wieder erfolgreich extrahiert werden können, wird außerdem die Länge der versteckten Daten steganografisch im Bild hinterlegt. Dazu wird die Länge des versteckten Arrays in Bytes zunächst als „Int32“ erfasst. Dabei handelt es sich um eine durch vier Bytes kodierte Ganzzahl. Diese wird nun als vierstelliges Byte-Array interpretiert, dessen Bytes von links nach rechts als Blauwerte der letzten vier Bildpixel in der untersten Zeile versteckt werden. Auf Basis der so entstandenen Pixelgrafik mit versteckten verschlüsselten Informationen wird nun eine neue Bilddatei erzeugt, welche die angegebene Zieldatei überschreibt.

3.5.3 MIDI-Steganografie

Wurde „MIDI - Insertion of Program Changes“ innerhalb der Steganografieauswahl im Reiter „Files“ selektiert, so wird zunächst die Zieldatei in ein Byte-Array umgewandelt, für Details siehe Abschnitt 3.7.3 auf Seite 51. Darauf folgend werden die Bytes des aus der Verschlüsselung stammenden Byte-Arrays jeweils in zwei Hälften aufgespalten. Die daraus resultierenden 4-Bit-Sequenzen werden dann in einem neuen Byte-Array hintereinander als separate Bytes abgelegt, von denen die ersten vier Bits logischerweise stets auf Null gesetzt sind. So ergibt sich aus den Bytes „*AB CD*“ beispielsweise „0*A* 0*B* 0*C* 0*D*“. Anschließend prüft „StegaCrypto“ durch eine If-Anweisung, ob an den dafür vorgesehenen Stellen der „MIDI File Header“ mit dem Standardlängenwert von sechs Bytes und der erste „MIDI Track Header“ detektiert werden können. Nur falls dieser Test positiv ausfällt, wird der weitere Code der MIDI-Steganografie ausgeführt. In diesem Fall besteht der nächste Schritt darin, dass die Position des ersten „Program Change“-Ereignisses ermittelt wird. Danach wird die Zieldatei vor dessen Start in zwei Hälften geteilt, welche jeweils als Byte-Array zwischengespeichert werden. Anschließend wird das Byte-Array für die einzufügenden „Program Change“-Ereignisse erstellt, welche jeweils ein halbes Byte beziehungsweise vier Bit verstecken können. Es werden also zwei solcher Ereignisse pro zu versteckendem Byte benötigt. Weiterhin umfasst jeder „Program Change“ insgesamt drei Bytes. Um die Länge des zwischen Dateianfang und

-ende einzuschiebenden Byte-Arrays zu ermitteln, wird daher die Länge des aus der Verschlüsselung stammenden Byte-Arrays erst verdoppelt und anschließend verdreifacht, was insgesamt einer Versechsfachung entspricht. Als nächstes wird das Byte-Array für die einzufügenden Daten mit entsprechenden Ereignissen befüllt. Dazu werden die Werte des Arrays für jede 3-Byte-Sequenz bzw. jeden „Program Change“ nach folgendem Schema festgelegt:

1. Setze das erste Byte, also die Deltazeit, auf den Wert 0x00. Dies hat wie am Ende von Abschnitt 2.3.3 erläutert den Sinn, dass zwischen den Ereignissen keine Töne erzeugt werden und die eingefügten Daten somit keinen Einfluss auf das Klangmuster bei Wiedergabe der Datei haben.
2. Setze das zweite Byte auf den Wert 0xC0. „C“ signalisiert, dass es sich um ein „Program Change“-Ereignis handelt. Die zweite Bytehälfte „0“ gibt an, dass dieses Ereignis auf Kanal Null stattfindet. Hier könnten genauso andere Kanäle gewählt werden, es spielt für die Funktionalität des Verfahrens keine Rolle.
3. Verstecke eine der zuvor durch die Byteteilung generierten 4-Bit-Sequenzen als Wert für das dritte Byte, um so Daten als Programmnummer im aktuellen Ereignis zu verstecken.

Damit die Daten im 3. Schritt auch vollständig und in der richtigen Reihenfolge versteckt werden, wird vor all diesen Wertzuweisungen eine Indexvariable mit dem Wert Null angelegt, welche nach jeder Ausführung des dritten Schrittes um Eins inkrementiert wird. Diese Indexvariable ist der Index des zu versteckenden halbierten Bytes im entsprechenden Array. Damit die nun versteckten Daten später auch wieder erfolgreich extrahiert werden können, wird außerdem die Anzahl der nach dem obigen Schema eingefügten Ereignisse steganografisch in der Zieldatei festgehalten. Dazu wird zunächst die Länge des Byte-Arrays mit den einzufügenden Daten durch Drei dividiert, da wie bereits angemerkt jeder „Program Change“ drei Bytes umfasst. Die so berechnete Ereignisanzahl wird als „Int32“ gespeichert, also als eine durch vier Bytes kodierte Ganzzahl. Diese vier Bytes werden dann ebenfalls nach dem obigen Schema in insgesamt acht „Program Changes“ versteckt. Damit die MIDI-Datei gültig bzw. ausführbar bleibt, wird noch die Länge des ersten „Track Headers“ um die Länge aller eingefügten Ereignisse erhöht. Da diese Längenkodierung in MIDI-Dateien in „Big Endian“ erfolgt, also vom MSB zum LSB, muss die Byte-Reihenfolge des als vierstelliges Byte-Array interpretierten Längenwertes zunächst umgekehrt werden. Nach dieser Umwandlung werden die längenkodierenden Bytes in das Längenfeld des „Track Headers“ geschrieben. Abschließend werden die Byte-Arrays für Dateianfang, Anzahl eingefügter „Program Changes“, eingefügte „Program Changes“ und Dateiende in genau dieser Reihenfolge zu einem gemeinsamen Byte-Array verknüpft. Die Bytes des neuen Arrays werden dann in eine neue Datei geschrieben, welche die ausgewählte Zieldatei ersetzt.

3.6 Konfiguration des „Decrypt“-Buttons im Reiter „Files“

Dieser Abschnitt bietet einen groben Überblick über die Funktionsweise des genannten Buttons und die Implementierung von steganografischer Extraktion in Kombination mit dateibasierter Entschlüsselung. Der „Decrypt“-Button prüft zunächst durch If-Anweisungen, welche Steganografie im Reiter „Files“ ausgewählt ist und führt deren zugehörigen Programmcode aus. Anschließend ermittelt er mit weiteren If-Anweisungen die ausgewählte Kryptografie und übergibt dem dazugehörigen Code das Ergebnis der Extraktion.

3.6.1 RGB-Steganografie

Wurde im Reiter „Files“ die RGB-Steganografie ausgewählt, so wird zunächst die Pixelgrafik aus der angegebenen Quelldatei in ein dafür angelegtes Objekt der „Bitmap“-Klasse geladen. Dieser Ladevorgang wird durch den Aufruf einer externen Funktion realisiert, siehe Abschnitt 3.7.2 auf Seite 51. Für eine erfolgreiche Extraktion muss zunächst die Länge des zuvor versteckten Byte-Arrays ermittelt werden. Dazu wird ein vierstelliges Byte-Array mit den Blauwerten der vier letzten Bildpixel in der untersten Zeile von links nach rechts befüllt. Anschließend wird dieses Byte-Array als „Int32“-Wert interpretiert, also als eine durch die vier extrahierten Bytes kodierte Ganzzahl. Diese Zahl entspricht der Länge des versteckten verschlüsselten Byte-Arrays und damit der Anzahl der Bildpixel, die Inhalte des Arrays verstecken und daher bei der kommenden Extraktion einbezogen werden müssen. Die Länge eines leeren Byte-Arrays, das bereits vor dem Aufruf des Codes für die RGB-steganografische Extraktion initialisiert wurde, wird nun auf die soeben ermittelte Länge der versteckten Daten festgelegt. Außerdem werden die Pixelkoordinaten „x“ und „y“ als $(0;0)$ vereinbart. Diese Koordinaten adressieren das Pixel in der oberen linken Bildecke. Durch eine Zählschleife wird nun das gesamte leere Byte-Array mit der zuvor bestimmten Länge durchlaufen. Für das aktuelle Byte werden dabei stets die RGB-Informationen des Pixels mit den aktuellen Koordinaten ausgelesen. Im Anschluss wird der Blauwert dieses Pixels in das aktuelle Byte des Arrays geladen. Solange noch nicht die fünfte Bildzeile von unten erreicht ist, wird am Ende jedes Schleifendurchlaufs die Y-Koordinate um Fünf inkrementiert. Ist bereits die letzte Bildzeile erreicht, wird die Y-Koordinate auf Null gesetzt, während der X-Wert um Fünf inkrementiert wird. Das Koordinatenintervall von Fünf wurde gewählt, um durch eine stärkere Datenverteilung eine möglichst unauffällige optische Verzerrung zu erreichen. Zusammengefasst wird also das Byte-Array mit der aus den letzten vier Bildpixeln der untersten Zeile ausgelesenen Länge byteweise mit den Blauwerten der Bildpixel befüllt, wobei die Pixelgrafik von oben links bis unten rechts spaltenweise durchlaufen wird. Das auf diese Weise mit Daten befüllte Byte-Array ist das finale Ergebnis der RGB-steganografischen Extraktion.

3.6.2 MIDI-Steganografie

Wurde „MIDI - Insertion of Program Changes“ innerhalb der Steganografieauswahl im Reiter „Files“ selektiert, so wird zunächst die Quelldatei in ein Byte-Array umgewandelt, für Details siehe Abschnitt 3.7.3 auf Seite 51. „StegaCrypto“ prüft durch eine If-Anweisung, ob an den dafür vorgesehenen Stellen der „MIDI File Header“ mit dem Standardlängenwert von sechs Bytes und der erste „MIDI Track Header“ detektiert werden können. Alle im Folgenden beschriebenen Aktionen erfolgen nur, wenn das Ergebnis dieser Prüfung positiv ist. Im nächsten Schritt werden dann die Startposition sowie die Länge der für die Steganografie eingeschobenen „Program Change“-Ereigniskette ermittelt. Dazu wird die als Byte-Array repräsentierte Quelldatei mit einer Zählschleife nach dem ersten Wert 0xC0 durchsucht, um den ersten „Program Change“ zu finden. Zu dieser Position werden 23 Bytes addiert, da die ersten acht „Program Changes“ (24 Bytes) lediglich die Länge der Ereigniskette kodieren und der erste 0xC0-Wert an der zweiten Stelle des ersten „Program Changes“ steht (im Byte davor steht die Deltazeit). Auf diese Weise wird also die Startposition der zu extrahierenden Daten ermittelt. Um anschließend noch die Länge dieser Ereigniskette zu bestimmen, werden die Programmnummern der ersten acht „Program Changes“ in ein separates Array geschrieben. Darauf folgend werden diese Programmnummern abwechselnd als linke und rechte Bytehälften interpretiert und so paarweise zu ganzen Bytes eines neuen Arrays zusammengesetzt. Das Resultat ist ein vierstelliges Byte-Array, welches dann als „Int32“, also eine durch vier Bytes kodierte Ganzzahl interpretiert wird. Diese Ganzzahl entspricht der Anzahl der zu extrahierenden „Program Changes“. Um nun die Länge der auszulesenden Ereigniskette in Bytes zu bestimmen, wird die soeben festgestellte Anzahl mit Drei multipliziert, da jedes „Program Change“-Ereignis drei Bytes umfasst. Nachdem Startposition und Länge der Ereigniskette bekannt sind, wird diese Reihe von „Program Changes“ vom Rest der Datei isoliert und in ein eigenes Byte-Array geschrieben. Darauf folgend werden unter Nutzung einer Schleife alle Programmnummern dieser Ereignisse in ein weiteres Byte-Array geschrieben. Diese Daten sind die halbierten Bytes der versteckten Inhalte, weswegen sie im Anschluss abwechselnd als linke und rechte Bytehälften interpretiert und so in einem neuen Array zu ganzen Bytes paarweise zusammengesetzt werden. Dieses Array mit den zusammengesetzten Bytes ist das finale Ergebnis der steganografischen Extraktion.

