Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Theoretische Informatik

Obfuskation in der Kryptographie

Masterarbeit

im Studiengang Informatik

von

Maria Buling
2788060

Prüfer:	**Prof. Dr. Heribert Vollmer**
Zweitprüfer:	**Dr. Arne Meier**
Betreuer:	**M. Sc. Martin Lück**

Hannover, 01.12.2016

Inhaltsverzeichnis

1 Einleitung

Die Aufgabe der Kryptographie ist unter anderem die Gewährleistung von Schutz geheimer Informationen. Erst seit kurzem wird versucht, auch Programmcode durch Verschleierung bzw. Obfuskation vor Missbrauch zu schützen [Bar+12]. Das Wort *Obfuskation* stammt aus dem Lateinischen und bedeutet „verschleiert“, „durcheinander“ oder „Dämmerung“. In der Kryptographie ist ein Obfuskator, intuitiv gesprochen, ein Algorithmus, der als Eingabe ein Programm P erhält und daraus ein neues Programm *O(P)* erzeugt, wobei die Funktionalität von P erhalten bleibt. $O(P)$ muss Geheimhaltung garantieren: der Code von $O(P)$ versteckt die Informationen des Ausgangsprogramms P.

Obfuskation ist ein Spezialgebiet der Kryptographie. In der allgemeinen Kryptographie will man die verschlüsselten Daten vollständig zurückgewinnen, aber genau kontrollieren, wer sie lesen darf. In der Obfuskation dürfen hingegen alle das obfuszierte Programm $O(P)$ lesen, aber man will genau kontrollieren, welche Informationen über P es preisgibt. Diese beiden Ansätze schließen sich nicht aus.

Bisherige Forschungsergebnisse zeigen, dass es noch keine Modelle gibt, die Obfuskation zufriedenstellende beschreiben, aber es ist noch nicht bekannt, ob es Klassen von Programmen gibt, die Ausnahmen darstellen könnten. In der Praxis wird Obfuskation dennoch angewendet, weil es ein Vorteil ist, den Aufwand des Angreifers wenigstens zu erhöhen.

Obfuskation von Programmcode wurde zum ersten mal 1997 von Tomborson und Kollberg [CTL97] erwähnt. Die Autoren dieser Arbeit forschten im Gebiet der Programmcodeverschleierung in erster Linie, um Eigentumsrechte für Algorithmen und kommerzieller Software zu ermöglichen, da immer mehr Programme durch die größere Verbreitung von z. B. Java nicht kompiliert wurden, sondern als Quellcode oder Bytecode ausgeliefert wurden. Auch in Computerviren wird Obfuskation angewendet, damit diese nicht von Antivirenprogrammen entdeckt werden.

Obfuskation wird auch dafür genutzt, um eine Verschlüsselung mit einem privatem Schlüssel in eine Verschlüsselung mit öffentlichem Schlüssel zu transformieren. Als öffentlicher Schlüssel dient ein obfusziertes Programm, das einen privaten Schlüssel enthält [Bar+12]. Dafür wird ein Verschlüsselungsverfahren mit einem geheimen Schlüssel gewählt. Das zu verschlüsselnde Programm, das mit diesem Schlüssel initialisiert ist, wird dermaßen verstrickt bzw. verschleiert, dass es sehr schwierig ist, den geheimen Schlüssel daraus zu extrahieren. Damit wird aus symmetrischer Verschlüsselung eine asymmetrische konstruiert.

In der theoretischen Informatik werden Programme üblicherweise als Turingmaschinen dargestellt, so dass man die δ-Funktion oder Übergangsfunktion obfusziert. In neuen Arbeiten wurden zusätzlich Boole'sche Schaltkreise als Berechnungsmodell verwendet [Bar+12; GK05].

Zwischen den theoretischen Anforderungen robuster Obfuskation von Programmen und der tatsächlich praktisch angewendeten Methoden ist eine große Kluft. Dennoch wurden schon viele vorgeschlagenen praktische Methoden zur Codeverschleierung realisiert.

Das Ziel dieser Arbeit ist, eine Übersicht der wichtigsten Fortschritte in der Forschung über Obfuskation zu geben. Die Autoren Barak u. a.[Bar+01; Bar+12] zeigten, dass es viele Möglichkeiten gibt, Obfuskation als robust zu definieren. Sie behandeln ausführlich das Konzept Obfuskation, insbesondere das sogenannte *virtuelle Black-Box-Modell*, das hier detailliert präsentiert wird.

Es heißt „Black-Box", weil ein Angreifer idealerweise nur Informationen berechnen kann, die er bereits durch Black-Box-Zugriff, d. h. Orakelzugriff, auf das Programm bekommt. Hierbei wird das obfuszierte Programm $O(P)$ ausgeführt und beobachtet. Ziel des Angreifers hierbei ist, das Ausgangsprogramm P aus den Ausgaben von $O(P)(x)$ zu rekonstruieren. Dementsprechend ist das Ziel des Eigentümers eines Programms P, das Programm zu verschleiern.

Darauf folgt das Forschungsergebnis von Kuzurin u. a. [Kuz+07] mit ausführlicher Darstellung einer geänderten Definition für Robustheit einer Obfuskation zum *virtuellen Grey-Box-Modell*. Bei diesem Modell sind die Anforderung an die Robustheit, im Vergleich zu denen im virtuellen Black-Box Modell, abgeschwächt. Das obfuszierte Programm wird, wie auch bei dem Black-Box-Modell, ausgeführt. Der Angreifer kann nicht nur auf die Tupel (Eingabe, Ausgabe) eines obfuszierten Programms $O(P)$ zugreifen, sondern auch auf den Berechnungspfad π von $P(x)$. Aus π könnten wichtige Informationen für die Rekonstruktion ermittelbar werden.