3.6.3 AES-256

Wurde AES-256 als dateibasierte Entschlüsselung ausgewählt, so werden die letzten 16 Bytes des aus der Steganografie stammenden Byte-Arrays als Initialisierungsvektor für die bevorstehende Entschlüsselung im CBC-Modus bestimmt. Der Salt für die Dechiffrierung entspricht wiederum den vorletzten 16 Bytes. Nachdem die letzten 32 Bytes für den Salt und den IV ausgelesen wurden, wird das Array um 32 Bytes gekürzt. Nach der Kürzung steht nur noch die Chiffre im Array. Mit Hilfe des im Reiter „Files“ eingegebenen Passwortes und des ausgelesenen Salts wird mittels PBKDF2 über 10.000 Iterationen der Schlüssel für die Entschlüsselung berechnet. Anschließend wird dieser Schlüssel mit einer Länge von 256 Bits neben weiteren Parametern, wie der Blockgröße von 128 Bits, als eine Eigenschaft der bevorstehenden Entschlüsselung vereinbart. Nach all diesen Vorbereitungen wird die Chiffre schließlich zu einem neuen Byte-Array entschlüsselt. Die Inhalte dieses resultierenden Arrays werden abschließend in eine neue Datei geschrieben, welche die im Reiter „Files“ angegebene Zieldatei überschreibt. Das finale Ergebnis dieses Entschlüsselungsprozesses ist also eine extrahierte und entschlüsselte Datei.

3.7 Implementierung sonstiger Funktionen

Im Folgenden geht es um alle wichtigen zusätzlich zur Krypto- und Steganografie implementierten Funktionalitäten. Die hier vorgestellten Softwarebestandteile sind zwar nicht der Kern der in den letzten Abschnitten erläuterten Verfahren, aber dennoch für deren ordnungsgemäße und intuitive Bedienung erforderlich.

3.7.1 Funktionen für die Dateiauswahl

Die grafische Oberfläche verfügt insgesamt über drei verschiedene Buttons zur Dateiauswahl mit der Beschriftung „Browse...“. Darüber können Träger-, Quell- und Zieldateien für die unterschiedlichen Verfahren ausgewählt werden. Beim Klick auf den Button für die Auswahl der Trägerdatei wird zuerst ein neues Objekt der Klasse „OpenFileDialog“ angelegt. Diese Klasse dient zur Erzeugung eines Explorerfensters, über das eine Datei ausgewählt bzw. geöffnet werden kann. Anschließend prüft „StegaCrypto“, welche Steganografie in der entsprechenden Listbox im gleichen Reiter ausgewählt wurde. Dies geschieht, um die Dateiauswahl auf zur jeweiligen Steganografie passende Formate einzuschränken. So werden für die RGB-Steganografie einige kompressionsfreie Formate wie PNG erlaubt, während für die MIDI-Steganografie ausschließlich MIDI-Dateien ausgewählt werden können. Die gleichzeitige Auswahl mehrerer Dateien wird durch eine dafür eingebaute Anweisung unterbunden. Als Startverzeichnis dient stets der Desktop des aktuellen Users. Durch die Methode „ShowDialog“ kommt es schließlich zur Anzeige des bisher konfigurierten Fensters und wenn die Dateiauswahl daraufhin nicht abgebrochen wird, erfolgt die Speicherung des Pfades der selektierten Datei in der hinter dem Schriftzug „Carrier File :“ positionierten Textbox. Die Auswahl der Quelldatei erfolgt weitgehend analog. Der einzige Unterschied besteht darin, dass im Browserfenster nicht nach bestimmten Dateiformaten gefiltert wird. Die Dateiendung der Quelldatei ist für den steganografischen Versteckvorgang völlig irrelevant, da jede beliebige Datei in ein Byte-Array umgewandelt werden kann und sich dieses dann verstecken lässt. Bei der steganografischen Extraktion muss das Format der Quelldatei jedoch zum gewählten Verfahren passen. Um diese Einschränkung zu berücksichtigen, aber dennoch eine Auswahl beliebiger Formate für den Versteckvorgang zu ermöglichen, wird der „Decrypt“-Button für die Extraktion nur bei geeigneten Kombinationen aus Steganografie und Quelldateityp freigegeben. Diese Button-Freigabe erfolgt durch eine separate Funktion, die später in Abschnitt 3.7.4 genauer erklärt wird. Der Programmcode für die Selektion der Zieldatei ist analog zum Prozedere für die Quelldateiauswahl. Da beim steganografischen Versteckvorgang eine kompatible Kombination aus Steganografie und Zieldateiformat vorliegen muss, wird auch der „Encrypt“-Button nur bei geeigneten Bedingungen durch die in Abschnitt 3.7.4 vorgestellte Funktion freigegeben.

3.7.2 Laden von Pixelgrafiken mit folgender Dateifreigabe

Im Programmcode ist eine Funktion mit dem Namen „loadBitmapUnlocked“ enthalten, welche zur Umgehung eines ganz bestimmten Problems implementiert wurde. Wie bereits in sämtlichen Abschnitten zur RGB-Steganografie im Methodenteil erwähnt, verwendet „StegaCrypto“ zum Laden von Pixelgrafiken aus Dateien die Klasse „Bitmap“. Normalerweise geht dieser Vorgang mit der Nebenwirkung einher, dass eine Manipulation der Ursprungsdatei dieser Grafik verhindert wird. Dieser Nebeneffekt ist in „StegaCrypto“ jedoch unerwünscht, da man somit bspw. eine Trägerdatei nicht mit der durch Steganografie bearbeiteten Version der Datei überschreiben kann. Man müsste das modifizierte Bild also stets in eine andere Datei schreiben. „loadBitmapUnlocked“ erlaubt jedoch das Laden einer Pixelgrafik mit anschließender Freigabe der Quelldatei für dateiverändernde Operationen. Für diesen Zweck initialisiert die Funktion ein Objekt der „Bitmap“-Klasse mit der Pixelgrafik aus der als Parameter übergebenen Datei unter Einsatz einer Using-Anweisung und sendet die geladene Grafik als Rückgabewert. „using“ sorgt dafür, dass Objekte gleich nach ihrer Verwendung wieder verworfen beziehungsweise freigegeben werden. Aus diesem Grund ist die Quelle der Pixelgrafik nach dem Ladevorgang nicht mehr für Veränderungen gesperrt. Der Programmcode für „loadBitmapUnlocked“ geht auf die Quelle [Stephens, 2014] zurück.

3.7.3 Konvertierung von Dateien in Byte-Arrays

Für die Realisierung der in den Abschnitten 3.5 und 3.6 beschriebenen dateibasierten Verfahren ist die Umwandlung von Dateien in Byte-Arrays von essentieller Bedeutung. Da das dafür notwendige Prozedere sehr oft im Code abgerufen werden muss und dieser aus Gründen der Übersichtlichkeit nicht unnötig aufgebläht werden sollte, wurde die Funktion „convertFileToBytes“ implementiert. Diese Funktion liest die ihr als Parameter übergebene Datei byteweise in ein Byte-Array und sendet dieses schließlich als Rückgabewert. Der Programmcode für „convertFileToBytes“ geht auf die Quelle [Kumar, 2021] zurück.

3.7.4 Freigabe von Steuerelementen

Um die Intuitivität der Bedienoberfläche zu gewährleisten, sollte die Nutzung von Steuerelementen unterbunden werden, solange keine gültigen Eingabedaten für die ihnen zugrunde liegenden Funktionen bereitstehen. So ergibt ein Klick auf „Encrypt“ im Reiter „Texts“ beispielsweise keinen Sinn, wenn noch keine zu versteckende Nachricht hinterlegt wurde. Darüber hinaus ist diese Form der UI-Automatisierung auch aus den in Abschnitt 3.7.1 geschilderten Gründen sinnvoll. Umgesetzt wurde dieser Sachverhalt in Form einer eigenständigen Funktion namens „control_dataChanged“, welche automatisch bei jeder Veränderung der Eingabedaten ausgeführt wird. Unter den „Eingabedaten“ sind dabei die Inhalte aller Textboxen sowie die aktuelle Auswahl in den „Listbox“-Steuerelementen für Krypto- und Steganografie zu verstehen. Die Funktion prüft bei jedem Aufruf für jeden einzelnen Button eine Reihe bestimmter Freigabekriterien. Sind diese kontrollierten Voraussetzungen erfüllt, wird der Button für die Ansteuerung freigegeben. Eine detaillierte und vollständige Übersicht über alle von „control_dataChanged“ untersuchten Freigabekriterien bietet Tabelle .3 im Anhang dieser Arbeit.

4 Ergebnisse

In diesem Teil der Masterarbeit wird das finale Ergebnis der im letzten Kapitel vorgestellten Methoden zusammengefasst und vorgestellt. Bei diesem Endresultat handelt es sich um die abgabefertige Version des Prototypen. Zu Beginn wird das Endprodukt mit dem ursprünglich formulierten Entwicklungskonzept verglichen. Darauf folgend werden alle in der Anwendung verfügbaren Verfahrenskombinationen aufgelistet, die sich aus den umgesetzten Bestandteilen des Entwicklungskonzeptes ableiten. Abschließend wird jede dieser Kombinationen auf ihre Funktionalität geprüft. Auf einen separaten Abschnitt für die finale UI wurde in diesem Kapitel dabei verzichtet, da die Beurteilung ihrer Inuitivität erst im Diskussionsteil der Arbeit erfolgt und ihr Aufbau exakt den Abbildungen sowie Erläuterungen auf den Seiten 36 und 37 entspricht. Weiterhin erfolgt das Testen der Bedienoberfläche bereits im Zusammenhang mit dem testen der Verfahrenskombinationen, da diese darüber gesteuert werden.

4.1 Abgleich des Prototypen mit dem Entwicklungskonzept

Die folgenden Punkte aus dem Entwicklungskonzept auf Seite 33 sind vollständig in der Software enthalten:

1. Ver- und Entschlüsselung von Textnachrichten durch AES-256
2. Ver- und Entschlüsselung von Dateien mittels AES-256
3. Verstecken und Extrahieren von Textnachrichten mittels RGB-Steganografie
4. Verstecken und Extrahieren von Dateien mittels RGB-Steganografie
5. Verstecken und Extrahieren von Textnachrichten mittels MIDI-Steganografie auf der Basis von „Program Change“-Ereignissen
6. Verstecken und Extrahieren von Dateien mittels MIDI-Steganografie auf der Basis von „Program Change“-Ereignissen
7. Kombination der in 1. und 2. aufgelisteten Verfahren mit einem Passwortschutz auf Basis der Schlüsselableitungsfunktion PBKDF2
8. Verknüpfung der in 1.-7. aufgelisteten Verfahren und einfache Steuerung der Anwendung durch eine intuitive Benutzeroberfläche

Aus diesem Abgleich geht hervor, dass alle geplanten Softwarebestandteile realisiert werden konnten. Da der Prototyp „StegaCrypto“ somit 100% des Entwicklungskonzeptes umfasst, wurde der Softwarename um die Versionsnummer 1.0 erweitert.

4.2 Verfahrenskombinationen

Aus der Gesamtheit aller in „StegaCrypto“ verfügbaren Funktionen ergeben sich vier mögliche Verfahrenskombinationen beziehungsweise Aufgaben, für welche die Software eingesetzt werden kann:

1. Kombination von AES-256 mit RGB-Steganografie (textbasiert)
2. Kombination von AES-256 mit MIDI-Steganografie (textbasiert)
3. Kombination von AES-256 mit RGB-Steganografie (dateibasiert)
4. Kombination von AES-256 mit MIDI-Steganografie (dateibasiert)

Für jede dieser vier Möglichkeiten wird die entwickelte Software in den nachfolgenden Abschnitten auf ihren ordnungsgemäßen Betrieb geprüft. Um den Beschreibungen der Interaktion mit der UI besser folgen zu können, sollten die beiden Abbildungen auf Seite 36 und Seite 37 oder die Software selbst hinzugezogen werden.

4.2.1 AES-256 und RGB-Steganografie (textbasiert)

Verschlüsseln und Verstecken
Im Reiter „Texts“ wurde zunächst „AES-256“ hinter dem Schriftzug „Cryptography :“ ausgewählt. Hinter dem Text „Steganography : “ wurde „RGB - Manipulation of Color Values“ als Steganografie selektiert. Anschließend wurde für unterschiedliche Textnachrichten und Trägerdateien das Verstecken und Verschlüsseln getestet. Die Verfahrenskombination funktionierte dabei wie vorgesehen. Mit bloßem Auge waren oftmals keine optischen Veränderungen im als Trägerdatei verwendeten Bild erkennbar. Auf die konkreten Faktoren für die Unauffälligkeit und die Sicherheit der Verfahren wird später in Abschnitt 5.1.3 auf Seite 61 genauer eingegangen. Zudem veränderten sich die Dateigrößen durch das steganografische Verstecken deutlich, in einem Fall beispielsweise von 513KB auf 0,98MB.

Extrahieren und Entschlüsseln
Wenn das Verschlüsseln und Verstecken für diese Verfahrenskombination ordnungsgemäß funktionieren, ist der Vorgang durch einen Klick auf „Decrypt“ rückführbar, insofern als Steganografie, Kryptografie und Passwort die gleichen Daten angegeben sind sowie die aus dem Verschlüsseln und Verstecken resultierende Trägerdatei ausgewählt wurde. Die entsprechenden Tests erwiesen sich als erfolgreich. Zuvor verschlüsselte und versteckte Botschaften konnten stets vollständig und fehlerfrei ermittelt und im entsprechenden Feld der Bedienoberfläche angezeigt werden, was ein klares Zeichen für die ordnungsgemäß funktionierende Extraktion und Entschlüsselung ist.

4.2.2 AES-256 und MIDI-Steganografie (textbasiert)

Verschlüsseln und Verstecken
Im Reiter „Texts“ wurde zunächst „AES-256“ hinter dem Schriftzug „Cryptography :“ ausgewählt. Hinter dem Text „Steganography : “ wurde „MIDI - Insertion of Program Changes“ als Steganografie selektiert. Anschließend wurde für unterschiedliche Textnachrichten und Trägerdateien das Verstecken und Verschlüsseln getestet. Die Verfahrenskombination funktionierte dabei wie vorgesehen. Nach dem Verschlüsselungs- und Versteckvorgang lagen in keinem Fall hörbare Veränderungen der Trägerdatei vor. Auf die Unauffälligkeit und Sicherheit der Verfahren wird später in Abschnitt 5.1.4 auf Seite 62 genauer eingegangen. Zudem veränderten sich die Dateigrößen durch das steganografische Verstecken, in einem Fall beispielsweise von 9,30KB auf 12,1MB.

Extrahieren und Entschlüsseln
Wenn das Verschlüsseln und Verstecken für diese Verfahrenskombination ordnungsgemäß funktionieren, ist der Vorgang durch einen Klick auf „Decrypt“ rückführbar, insofern als Steganografie, Kryptografie und Passwort die gleichen Daten angegeben sind sowie die aus dem Verschlüsseln und Verstecken resultierende Trägerdatei ausgewählt wurde. Die entsprechenden Tests erwiesen sich als erfolgreich. Zuvor verschlüsselte und versteckte Botschaften konnten stets vollständig und fehlerfrei ermittelt und im entsprechenden Feld der Bedienoberfläche angezeigt werden, was ein klares Zeichen für die ordnungsgemäß funktionierende Extraktion und Entschlüsselung ist.

4.2.3 AES-256 und RGB-Steganografie (dateibasiert)

Verschlüsseln und Verstecken
Im Reiter „Files“ wurde zunächst „AES-256“ hinter dem Schriftzug „Cryptography :“ ausgewählt. Hinter dem Text „Steganography : “ wurde „RGB - Manipulation of Color Values“ als Steganografie selektiert. Anschließend wurde für unterschiedliche Quell- und Zieldateien das Verstecken und Verschlüsseln getestet. Die Verfahrenskombination funktionierte dabei wie vorgesehen. Mit bloßem Auge waren oftmals keine optischen Veränderungen im als Zieldatei verwendeten Bild erkennbar. Auf die konkreten Faktoren für die Unauffälligkeit und die Sicherheit der Verfahren wird später in Abschnitt 5.1.3 auf Seite 61 genauer eingegangen. Zudem veränderten sich die Dateigrößen durch das steganografische Verstecken, in einem Fall beispielsweise von 3,27MB auf 3,50MB. Die Quelldatei kann in einem beliebigen Dateiformat vorliegen. Bei den Tests kamen unter anderem PNG- und MIDI-Dateien zum Einsatz. Bei der Zieldatei muss es sich aber logischerweise um eine PNG-, BMP- oder GIF-Datei handeln, damit eine Kompatibilität zur ausgewählten Steganografie gegeben ist.

Extrahieren und Entschlüsseln
Wenn das Verschlüsseln und Verstecken für diese Verfahrenskombination ordnungsgemäß funktionieren, ist der Vorgang durch einen Klick auf „Decrypt“ rückführbar, insofern als Steganografie, Kryptografie und Passwort die gleichen Daten angegeben sind sowie die aus dem Verschlüsseln und Verstecken resultierende Datei als Quelldatei ausgewählt wurde. Die selektierte Zieldatei wird nach erfolgreicher Extraktion und Entschlüsselung durch die dabei ausgelesene Datei überschrieben und gibt lediglich Dateiformat und -name für deren Speicherung vor. Die entsprechenden Tests erwiesen sich als erfolgreich. Zuvor verschlüsselte und versteckte Dateien konnten stets vollständig und fehlerfrei ermittelt und unter dem entsprechenden Pfad im Zieldateifeld der Bedienoberfläche erzeugt werden, was ein klares Zeichen für die ordnungsgemäß funktionierende Extraktion und Entschlüsselung ist. Die Zieldatei für die Extraktion kann in einem beliebigen Dateiformat vorliegen. Bei der Quelldatei muss es sich aber logischerweise um eine PNG-, BMP- oder GIF-Datei handeln, damit eine Kompatibilität zur ausgewählten Steganografie gegeben ist.

4.2.4 AES-256 und MIDI-Steganografie (dateibasiert)

Verschlüsseln und Verstecken
Im Reiter „Files“ wurde zunächst „AES-256“ hinter dem Schriftzug „Cryptography :“ ausgewählt. Hinter dem Text „Steganography : “ wurde „MIDI - Insertion of Program Changes“ als Steganografie selektiert. Anschließend wurde für unterschiedliche Quell- und Zieldateien das Verstecken und Verschlüsseln getestet. Die Verfahrenskombination funktionierte dabei wie vorgesehen. Es kam bei keinem der Tests zu hörbaren Veränderungen der Zieldatei. Auf die konkreten Faktoren für die Unauffälligkeit und die Sicherheit der Verfahren wird später in Abschnitt 5.1.4 auf Seite 62 genauer eingegangen. Zudem veränderten sich die Dateigrößen durch das steganografische Verstecken deutlich, in einem Fall beispielsweise von 12,3KB auf 65,6KB. Die Quelldatei kann in einem beliebigen Dateiformat vorliegen. Bei der Zieldatei muss es sich aber logischerweise um eine MIDI-Datei handeln, damit eine Kompatibilität zur ausgewählten Steganografie gegeben ist.

Extrahieren und Entschlüsseln

Wenn das Verschlüsseln und Verstecken für diese Verfahrenskombination ordnungsgemäß funktionieren, ist der Vorgang durch einen Klick auf „Decrypt“ rückführbar, insofern als Steganografie, Kryptografie und Passwort die gleichen Daten angegeben sind sowie die aus dem Verschlüsseln und Verstecken resultierende Datei als Quelldatei ausgewählt wurde. Die selektierte Zieldatei wird nach erfolgreicher Extraktion und Entschlüsselung durch die dabei ausgelesene Datei überschrieben und gibt lediglich Dateiformat und -name für deren Speicherung vor. Die entsprechenden Tests erwiesen sich als erfolgreich. Zuvor verschlüsselte und versteckte Dateien konnten stets vollständig und fehlerfrei ermittelt und unter dem entsprechenden Pfad im Zieldateifeld der Bedienoberfläche erzeugt werden, was ein klares Zeichen für die ordnungsgemäß funktionierende Extraktion und Entschlüsselung ist. Die Zieldatei für die Extraktion kann in einem beliebigen Dateiformat vorliegen. Bei der Quelldatei muss es sich aber logischerweise um eine Datei im MIDI-Format handeln, damit eine Kompatibilität zur ausgewählten Steganografie gegeben ist.

5 Diskussion

5.1 Evaluation des Prototypen

Dieser Abschnitt der Masterarbeit befasst sich mit der Bewertung der erzielten Ergebnisse. Zuerst werden die finalen Resultate quantitativ im Hinblick auf die in der Einleitung formulierte Zielstellung sowie das auf Seite 33 formulierte Entwicklungskonzept bewertet. Anschließend erfolgt die qualitative Beurteilung für jedes einzelne in „StegaCrypto“ implementierte Verfahren durch eine detaillierte Analyse. Abschließend wird die Intuitivität der integrierten Bedienoberfläche anhand der auf Seite 34 aufgestellten Intuitivitätskriterien untersucht.

5.1.1 Allgemeine Einordnung des Gesamtergebnisses

Aus Abschnitt 4.1 auf Seite 53 geht hervor, dass die finale Version des Prototypen 100% des ursprünglich aufgestellten Entwicklungskonzeptes umfasst. Wie die nächsten Abschnitte des Diskussionskapitels zeigen werden, bietet der Prototyp mehrere Möglichkeiten zur sicheren Speicherung von Daten bei glaubhafter Abstreitbarkeit und ist ein nicht zu unterschätzendes antiforensisches Werkzeug. Die erzielten Ergebnisse sind damit insbesondere hinsichtlich des Entwicklungskonzeptes weitaus mehr als zufriedenstellend, auch wenn für die praktische Anwendung gewisse Grenzen existieren beziehungsweise bestimmte Umstände erfüllt sein müssen. In den folgenden Abschnitten werden diese Gegebenheiten detailliert erläutert.

5.1.2 Kryptoanalyse für AES-256 und PBKDF2

Wie bereits in Abschnitt 2.1.2 des Grundlagenteils dieser Arbeit dargelegt, handelt es sich bei AES-256 um einen weit verbreiteten und als sehr sicher geltenden kryptografischen Standard. Sollten bedeutende Vulnerabilitäten der in „StegaCrypto“ implementierten Ver- und Entschlüsselung vorliegen, sind diese also auf die konkrete Implementierung innerhalb des Programmcodes und nicht auf den Algorithmus selbst zurückzuführen. Um derartige Fehler zu vermeiden, wurden offiziell bereitgestellte Bibliotheken des Frameworks „.NET Core 3.1“ für die Implementierung der für die Kryptografie bedeutsamen Verfahren verwendet. Für die Anwendung dieser Frameworkbestandteile dienten stets die in der offiziellen Sprachdokumentation von Microsoft formulierten Beispiele und Erklärungen als Vorbild und wichtigste Quelle. Die für den Salt und Initialisierungsvektor benötigten Zufallswerte, wurden durch exakt für kryptografische Zwecke entwickelte Zufallsfunktionenn des Frameworks generiert. Wird für PBKDF2 eine zu geringe Iterationszahl verwendet, könnten aus gespeicherten Hashes die Eingabedaten durch Brute-Force-Angriffe abgeleitet

werden. Da „StegaCrypto“ PBKDF2 jedoch nicht zur Speicherung von Passworthashes in Dateien und Datenbanken verwendet, sondern die Funktion zur Ableitung von für die Kryptografie geeigneten Schlüsseln dient, ist diese Vulnerabilität für dieses Anwendungsbeispiel weniger relevant. Sicherheitshalber wurden dennoch wie in [Grassi et al., 2017] empfohlen 10.000 Iterationen verwendet. Die so erzeugten Schlüssel für AES-256 werden ebenfalls nicht in Dateien oder Datenbanken gespeichert, sondern nach ihrer Verwendung innerhalb der kryptografischen Prozesse verworfen. Da die Sicherheit der Kryptografie von der Geheimhaltung des Schlüssels abhängt, ist dieses Vorgehen essentiell. Das Passwort wird selbstverständlich ebenfalls nicht gespeichert, da man daraus den Schlüssel unter Nutzung der Hashfunktion ableiten könnte. Aufgrund der bisher formulierten Umstände fallen bei der Evaluation zunächst keine Vulnerabilitäten auf. Eine Rekonstruktion der verschlüsselten Daten ohne Informationen über das Passwort oder den daraus abgeleiteten Schlüssel scheint bislang unmöglich. Die Implementierung von AES-256 erfüllt damit ihr in Abschnitt 1.2 formuliertes Ziel, eine hohe Sicherheit der Informationen bzw. Daten zu gewährleisten. Es wird hier jedoch ganz ausdrücklich darauf hingewiesen, dass die Sicherheit kryptografischer Implementierungen stets von möglichst vielen Persönlichkeiten mit entsprechender Fachexpertise überprüft werden sollte. Die durch den Autor dieser Arbeit erfolgten Bemerkungen zur Sicherheit sind daher mit äußerster Vorsicht und ohne Gewähr zu betrachten.

5.1.3 Diskretion der RGB-Steganografie

Unter der Diskretion des Verfahrens ist zu verstehen, dass dessen Präsenz nicht ohne Weiteres erfassbar ist, insbesondere im Hinblick auf die menschliche Wahrnehmung. Für die RGB-Steganografie konnten Zusammenhänge zwischen der Verfahrensdiskretion und folgenden Faktoren festgestellt werden:

1. Verhältnis zwischen Umfang der zu versteckenden Daten und Auflösung der Pixelgrafik in der Trägerdatei
2. Eignung des für die Steganografie selektierten Farbwertes für den Farbverlauf der Pixelgrafik in der Trägerdatei
3. Verteilungsintervall für die steganografisch verarbeiteten Pixel

Die Diskretion der RGB-Steganografie beruht zu einem wesentlichen Teil auf der Tatsache, dass nur ein Bruchteil aller zur Grafik in der Trägerdatei gehörenden Pixel verändert wird. Dadurch bleiben die Manipulationen für das menschliche Auge bei konventioneller Bildbetrachtung verborgen. Eine unkonventionelle Betrachtungsweise wäre beispielsweise ein Vergleich der einzelnen Pixel vor und nach der Steganografie unter Nutzung eines sehr großen Zoomfaktors. Je umfangreicher die zu versteckende Datenmenge und je niedriger die Auflösung der Grafik in der Trägerdatei ist, desto größer wird der Anteil der steganografisch modifizierten Pixel am gesamten Bild. Wenn dieser Anteil zu groß wird, ist eine klare optische Verzerrung auch bei konventioneller Betrachtung erkennbar. Je nachdem, ob die zu versteckenden Daten in den Rot-, Grün- oder Blauwerten der Pixel kodiert werden, sind unterschiedliche Farbverläufe in den Trägergrafiken besonders hilfreich für die Diskretion. Wird zum Beispiel stets der Blauwert für die RGB-Steganografie verwendet, so entstehen in der Regel viele blaue Punkte in der Grafik. Handelt es sich dabei jedoch um ein Bild mit vorwiegend blauen Tönen, sind diese Punkte selbst bei größerer Dichte und Anzahl oft nicht erkennbar. Das im dritten Faktor genannte Verteilungsintervall bestimmt, wie weit die X- und Y-Koordinaten der in die Steganografie involvierten Pixel gestreut sind. Je stärker die vom Verfahren modifizierten Pixel über die Grafik verteilt sind, desto unauffälliger bzw. diskreter ist die Steganografie. Der Nachteil großer Verteilungsintervalle besteht jedoch darin, dass eine deutlich höhere Auflösung der Trägergrafik für die gleiche Menge zu versteckender Daten benötigt wird, um diese vollständig darin unterzubringen. Bei den für die RGB-Steganografie durchgeführten Tests erwies sich das Verfahren als extrem diskret. Keines der Testbeispiele führte zu optischen Verzerrungen, die bei konventioneller Bildbetrachtung ersichtlich waren. Das in 1.2 formulierte Ziel der Gewährleistung einer glaubhaften Abstreitbarkeit wird damit grundlegend durch die Implementierung erfüllt. Diese enorme Diskretion wurde vor allem durch eine sehr starke Verteilung der Pixel erreicht. Nur jede fünfte Spalte und darin jedes fünfte Pixel werden vom Verfahren modifiziert. Dementsprechend schwierig gestaltet sich jedoch das Verstecken besonders großer Datenmengen. Somit können bislang nur kleinere Dateien (<1MB) in hochauflösenden Grafiken (z.B. 1920x1200) versteckt werden. Abschließend ist noch zu erwähnen, dass es durch das Verfahren zu Veränderungen der Dateigröße kommt, welche bei den Tests

der textbasierten Variante besonders deutlich ausfielen. So wurde bei einem der Tests ein Größenzuwachs von 513KB auf 0,98MB beobachtet. Solange die Größe der Datei im unbearbeiteten Originalzustand nicht bekannt ist und die Größe der Trägerdatei nicht explizit auf ihre Plausibilität geprüft wird, sollte dies jedoch nicht weiter auffallen. Für die menschliche Wahrnehmung bleiben die erfolgten Prozesse also weitgehend verborgen.

5.1.4 Diskretion der MIDI-Steganografie

Unter der Diskretion des Verfahrens ist zu verstehen, dass dessen Präsenz nicht ohne Weiteres erfassbar ist, insbesondere im Hinblick auf die menschliche Wahrnehmung. Sowohl die textbasierte als auch die dateibasierte Implementierung der MIDI-Steganografie erwies sich wie geplant als äußerst diskret und unauffällig bei der Dateiausführung. Im Rahmen der erfolgten Tests konnten keine auditiven Verzerrungen in den durch die Steganografie modifizierten Dateien festgestellt werden. In Abschnitt 2.3.3 wurde bereits erläutert, dass derartige Abweichungen vom Original komplett vermeidbar für das verwendete Verfahren sind. Die Menge der auf diese Weise versteckten Daten spielt daher für die auditive Diskretion überhaupt keine Rolle. Werden die Dateien jedoch im Windows-Explorer oder anderweitig in Kombination mit ihrer Dateigröße betrachtet, so fallen die steganografisch manipulierten MIDI-Dateien teilweise sehr stark durch ihre Größe auf. Für die meisten Texte und sehr kleine versteckte Dateien (z.B. 1KB) sind diese resultierenden Veränderungen nicht besonders auffällig, solange die Größe nicht mit der unbearbeiteten Datei verglichen wird. Für größere Dateien oder Texte von wirklich extraordinärer Länge (bspw. 40KB) sind die die Abweichungen jedoch enorm. So konnte für ein Dateipaar beispielsweise ein Größenzuwachs von 12,3KB auf 65,6KB beobachtet werden. Weiterhin ist die auditive Unauffälligkeit nicht nur von Vorteil, da es somit keinen Grund abseits der Steganografie gibt, um mehrere „Program Change“-Ereignisse direkt hintereinander zu platzieren. Das Verfahren kann daher sehr einfach und effektiv durch eine automatisierte Steganalyse detektiert werden. Es muss dafür lediglich nach direkt aufeinander folgenden „Program Changes“ mit einer Deltazeit von Null gesucht werden. Die Implementierung der MIDI-Steganografie erfüllt somit grundlegend das in Abschnitt 1.2 formulierte Ziel der Gewährleistung einer glaubhaften Abstreitbarkeit durch eine ausreichende Diskretion, weil die menschliche Wahrnehmung nicht ohne Weiteres zur Detektion genügt. Es muss jedoch darauf geachtet werden, dass nicht zu große Datenmengen mit diesem Verfahren versteckt werden, wenn eine auditive Diskretion allein nicht für den Anwendungsfall ausreicht.

5.1.5 Intuitivität der UI

Die Bewertung der Intuitivität der Bedienoberfläche erfolgt anhand der in Abschnitt 3.2 auf Seite 34 aufgestellten Intuitivitätskriterien.

1. Keine Redundanz:
   Die grafische Oberfläche verfügt über keine redundanten Steuerelemente und Ausgabeinformationen. Das erste Kriterium ist damit erfüllt.
2. Vollständigkeit:
   Sowohl beim Erfolg der Verfahren als auch bei Fehlern wird der die Software steuernde Mensch stets entsprechend durch die UI über diese Ereignisse informiert. Alle weiteren relevanten Informationen kann der Mensch seinen eigenen Eingaben entnehmen, so z.B. den Dateipfad einer verwendeten Trägerdatei. Das zweite Kriterium kann somit ebenfalls als erfüllt angesehen werden.
3. Logische Anordnung:
   Zueinander gehörende bzw. sich gegenseitig beeinflussende Steuerelemente wurden dementsprechend auf der grafischen Oberfläche gruppiert. Weiterhin sind die Steuerelemente auch entsprechend der Reihenfolge, in welcher der Prototyp mit Eingabedaten zu versorgen ist, angeordnet. Auch das dritte Intuitivitätskriterium wird also durch die entwickelte Bedienoberfläche von „StegaCrypto“ erfüllt.
4. Übersichtlichkeit:
   Alle Funktionen der Software sind schnell und leicht zu finden. Sie wurden übersichtlich in text- und dateibasierte Verfahren gruppiert. Anschließend können die konkreten krypto- und steganografischen Methoden aus vorgefertigten Übersichten bequem per Klick ausgewählt werden. Zu einer Verstärkung der Übersichtlichkeit wurden die Steuerelemente für text- und dateibasierte Verfahren in möglichst gleicher Reihenfolge bzw. an ähnlichen Positionen angeordnet. Das vierte Kriterium ist damit erfüllt.
5. Pragmatismus:
   Alle Funktionen von „StegaCrypto“ können schnell über ein gemeinsames Fenster angesteuert und verwendet werden. Lediglich die Meldungen für erfolgreiche und fehlgeschlagene Verfahrensausführungen werden in separaten Fenstern angezeigt. Es sind nur wenige Steuerbefehle für den ordnungsgemäßen Betrieb erforderlich. Das fünfte Intuitivitätskriterium ist damit erfüllt.
6. Prinzipientreue:
   Die Anordnung, Optik und Funktionalität der Steuerelemente steht nicht im Widerspruch zu als Allgemeinwissen geltenden Standards und Prinzipien. Grafische Oberflächen anderer Anwendungen und geläufige Prinzipien wurden berücksichtigt. Auch das sechste Intuitivitätskriterium ist damit erfüllt.

Insgesamt ist die wurde das Ziel der Gewährleistung einer einfachen Softwarebedienung durch die intuitive UI also absolut erfüllt. Alle ursprünglich formulierten Intuitivitätskriterien konnten durch die entwickelte Bedienoberfläche eingehalten werden.

5.1.6 Praktische Einsatzmöglichkeiten

Bevor es zum Einsatz von „StegaCrypto“ in praktischen Szenarien innerhalb der Menschenrechtsarbeit und des investigativen Journalismus kommt, sollte unbedingt die Sicherheit der Implementierung von AES-256 durch möglichst viele Personen mit entsprechender Fachexpertise überprüft werden. Da von der technischen Korrektheit und Sicherheit der in diesen Bereichen eingesetzten Technologien sogar Menschenleben abhängen können, ist diese äußerst kritisch zu untersuchen. Im Hinblick auf die Steganografie ist eine praktische Anwendung durchaus vorstellbar. Es sind dabei jedoch die in den Abschnitten 5.1.3 und 5.1.4 aufgeführten Grenzen bzw. Diskretionsfaktoren der implementierten Verfahren zu berücksichtigen. Insbesondere wenn sehr große Datenmengen steganografisch versteckt werden sollen, ist eine Weiterentwicklung des Prototypen definitiv erforderlich. Weiterhin ist zu empfehlen, ein möglichst großes Rauschen in den verwendeten Datenspeichern zu erzeugen. Das bedeutet, dass die steganografisch und kryptografisch hinterlegten Informationen zusammen mit vielen aus forensischer Sicht völlig unbedeutenden Daten gespeichert werden sollen, um eine extrem tiefgehende Dateianalyse für jede einzelne Datei im Speicher zu einem nicht praktikablen Vorhaben zu machen. So kann die in dieser Arbeit thematisierte und verwirklichte MIDI-Steganografie bspw. sehr leicht durch die Betrachtung innerhalb eines Hex-Editors festgestellt werden, wenn beim forensischen Personal hinreichende Kenntnisse über den Aufbau von MIDI-Dateien vorliegen oder diese recherchiert werden. Eine derart tiefgehende und gewissenhafte Analyse jeder einzelnen Datei ist aber aus kapazitiven Gründen nicht wirklich in der forensischen Praxis realisierbar, wenn die Menge der zu untersuchenden Daten sehr umfangreich ist und bspw. mehrere Terrabytes umfasst. Die forensische Arbeit wird auch dadurch erschwert, dass es sehr viele Möglichkeiten bzw. sehr unterschiedliche Varianten für die Steganografie gibt. So können die in der RGB-Steganografie versteckten Daten durch unterschiedliche Farbwerte kodiert und auch an völlig unterschiedlichen Positionen im Bild versteckt werden. Auch das macht eine umfassende Steganalyse bei einem großen Rauschen im Datensatz äußerst aufwendig und anspruchsvoll. Eine weitere Herausforderung für die forensisch agierende Seite bei der Erkennung von Steganografie besteht darin, dass selbst erfolgreich extrahierte Daten zunächst absolut keinen Sinn ergeben, da es sich um nicht ohne Weiteres interpretierbare Chiffren handelt. Durch eine zukünftige Weiterentwicklung von „StegaCrypto“ könnte das antiforensische Potential des Prototypen noch deutlich erweitert werden, mehr dazu später in Abschnitt 5.3 Ausblick.

5.2 Einordnung der Arbeit in die aktuelle Forschung und Abgrenzung

Es handelt sich bei den in den Prototyp implementierten Verfahren nicht um grundlegend neue Technologien. Dies geht beispielsweise aus den Quellen [Paar und Pelzl, 2011], [John, 2004] und [Kanal D3NCE, 2020] hervor. Allerdings spezialisierten sich die im Rahmen dieser Arbeit gesichteten praktischen Quellen stets auf ein krypto- oder steganografisches Verfahren. Weiterhin wurde in den Quellen stets die textbasierte, aber nicht die dateibasierte Verfahrensvariante thematisiert und praktisch umgesetzt. Die vorliegende Matserarbeit grenzt sich also nicht durch die zugrunde liegende Theorie ihrer Kernbestandteile, sondern durch die Kombination sehr unterschiedlicher Verfahren innerhalb eines gemeinsamen Softwareprojektes sowie durch die Weiterentwicklung der Verfahren und ihre Ausweitung auf Dateiverarbeitung ab. So können die in „StegaCrypto“ implementierten Versionen der Verfahren weitaus größere Datenmengen verstecken als in den aufgeführten Quellen. Die in [Kanal D3NCE, 2020] demonstrierte Implementierung der RGB-Steganographie kann z.B. nur Texte mit einer Größe von maximal 255 Bytes verstecken, während „StegaCrypto“ Längen von bis zu 2.147.483.647 Bytes kodieren kann. Die Maximalmenge der steganografisch versteckten Daten wird somit praktisch nur durch die Pixelanzahl der Trägergrafik und die in Abschnitt 5.1.3 erörterten Diskretionsfaktoren begrenzt. Die Implementierung der MIDI-Steganografie kann in der Theorie bis zu 1.073.741.823 Bytes verstecken und deren Länge kodieren. Auch für dieses Verfahren hängen die Grenzen der Datenmenge also eher von der dadurch beeinflussten Verfahrensdiskretion ab, siehe dafür Abschnitt 5.1.4. Die dateibasierten Versionen der steganografischen Verfahren stellen eine Neuerung und Weiterentwicklung gegenüber den verwendeten Quellen dar. Auch die vorliegende Kombination der Steganografie mit AES-256 ist eine Optimierung und gewährleistet neben der Diskretion auch eine erhebliche Informationssicherheit. Selbst wenn die Daten nach erfolgreicher Steganalyse extrahiert werden können, sind diese zunächst ohne Kenntnisse über das verwendete Passwort oder den Schlüssel selbst nicht interpretierbar. Es gibt noch viele weitere steganografische Verfahren, deren Erforschung sich die Wissenschaft widmet. So zeigt beispielsweise [Wu et al., 2019] viele andere Möglichkeiten für das steganografische Verstecken von Daten in MIDI-Dateien auf. Zukünftige Arbeiten könnten sich daher mit der Erweiterung des Prototypen um weitere Verfahren sowie mit der Optimierung der bisher implementierten Technologien befassen. Konkrete Vorschläge für derartige Weiterentwicklungen werden in Abschnitt 5.3 Ausblick eingebracht. Durch die implementierten Verfahren und insbesondere deren Kombinationsmöglichkeiten lässt sich einerseits die Sicherheit der Daten und auch die Sicherheit der die Daten speichernden Personen erhöhen. Andererseits leiten sich aus dem testen und Evaluieren der Verfahren auch Ideen für die forensische Analyse ab. Dies wird in den Abschnitten 5.1.3 und 5.1.4 verdeutlicht.

5.3 Ausblick

Dieser Abschnitt befasst sich mit möglichen und sinnvollen Erweiterungen sowie Optimierungen von „StegaCrypto“. Er zeigt damit auch auf, wie zukünftige Arbeiten an die Ergebnisse dieses Projektes anknüpfen könnten. Als Optimierungen werden Softwareverbesserungen aufgezählt, die direkt an die bereits bestehenden Inhalte bzw. die bisher implementierten Verfahren anknüpfen und deren Qualität verbessern. Als Erweiterungen hingegen werden komplett eigenständige Verfahren aufgeführt, die in keinem direkten funktionalen Zusammenhang mit den bisher vorliegenden Softwarebestandteilen stehen und in Zukunft zur Steigerung des Funktionsumfangs integriert werden könnten.

5.3.1 Optimierungen

Wie bereits in Abschnitt 5.1.3 erläutert, hängt die Unauffälligkeit der RGB-Steganografie sehr stark vom Verteilungsintervall für die in das Verfahren eingebundenen Pixel der Trägergrafik ab. Auch das Größenverhältnis zwischen der Anzahl der zu versteckenden Bytes und der Pixelanzahl in der Trägergrafik spielt wie im selben Abschnitt beschrieben eine elementare Rolle. Beide Faktoren könnten sehr stark zugunsten einer besseren Verfahrensdiskretion beeinflusst werden, indem man die Verteilung der der zu versteckenden Daten über mehrere Trägerdateien beziehungsweise -grafiken hinweg erlaubt. Verteilt man eine zuvor in einer Grafik versteckte Datenmenge auf mehrere Grafiken, nimmt logischerweise der Anteil der steganografisch bearbeiteten Pixel am jeweiligen Gesamtbild stark ab. Außerdem können durch den Einsatz von vielen Trägergrafiken insgesamt so große Pixelmengen erzielt werden, dass die Unterbringung aller zu versteckenden Daten selbst bei äußerst großzügigen Verteilungsintervallen praktikabel bleibt. Ein weiterer Vorteil dieser Verfahrensoptimierung würde darin bestehen, dass sich eine Extraktion versteckter Daten durch unautorisierte Personen noch schwerer gestalten würde als bisher. Für einen erfolgreichen Angriff auf die in „StegaCrypto“ verfügbaren Verfahrenskombinationen müssten nicht nur eine Verschlüsselung mit AES-256 gebrochen und eine Steganografie erkannt werden, sondern man müsste für jeden versteckten Datensatz alle Trägerdateien in korrekter Reihenfolge ermitteln. Aus diesem Grund würde auch die MIDI-Steganografie von einer Verteilung über mehrere Dateien profitieren. Für die Realisierung dieser Optimierung könnte man den zu versteckenden Datensatz zuerst in Bytesequenzen fester Länge unterteilen, wobei die letzte Sequenz potentiell kürzer ausfällt. Danach könnte man jede Sequenz in einer separaten Grafik verstecken. Zur Kodierung der Reihenfolge der Trägerdateien könnten beim Versteckvorgang ein Farbwert innerhalb der RGB-Steganografie oder auch zwei „Program Change“-Ereignisse innerhalb der MIDI-Steganografie für die Kodierung eines Index von Null bis 255 verwendet werden. Dies ist natürlich nur ein Beispiel für eine mögliche Implementierung. Auch die bei der MIDI-Steganografie auftretenden Veränderungen der Dateigrößen könnten durch das Verstecken über mehrere Dateien hinweg reduziert werden, weil somit nicht alle zum Verstecken eingefügten Ereignisse in einer Datei gespeichert werden müssen.

Die zur Längenkodierung genutzten „Int32“-Werte können mit 2.147.483.647 Bytes für die RGB-Steganografie und 1.073.741.823 Bytes für die MIDI-Steganografie bereits sehr große Mengen verschlüsselter Daten verstecken. Es könnte dennoch sein, dass Bedarf an der Verarbeitung noch größerer Datenmengen besteht. Das ist insbesondere im Zusammenhang mit der im letzten Absatz geschilderten Optimierungsmöglichkeit vorstellbar. Um eine noch größere Anzahl von Bytes erfolgreich verarbeiten zu können, könnte man „Int64“ statt „Int32“ zur Längenkodierung einsetzen. Dazu müssten allerdings doppelt so viele Farbwerte innerhalb der RGB-Steganografie und doppelt so viele „Program Change“-Ereignisse innerhalb der MIDI-Steganografie für die Längenkodierung verwendet werden. Ein „Int64“-Wert besteht nämlich, wie bereits an der Bezeichnung zu erkennen ist, aus 64 Bits beziehungsweise acht Bytes. Eine Ganzzahl vom Typ „Int32“ umfasst nur halb so viele Bytes.

Um die Steganalyse noch zusätzlich zu erschweren, könnte man in Zukunft eine Verschachtelung steganografischer Verfahren erlauben. So wäre es beispielsweise möglich, eine Bilddatei per MIDI-Steganografie zu verstecken, in welcher zuvor verschlüsselte Daten per RGB-Steganografie versteckt wurden. Insbesondere nach der Ergänzung vieler weiterer Steganografieverfahren (siehe Abschnitt 5.3.2) könnten sich so unzählige Kombinationsmöglichkeiten ergeben. Durch derartige Verfahrensverschachtelungen wäre für unautorisierte Personen kaum nachvollziehbar, welche Daten auf welche Art und Weise kodiert wurden. Der erforderliche Aufwand für das Ausprobieren aller möglichen Kombinationen wäre immens.

Des Weiteren könnte die Steganalyse durch eine größere Variabilität im Ablauf der steganografischen Verfahren erschwert werden. So könnte eine Zufallsfunktion implementiert werden, welche aus mehreren möglichen Verteilungsmustern wählt und den Index des gewählten Musters an einer dafür reservierten Stelle in der Pixelgrafik oder der MIDI-Datei kodiert. Eine größere Variabilität erschwert die automatisierte Erkennung der steganografischen verfahren, da diese nicht immer die gleichen Abschnitte der Trägerdaten für den steganografischen Prozess manipulieren und sich somit keine deutlichen gemeinsamen Muster in den bearbeiteten Dateien herauskristallisieren. Darüber hinaus müsste für automatisierte Angriffe auf das Verfahren jeder vom Prototypen auswählbare Pfad für die Datenkodierung auf verdächtige Merkmale überprüft werden. Der Aufwand für die forensisch agierende Seite steigt dadurch enorm.

Wie bereits in Abschnitt 5.1.3 beschrieben, stellt die Kombination aus der für die RGB-Steganografie selektierten Farbkomponente und dem Farbverlauf der Trägergrafik einen wichtigen Faktor für die Verfahrensdiskretion dar. Eventuell wäre es daher sinnvoll, die Auswahl der zu verwendenden Farbkomponente über die UI zu ermöglichen. Außerdem könnte dadurch die Variabilität des steganografischen Prozesses gesteigert werden, weil nicht immer die gleiche Farbkomponente zur Kodierung versteckter Daten eingesetzt wird. Die antiforensische Bedeutung einer größeren Verfahrensvariabilität wurde bereits im letzten Absatz näher erklärt.

Um die Sicherheit der eingesetzten Kryptografie zu erhöhen, könnte man bestimmte Sicherheitskriterien für die verwendeten Passwörter einführen, darunter eine Mindestlänge. Da aus den Passwörtern unter Einsatz der Schlüsselableitungsfunktion PBKDF2 ein Schlüssel für den kryptografischen Prozess abgeleitet wird, stellen leicht zu erratende oder einfach abzuleitende Passwörter ein erhebliches Sicherheitsrisiko für die Verwendung des Prototypen dar. Eine derartige unsachgemäße Bedienung durch entsprechende Sicherheitsmechanismen bereits von Anfang an zu unterbinden, wäre daher sinnvoll.

Aktuell gibt die Bedienoberfläche nur sehr allgemeine Rückmeldungen zu den Erfolgen und Fehlschlägen der Verfahrensanwendungen. In Zukunft könnten diese Meldungen präzisiert werden. So könnte die Anwendung ganz konkret darauf hinweisen, dass das eingegebene Passwort nicht korrekt ist oder dass eine für die RGB-Steganographie selektierte Zieldatei nicht ausreichend Platz für die zu versteckenden Daten bereitstellen kann.

Der Programmcode von StegaCrypto könnte in der Zukunft in eine noch einfacher zu wartende sowie erweiternde Form gebracht werden. So enthält der Quellcode aktuell beispielsweise einige redundante Abschnitte, was den Programmcode etwas unübersichtlicher macht und außerdem den Wartungsaufwand vergrößert. Möchte man beispielsweise das Verteilungsintervall für die Steganografie global ändern, so müssen an vier unterschiedlichen Positionen des Quelltextes Änderungen vorgenommen werden. Durch eine Auslagerung der entsprechenden Abschnitte in eine externe Funktion, die dann an den gewünschten Stellen abgerufen werden kann, ließe sich diese Anpassung mit nur einer einzigen Veränderung im Quelltext vornehmen. Dies ist nur ein Beispiel für die möglichen Optimierungen. Natürlich verbessern derartige Maßnahmen weder die Funktionalität der Software noch deren Bedienung, aber der Entwicklungs- bzw. Programmieraufwand für zukünftige Projekte auf der Basis von „StegaCrypto 1.0“ kann dadurch signifikant reduziert werden.

Die Optik der Bedienoberfläche könnte in Zukunft ansprechender gestaltet werden. Aktuell erfüllt die UI zwar funktional vollständig ihren Zweck, jedoch verfügt sie über eine sehr minimalistische und monotone Optik. Es handelt sich bei diesem Erweiterungsvorschlag allerdings um eine rein ästhetische Optimierung ohne funktionalem Mehrwert.

5.3.2 Erweiterungen

Um die kryptografische Vielfalt des Prototypen zu erhöhen, könnten weitere Verschlüsselungen in „StegaCrypto“ integriert werden. Die in Abschnitt 2.1.3 geschilderten Grundlagen zum TwoFish-Algorithmus könnten sich dabei gegebenenfalls als hilfreich erweisen. Es sei an dieser Stelle jedoch mit aller Deutlichkeit darauf hingewiesen, dass eine Implementierung weiterer kryptografischer Algorithmen insbesondere bei Fehlen offizieller Bibliotheken für den jeweiligen Algorithmus äußerst anspruchsvoll ist. Weiterhin können bei der komplett selbständigen Programmierung kryptografischer Verfahren Fehler gemacht werden, die am Ende die Sicherheit der Chiffre massiv gefährden. Derartige Erweiterungen des Prototypen sollten also mit äußerster Vorsicht sowie viel Geduld entwickelt und hinzugefügt werden. Eine weitere Option für die Weiterentwicklung von „StegaCrypto“ besteht im Hinzufügen zusätzlicher steganografischer Verfahren. Da die Möglichkeiten im Bereich der Steganografie extrem vielfältig sind, werden in diesem Absatz nur einige bedeutsame Vertreter aufgeführt. So geht aus [Sadek et al., 2014] hervor, wie Daten steganografisch in Videodateien versteckt werden können. Die Quelle [Wu et al., 2019] erwähnt und demonstriert weitere Möglichkeiten für die MIDI-Steganografie. So wird beispielsweise veranschaulicht, wie für die Lautstärke zuständige „Velocity“-Werte zum Verstecken von Daten eingesetzt werden können. Aus der Quelle geht hervor, dass sich insbesondere MIDI-Daten mit vielen Simultannoten und einer hohen Variabilität der „Velocity“-Werte für den Versteckvorgang eignen. Durch das Einspielen von Werken mit geeigneter Notation und starken „Velocity“-Schwankungen als MIDI-Daten können somit hervorragend für das Verstecken geeignete Trägerdateien erschaffen werden. Um die großen Wertschwankungen zu erzielen, könnte man als Instrument z.B. ein Digitalpiano verwenden, welches die Lautstärke an die Anschlagstärke jeder Tastenbetätigung anpasst. Eine weitere Form von Steganografie geht aus [Aysan, 2017] sowie [Neatnik, 2021] hervor. Diese Quelle demonstriert die steganografische Ausnutzung von Textzeichen mit einer Zeichenbreite von Null. Durch das Einfügen derartiger Elemente in Zeichenketten können zusätzliche Informationen ohne optisch erkennbare Veränderungen übermittelt werden. Dieses Verfahren eignet sich daher auch besonders für die in Abschnitt 5.3.1 beschriebene Möglichkeit der Verschachtelung steganografischer Verfahren. Man könnte somit beispielsweise vortäuschen, dass ein per RGB-Steganografie extrahierter und anschließend mit AES-256 entschlüsselter Text gar keine brisanten, sondern lediglich besonders private Informationen enthält. Die Steganografie erhält dadurch eine Verteidigung in der Tiefe, also mehrere Sicherheitslinien/-hürden, die es für einen Angriff zu überbrücken gilt. Sollte der erste Täuschungsversuch überwunden werden, so wird zunächst der Eindruck vermittelt, es wäre auch der einzige.

Im Fokus der steganografischen Verfahren in dieser Arbeit steht die Täuschung der menschlichen Wahrnehmung. Ein interessanter Ansatz für zukünftige Projekte könnte darin bestehen, die „Wahrnehmung“ von Computersystemen zu täuschen. Eine Möglichkeit dafür wäre die Verkleinerung der Sektorenanzahl von Festplatten durch bestimmte Firmware-Kommandos, nachdem in den letzten Sektoren bestimmte Daten gespeichert wurden. Für

das Computersystem ist dieser Speicher dann zunächst unsichtbar und nicht ohne weiteres zugreifbar. Bei besonders hohen Sicherheitsanforderungen könnten derartige Techniken natürlich mit den bisher in „StegaCrypto“ integrierten Funktionen kombiniert werden. Da die Firmware für Speichermedien jedoch nicht standardisiert, in der Regel nicht offiziell dokumentiert und außerdem nicht einfach zugänglich und ansteuerbar ist, dürfte sich eine Erweiterung des Prototypen um derartige Mechanismen sowie eine Automatisierung derartiger Steganografie jedoch äußerst schwierig gestalten.

Der größte Angriffspunkt für die dateibasierten Verfahrenskombinationen in „StegaCrypto“ besteht vermutlich nicht in diesen Verfahren selbst, sondern in der Funktionsweise von Speichermedien und Datei- sowie Betriebssystemen. Es ist allgemein bekannt, dass vermeintlich gelöschte Daten durch IT-forensische Maßnahmen oft rekonstruierbar sind. In der Regel werden die von den scheinbar gelöschten Daten belegten Speicherstellen nicht verändert, sondern lediglich zur Überschreibung freigegeben und nicht länger vom Dateisystem adressiert. Der Inhalt einer gelöschten Datei steht somit vorerst weiter im Speicher. Werden neue Daten in den Speicher geschrieben, können diese gegebenenfalls den Inhalt der gelöschten Datei überschreiben, weil dieser Speicherbereich inzwischen freigegeben ist. Dieses Prinzip wird auch in [Hoffman, 2018] verdeutlicht. Um eine Umgehung der Krypto- und Steganografie durch Dateiwiederherstellung zu vermeiden, könnte man „StegaCrypto“ um Funktionen erweitern, welche die Originaldaten nach dem Verschlüsseln und Verstecken mit Nullen oder auch zufälligen Daten überschreiben. Eine besondere Schwierigkeit besteht hierbei in der Vielfalt von Dateisystem und Speichermedien, deren unterschiedliche Funktionsweise zu sehr unterschiedlichen notwendigen Maßnahmen für die erfolgreiche Spurenbeseitigung führt. Eine weitere Herausforderung besteht darin, dass auch betriebssystembedingt ungewollte Spuren der Originaldaten im Speicher liegen können. So können beispielsweise unter Windowssystemen Hinweise auf die Verwendung gelöschter Dateien in den sogenannten „Prefetches“ gespeichert sein [McQuaid, 2014]. Alle OS-bedingten Spuren für alle gängigen Betriebssysteme mit entsprechenden Implementierungen zu bereinigen, könnte sich als sehr aufwändig und anspruchsvoll erweisen. Es bleibt daher zu empfehlen,Daten wie bei den textbasierten Verfahrenskombinationen gleich bei ihrer Erzeugung mit entsprechenden Sicherheitsmechanismen im Speicher abzulegen. Wenn dies nicht praktikabel ist, sollten die Daten von einem nur dafür vorgesehenen externen Datenträger unter Einsatz der in „StegaCrypto“ zur Verfügung stehenden Verfahren übertragen werden. Der externe Datenträger ist danach komplett bitweise mit Nullen oder zufälligen Daten zu überschreiben. Auch eine komplette physische Zerstörung kann zur Antiforensik beitragen, ist jedoch weder eine nachhaltige noch besonders unauffällige Lösung. Die völlige Zerstörung eines Datenträgers führt zu analogen Spuren. Weiterhin existieren in der Regel Spuren auf Computersystemen, die auf die in der Vergangenheit angeschlossenen Geräte hindeuten. Bei Windowssystemen können beispielsweise in der Vergangenheit angeschlossene USB-Sticks über die Windows-Registrierung ermittelt werden [Chandel, 2020].

Um die Intuitivität der Softwarebedienung noch weiter zu verbessern, könnte die UI in Zukunft in unterschiedlichen Sprachen zur Verfügung gestellt werden. Die zu verwendende Sprache könnte dann über ein Dropdown-Menü ausgewählt werden. Vorerst wurde nur die englische Sprache verwendet, da diese als Weltsprache gilt und somit eine hohe internationale Praxistauglichkeit aufweist.

Auch eine steganografische Tarnung des Prototypen selbst wäre sicherlich äußerst nützlich. So könnte „StegaCrypto“ zukünftig um Funktionen ergänzt werden, welche die Software zunächst als ein völlig anderes Programm beziehungsweise eine völlig andere Datei tarnen. Das Programm könnte zum Beispiel vortäuschen, eine Excel-Tabelle zu sein und nur bei Betätigung einer bestimmten Tastenkombination die grafische Oberfläche von „StegaCrypto“ anzeigen.

5.4 Fazit

Insgesamt ist das Ergebnis dieser Arbeit extrem zufriedenstellend. Obwohl es sich nur um einen Prototyp handelt und von Grund auf eine Einarbeitung in die aufgegriffenen, implementierten und optimierten Verfahren sowie die genutzte Programmiersprache C# erfolgen musste, konnten alle drei Punkte der auf Seite 1 formulierten Zielstellung erfüllt werden. Durch den Einsatz von AES-256 als Verschlüsselungsverfahren kann der Prototyp die Sicherheit der Informationen und Daten gewährleisten. Die RGB- und MIDI-Steganografie wiederum sorgen bei einer ordnungsgemäßen Verwendung innerhalb der in den Abschnitten 5.1.3 und 5.1.4 geschilderten Grenzen für eine glaubwürdige Abstreitbarkeit von als Verfolgungsgrund dienendem Verhalten. Die Bedienoberfläche ermöglicht zudem eine unkomplizierte, einfache, schnelle und intuitive Bedienung. Das gilt auch für Personen, welche über keine fundierten Fachkenntnisse auf dem Gebiet der Informatik verfügen. Als zusätzliche Hilfestellung liegt der Software die im Anhang unter Abschnitt .4 beigefügte Bedienungsanleitung bei. Für kleinere Datenmengen ist die Software somit innerhalb des in Abschnitt 5.1.6 definierten Rahmens praxistauglich.

.1 AES-256

Tabelle .1: S-Box für AES. Quellen: [Kanal AppliedGo, 2017, 1:04 min], [Paar und Pelzl, 2011, S. 101], [Wikipedia Editierende, 2021a]

	00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0A**	**0B**	**0C**	**0D**	**0E**	**0F**
00	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
10	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
20	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
30	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
40	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
50	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
60	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
70	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
80	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
90	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A0	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B0	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C0	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D0	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E0	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F0	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

.2 MIDI-Steganografie

Tabelle .2: Liste diverser „Channel Voice Events“. Quelle: eigene Arbeit, angelehnt an [Birch, 2021b], [Hass, 2020], [John, 2004] und [MIDI Association, 2021]

Code	Name	Daten	Bedeutung
8_	Note-Off	Zwei Bytes: Notenbyte, Lautstärkebyte	Das Spielen der in den Ereignisdaten spezifizierten Note wird eingestellt.
9_	Note-On	Zwei Bytes: Notenbyte, Lautstärkebyte	Die in den Ereignisdaten spezifizierte Note wird gespielt.
A_	After Touch	Zwei Bytes: Notenbyte, Druckbyte	Während eine Note gespielt wird, schwankt ihre Lautstärke anhand des angegebenen Drucks, mit dem die zugehörigen Tasten des Instruments gehalten werden.
B_	Controller Change	Zwei Bytes: Einstellung, Einstellungswert	Eine gerätespezifische Einstellung wird modifiziert.
C_	Program Change	Ein Byte: Programmnummer	Es erfolgt die Auswahl des durch die Programmnummer spezifizierten Instrumentes.
D_	Channel Pressure	Ein Byte: Druck	Legt After-Touch mit angegebenem Druck für einen gesamten Channel fest. „Channel Pressure“ ist für Instrumente wichtig, die nicht für jede Taste einzeln einen Drucksensor besitzen.
E_	Pitch Wheel	2 Bytes (kombiniert zu einem 14-Bit-Wert)	Das „Pitch Wheel“ wird ausgerichtet.

.3 Implementierung sonstiger Funktionen

Tabelle .3: Freigabekriterien aller bei Programmstart deaktivierten Steuerelemente

Button	**Freigabekriterien**
„Browse...“-Button für Auswahl der Trägerdatei	Im Reiter „Texts“ müssen Krypto- und Steganografie ausgewählt sein.
„Browse...“-Button für Auswahl der Quelldatei	Im Reiter „Files“ müssen Krypto- und Steganografie ausgewählt sein.
„Browse...“-Button für Auswahl der Zieldatei	Im Reiter „Files“ müssen Krypto- und Steganografie ausgewählt sein.
„Encrypt“-Button im Reiter „Texts“	Nachrichten-, Trägerdatei- und Passwortfeld im Reiter „Texts“ dürfen nicht leer sein. Im Reiter „Texts“ müssen Krypto- und Steganografie ausgewählt sein.
„Decrypt“-Button im Reiter „Texts“	Trägerdatei- und Passwortfeld im Reiter „Texts“ dürfen nicht leer sein. Im Reiter „Texts“ müssen Krypto- und Steganografie ausgewählt sein.
„Encrypt“-Button im Reiter „Files“	Quelldatei-, Zieldatei- und Passwortfeld im Reiter „Files“ dürfen nicht leer sein. Im Reiter „Files“ müssen Krypto- und Steganografie ausgewählt sein. Ist „RGB - Manipulation of Color Values“ als Steganografie im Reiter „Files“ ausgewählt, muss eine Zieldatei im PNG-, BMP- oder GIF-Format vorliegen. Ist „MIDI - Insertion of Program Changes“ als Steganografie im Reiter „Files“ ausgewählt, muss eine Zieldatei im MIDI-Format vorliegen.
„Decrypt“-Button im Reiter „Files“	Quelldatei-, Zieldatei- und Passwortfeld im Reiter „Files“ dürfen nicht leer sein. Im Reiter „Files“ müssen Krypto- und Steganografie ausgewählt sein.

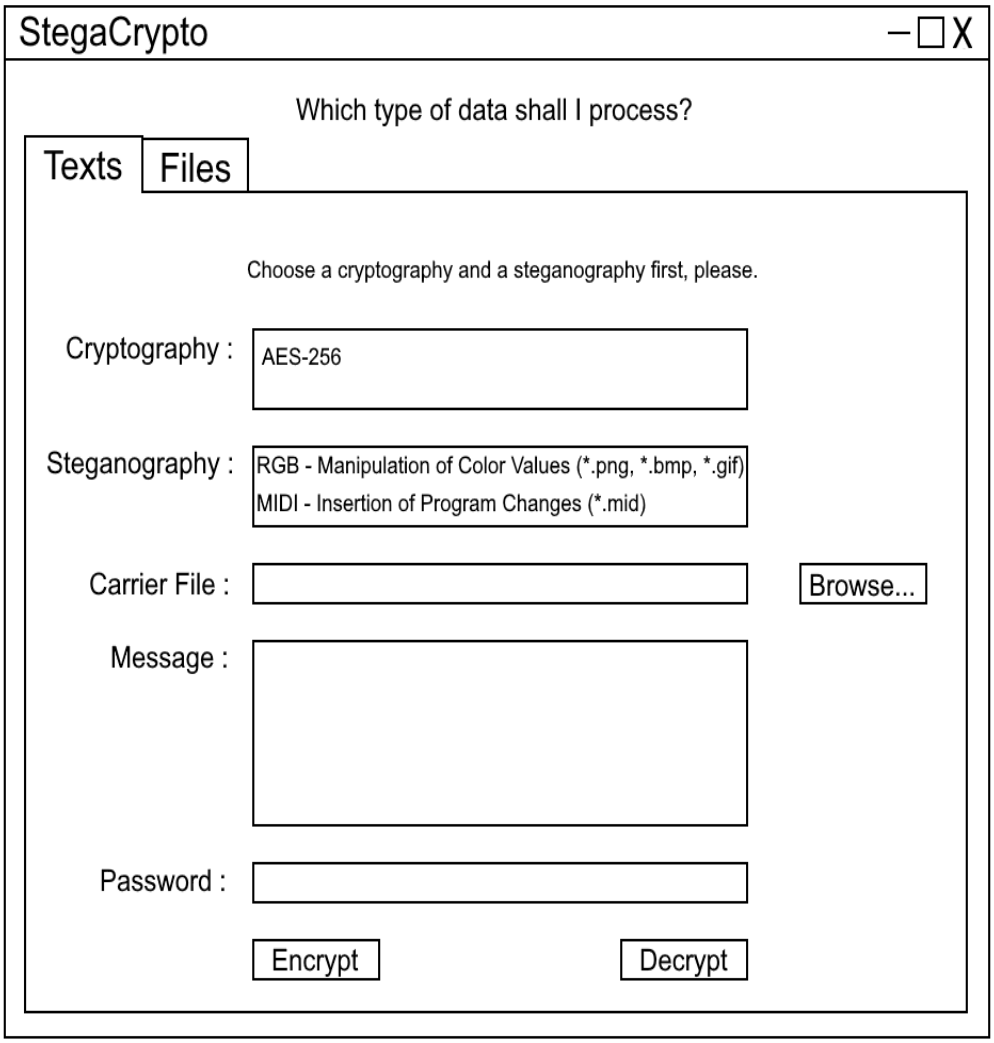
	Ist „RGB - Manipulation of Color Values“ als Steganografie im Reiter „Files“ ausgewählt, muss eine Quelldatei im PNG-, BMP- oder GIF-Format vorliegen. Ist „MIDI - Insertion of Program Changes“ als Steganografie im Reiter „Files“ ausgewählt, muss eine Quelldatei im MIDI-Format vorliegen.

.4 Bedienungsanleitung

StegaCrypto 1.0
Manual

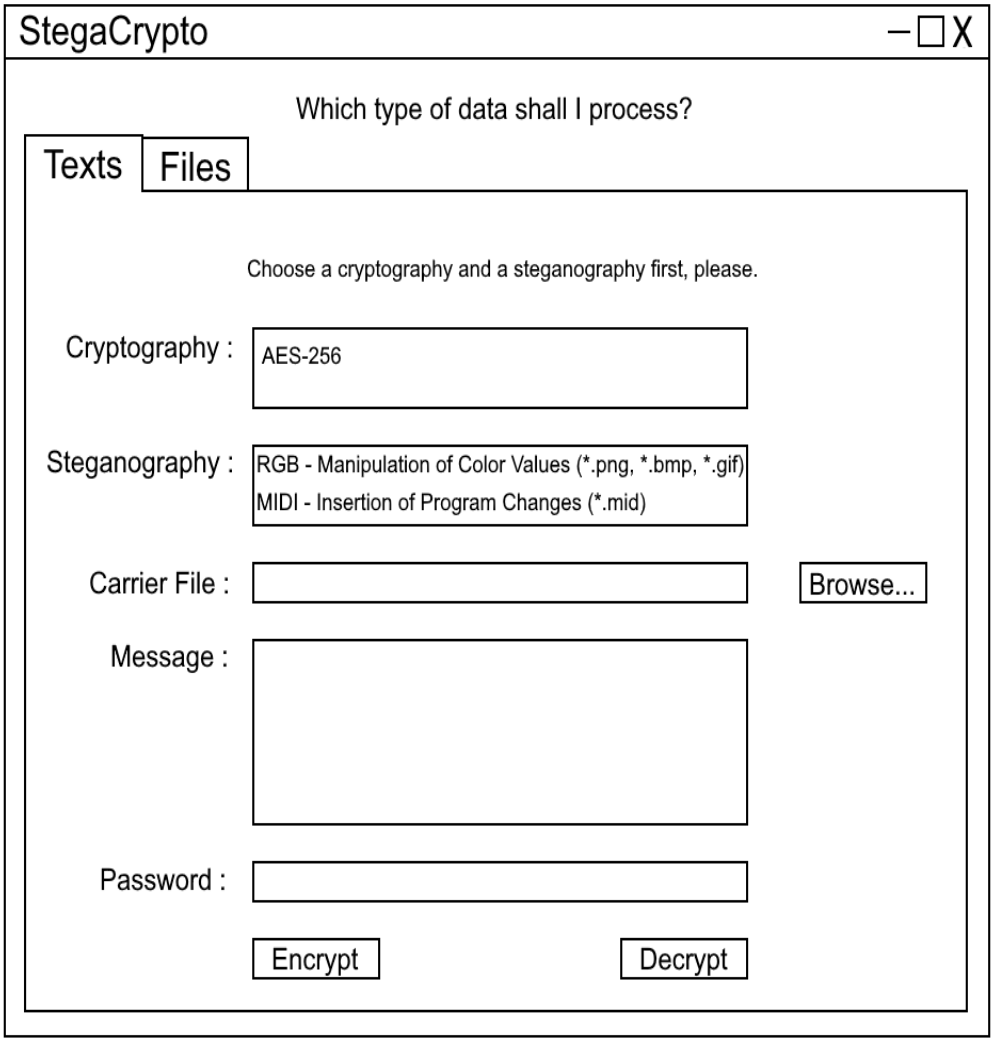
Encryption and Hiding of texts

1. click on the tab “Texts”
2. select one cryptographic method
3. select one steganographic method
4. provide a carrier file (the message of the next step will be stored here)
5. enter the message that you want to encrypt and hide
6. enter a password
7. click on “Encrypt”



Extraction and decryption of texts

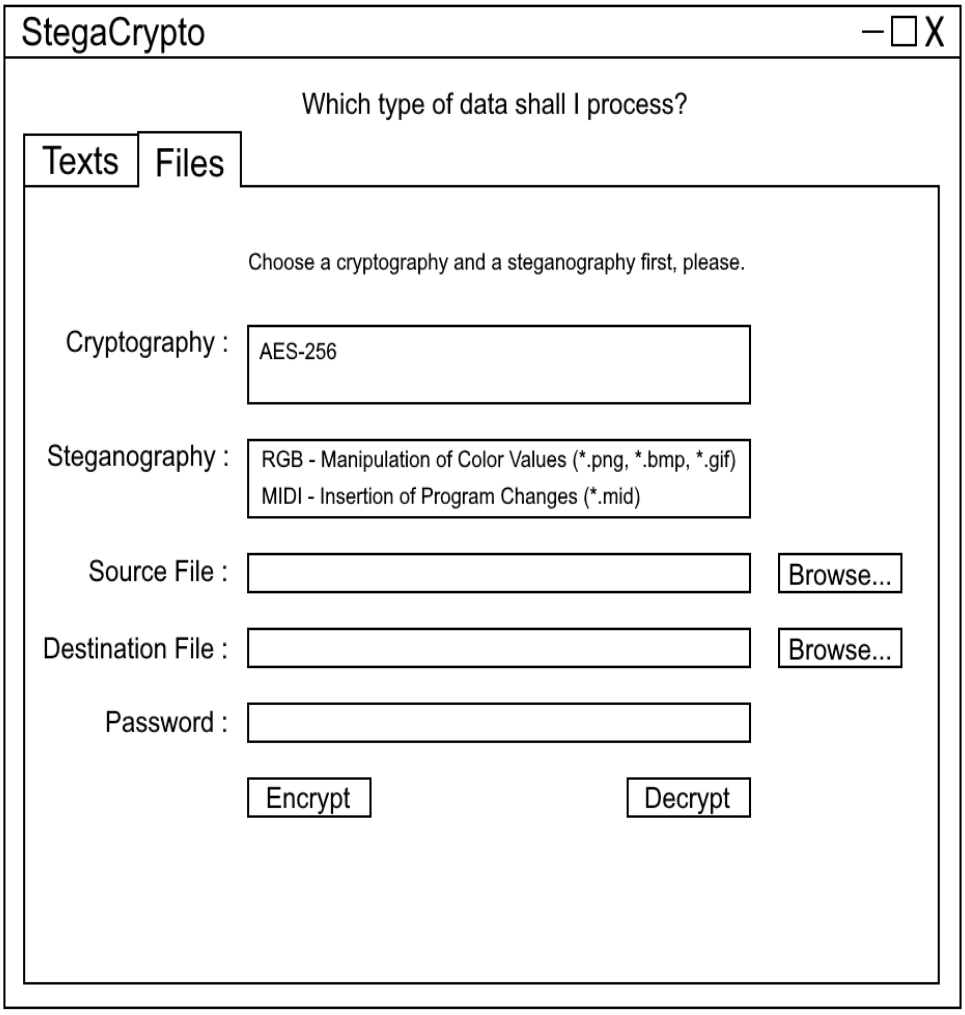
1. click on the tab “Texts”
2. select the previously used cryptographic method
3. select the previously used steganographic method
4. provide the carrier file with the hidden and encrypted data
5. enter the correct password
6. click on “Decrypt”



Encryption and Hiding of files

1. click on the tab “Files”
2. select one cryptographic method
3. select one steganographic method
4. provide a source file that you want to encrypt and hide
5. provide a destination file (the encrypted source file will be hidden here)
6. enter a password
7. click on “Encrypt”

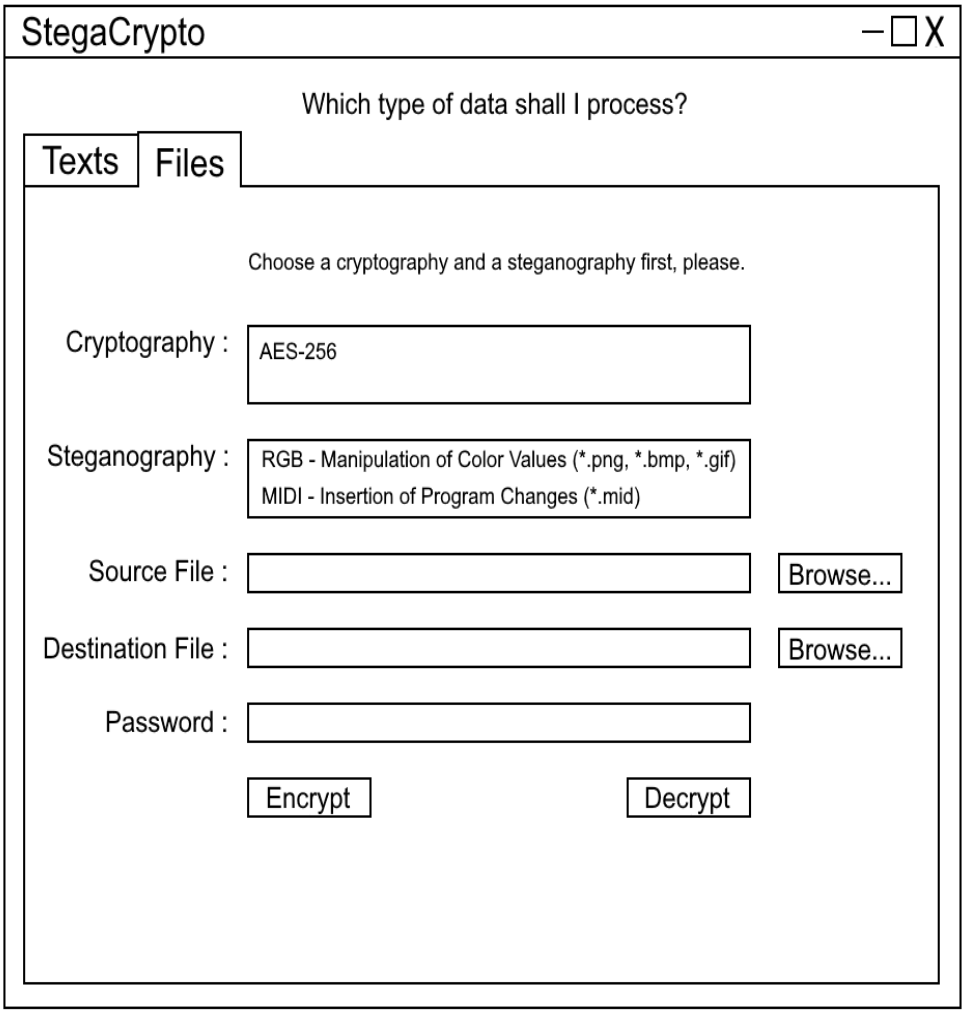
CAUTION: The destination file format has to be compatible with the chosen steganography. RGB steganography is compatible with *.png, *.bmp and *.gif. MIDI steganography is compatible with *.mid.



Extraction and decryption of files

1. click on the tab “Files”
2. select the previously used cryptographic method
3. select the previously used steganographic method
4. provide the source file with the hidden and encrypted data
5. provide a destination file (the extracted and decrypted file will overwrite it)
6. enter a password
7. click on “Decrypt”

CAUTION: The source file format has to be compatible with the chosen steganography. RGB steganography is compatible with *.png, *.bmp and *.gif. MIDI steganography is compatible with *.mid. The extracted file will overwrite your destination file but it will adopt its file format. That is why the destination file must be of the same type as the file that you want to extract.



Safety guidelines and further information

The author of StegaCrypto 1.0, its manual and the related master's thesis does not take any responsibility for any damage that is related to malicious use of the software or to any security flaws of its functionalities. Any usage of this software is done at one's own risk. Please keep in mind that StegaCrypto 1.0 is a prototype and further testing is required to fully evaluate its potential and its limits. When using the file-based functionalities, make sure to **safely delete** any original files. Otherwise the whole procedure of encrypting and hiding might be bypassed by using simple file restoration techniques. If you receive error messages though you completely follow the instructions in this manual, you are probably trying to hide too much data in too small files. The application of incorrect passwords when decrypting will also result in an error message.

Literaturverzeichnis

Announcing the ADVANCED ENCRYPTION STANDARD (AES). Technical report, November 2001. URL https://doi.org/10.6028/nist.fips.197.

Amnesty International. Torture, 2021a. URL https://www.amnesty.org/en/what-we-do/torture/. [Online; Stand 05. November, 2021].

Amnesty International. Report: China's mass internment, torture, and persecution of Muslims, 2021b. URL https://www.amnesty.org.uk/report-chinas-mass-internment-torture-and-persecution-muslims. [Online; Stand 05. November, 2021].

Z. Aysan. Zero-Width Characters, Invisibly fingerprinting text, 2017. URL https://www.zachaysan.com/writing/2017-12-30-zero-width-characters. [Online; Stand 21. Oktober, 2021].

S. A. Birch. The MIDI File Format, 2021a. URL https://www.personal.kent.edu/~sbirch/Music_Production/MP-II/MIDI/midi_file_format.htm. [Kent State University, Online; Stand 16. September 2021].

S. A. Birch. MIDI Channel Voice Messages, 2021b. URL https://www.personal.kent.edu/~sbirch/Music_Production/MP-II/MIDI/midi_channel_voice_messages.htm. [Kent State University, Online; Stand 04. November 2021].

N. Boughen. *LightWave 3D® 7.5 Lighting*. Wordware Publishing, Inc., 2320 Los Rios Boulevard, Plano, Texas 75074, USA, 2003. ISBN 1-55622-354-4.

R. Chandel. USB Forensics: Detection & Investigation, 2020. URL https://www.hackingarticles.in/usb-forensics-detection-investigation/. [Online; Stand 24. November, 2021].

J. Daemen und V. Rijmen. *The Design of Rijndael*. Springer Science+Business Media S.A., 14197 Berlin, Heidelberger Platz 3, 2002. ISBN 978-3-540-42580-9.

P. A. Grassi, J. L. Fenton, E. M. Newton, R. A. Perlner, A. R. Regenscheid, W. E. Burr, J. P. Richer, N. B. Lefkovitz, J. M. Danker, Y.-Y. Choong, K. K. Greene, und M. F. Theofanos. Digital Identity Guidelines: Authentication and Lifecycle Management. Technical report, Juni 2017. URL https://doi.org/10.6028/nist.sp.800-63b.

J. Hass. *Introduction to Computer Music*. Jacobs School of Music, USA, Indiana

University, Jacobs School of Music, 1201 East 3rd Street, 2020. URL https://cmtext.indiana.edu/index.php.

C. Hoffman. Why Deleted Files Can Be Recovered, and How You Can Prevent It, 2018. URL https://www.howtogeek.com/125521/htg-explains-why-deleted-files-can-be-recovered-and-how-you-can-prevent-it/. [Online; Stand 24. November, 2021].

C. John. Steganography V — Hiding Messages in MIDI Songs, 2004. URL https://www.codeproject.com/Articles/5390/Steganography-V-Hiding-Messages-in-MIDI-Songs. [Online; Stand 16. September 2021; Lizenz: CDDL 1.0, https://opensource.org/licenses/cddl1.php].

Kanal Abdullah AlQahtani. The Twofish Encryption Algorithm, 2015. URL https://www.youtube.com/watch?v=TNWksRzeQpI. [Online; Stand 28. August 2021].

Kanal AppliedGo. AES Rijndael Cipher explained as a Flash animation, 2017. URL https://www.youtube.com/watch?v=gP4PqVGudtg. [Online; Stand 11. August 2021; angelehnt an Zabala.2008].

Kanal D3NCE. Steganographie in C# - Part 2: Verschlüsselung, 2020. URL https://www.youtube.com/watch?v=SBx4Qc-Dc_A. [Online; Stand 8. September 2021].

P. Kathayat. How to Bypass Geo-Blocking and Internet Censorship?, 2020. URL https://techgeekers.com/how-to-bypass-geo-blocking-and-internet-censorship/#Tor_Browser. [Online; Stand 05. November, 2021].

A. Kerckhoff. La cryptographie militaire. *Journal des sciences militaires*, vol. IX, Januar, Februar 1883.

A. Kumar. Convert file to Byte array in c#, 2021. URL https://findnerd.com/list/view/Convert-file-to-Byte-array-in-c/2952/. [Online; Stand 24. Oktober 2021].

H. G. Liddel, R. Scott, H. S. Jones, und R. McKenzie. *A Greek-English Lexicon*. Oxford University Press, Vereinigtes Königreich, Oxford OX2 6DP, Great Clarendon Street, 1984. ISBN 978-0-19-864226-8.

M. Marlinspike und T. Perrin. The Sesame Algorithm: Session Management for Asynchronous Message Encryption. April 2017. [Revision 2].

J. McQuaid. Forensic Analysis of Prefetch files in Windows, 2014. URL https://www.magnetforensics.com/blog/forensic-analysis-of-prefetch-files-in-windows/. [Online; Stand 24. November, 2021].

MDN Web Docs Editierende. Image file type and format guide, 2021. URL https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Image_types. [Online; Stand 8. September 2021].

MIDI Association. Summary of MIDI 1.0 Messages, 2021. URL https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message. [Online; Stand 16. September 2021].

K. Moriarty, B. Kaliski, und A. Rusch. PKCS #5: Password-Based Cryptography Specification Version 2.1. RFC 8018, Januar 2017. URL https://datatracker.ietf.org/doc/html/rfc8018.

Neatnik. Steganographr, 2021. URL https://neatnik.net/steganographr/. [Online; Stand 24. November, 2021].

Oscar de Lama. RGB Cube, 2014. URL https://www.flickr.com/photos/roofwalker/13829769224/in/photostream/. [Online; Stand 8. September 2021; Lizenz: CC BY-NC 2.0, https://creativecommons.org/licenses/by-nc/2.0/].

C. Paar und J. Pelzl. *Understanding Cryptography*. Springer Science+Business Media S.A., 14197 Berlin, Heidelberger Platz 3, 2011. ISBN 978-3-642-04100-6.

F. Purnama. Bypass Censorship By TOR, 2019. URL https://hive.blog/censorship/@fajar.purnama/bypass-censorship-by-tor. [Online; Stand 05. November, 2021].

M. M. Sadek, A. S. Khalifa, und M. G. M. Mostafa. Video steganography: a comprehensive review. *Multimedia Tools and Applications*, 74(17), März 2014. doi: 10.1007/s11042-014-1952-z. URL https://doi.org/10.1007/s11042-014-1952-z.

B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, und N. Ferguson. Twofish: A 128Bit Block Cipher. Januar 1998.

R. Shah. Snowden: Encryption Is Critical to Protect Your Privacy, 2020. URL https://www.shortform.com/blog/edward-snowden-encryption/. [Online; Stand 05. November, 2021].

A. R. Smith. Alpha and the History of Digital Compositing. In *Microsoft Technical Memo #7*, 1995.

R. Stephens. Load images without locking their files in C#, 2014. URL http://csharphelper.com/blog/2014/07/load-images-without-locking-their-files-in-c/. [Online; Stand 28. November 2021].

M. S. Taha, M. S. M. Rahim, S. A. Lafta, M. M. Hashim, und H. M. Alzuabidi. Combination of Steganography and Cryptography: A short Survey. *IOP Conference Series: Materials Science and Engineering*, 518, jun 2019. doi: 10.1088/1757-899x/518/5/052003. URL https://doi.org/10.1088/1757-899x/518/5/052003.

M. S. Turan, E. Barker, W. Burr, und L. Chen. Recommendation for password-based key derivation, Part 1: Storage Applications. Technical report, 2010. URL https://doi.org/10.6028/nist.sp.800-132.

Wikipedia Editierende. Rijndael S-box — Wikipedia, The Free Encyclopedia, 2021a. URL https://en.wikipedia.org/w/index.php?title=Rijndael_S-box&oldid=1009834201. [Online; Stand 12. August 2021].

Wikipedia Editierende. AES key schedule — Wikipedia, The Free Encyclopedia, 2021b. URL https://en.wikipedia.org/w/index.php?title=AES_key_schedule&oldid=1035725805. [Online; Stand 11. August 2021].

Wikipedia Editierende. PBKDF2 — Wikipedia, The Free Encyclopedia, 2021c. URL https://en.wikipedia.org/w/index.php?title=PBKDF2&oldid=1037722647. [Online; Stand 3. September 2021].

D.-C. Wu, C.-Y. Hsiang, und M.-Y. Chen. Steganography via MIDI Files by Adjusting Velocities of Musical Note Sequences With Monotonically Non-Increasing or Non-Decreasing Pitches. *IEEE Access*, 7, Oktober 2019. doi: 10.1109/ACCESS.2019.2948493.

E. Zabala. Rijndael_Animation_v4_eng.exe, 2008. URL https://www.formaestudio.com/rijndaelinspector/. [Online; Stand 11. August 2021].

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 12. Januar 2022