Eine Voraussetzung dafür, dass Obfuskation sinnvoll ist, ist, dass die zu obfuszierenden Klassen C von Programmen c nicht *erlernbar* sind. Das heißt, dass es keinen Algorithmus geben darf, der jedes $c \in C$ in Polynomialzeit identifizieren kann. Valiant hat die sogenannten *Learning Theory* erforscht, die hier vorgestellt wird [Val84]. Er hat ein Lernprotokoll konstruiert, das bestimmte Klassen von Programmen erlernen kann. Das bedeutet, dass es nichtobfuszierbare Klassen von Programmen gibt.

1.1 Von der Kryptographie zur Obfuskation

Kryptographie ist die Wissenschaft der Datensicherung unter Anwendung mathematischer Verfahren. Vertrauliche Daten können gespeichert, über unsichere Netze übertragen und nur von dem eigentlichen Empfänger gelesen werden.

Daten können stark oder schwach verschlüsselt werden. Das Maß für die Stärke der Verschlüsselung eines Textes ist die Zeit, die zur Entschlüsselung des Textes benötigt wird. Um einen starken kryptographischen Code, auch ein chiffrierter Text genannt, zu entschlüsseln, müssen geeignete Decodierungsverfahren angewendet werden.

Ein Verschlüsselungsalgorithmus ist eine Funktion zur Ver- und Entschlüsselung, der mit einem Schlüssel einen Klartext in einen chiffrierten Text verwandelt. Derselbe Klartext kann durch Verschlüsselung mit unterschiedlichen Schlüsseln unterschiedlich chiffrierten Text ergeben. Die Sicherheit der verschlüsselten Daten ist von der Stärke des Verschlüsselungsalgorithmus und der Geheimhaltung des Schlüssels abhängig.

Symmetrische Verschlüsselung

Bei der Verschlüsselung mit *Geheimschlüsseln* oder *symmetrischen Schlüsseln* wird ein einzelner privater Schlüssel sowohl für die Ver- als auch die Entschlüsselung verwendet. Diese Verfahren sind schnell und bei entsprechend langen Schlüsseln bieten sie auch eine hohe Sicherheit.

Ein ganz einfaches Beispiel dafür ist ist ein Ersetzungschiffriercode. Dabei werden Bestandteile der Daten gegeneinander ausgetauscht, z. B. durch Vertauschen einzelner Buchstaben im Alphabet. Dabei wird das Alphabet verschoben, wobei der Schlüssel die Anzahl der Zeichen ist, um die das Alphabet verschoben wurde. Der sichere Schlüsselaustausch zwischen den Kommunikationspartnern ist eines der vielen Probleme der Kryptographie. Mit der asymmetrischen Kryptographie versucht man dieses Problem zu lösen. Weil die asymmetrische Kryptographie weit komplexere Verfahren umfasst, kombinieren die übliche kryptographischen Protokolle sowohl symmetrische als auch asymmetrische Verfahren.

Asymmetrische Verschlüsselung

Kryptographie mit öffentlichen Schlüsseln ist ein asymmetrisches Schema, bei dem zur Verschlüsselung statt eines Schlüssels ein Schlüsselpaar verwendet wird, die beiden Schlüssel hängen mit einem mathematischen Algorithmus eng zusammen. Mit einem öffentlichen Schlüssel werden Daten verschlüsselt, und mit dem dazugehörigen privaten bzw. geheimen Schlüssel werden Daten entschlüsselt. Der

öffentliche Schlüssel ist allen bekannt, der private Schlüssel dagegen bleibt geheim. Es ist nicht bekannt, ob es effiziente Verfahren gibt, die es ermöglichen, den privaten Schlüssel aus dem öffentlichen abzuleiten.

Jeder kann mit einem öffentlichen Schlüssel die Daten verschlüsseln, aber nicht entschlüsseln. Nur die Person, die den entsprechenden privaten Schlüssel kennt, kann die Daten entschlüsseln.

Der große Vorteil der Verschlüsselung mit öffentlichen Schlüsseln liegt darin, dass Nachrichten sicher ausgetauscht werden können, ohne dass man sich vorher absprechen muss. Das Übertragen von geheimen Schlüsseln zwischen Absender und Empfänger über einen sicheren Kanal ist nicht mehr notwendig. Für jede Kommunikation sind nur noch öffentliche Schlüssel erforderlich, private Schlüssel werden dagegen nicht übertragen oder gemeinsam verwendet. Ein Beispiel für Verschlüsselungssysteme mit öffentlichen Schlüsseln ist die RSA-Verschlüsselung, nach den Erfindern Ron Rivest, Adi Shamir und Leonard Adleman benannt.

Asymmetrische Verschlüsselung und Einwegfunktionen

Bei der asymmetrischen Verschlüsselung wählt man eine einfach berechenbare Funktion, deren Umkehrung dagegen sehr aufwendig sein muss, eine so genannte *Einwegfunktion*. Sie ist ein wichtiges kryptographisches Werkzeug.

Eine Einwegfunktion $f : X \to Y$ ist eine Funktion mit folgenden Eigenschaften:

- Die Berechnung des Funktionswerts $y = f(x)$ ist für jedes $x \in X$ „einfach“, das bedeutet, es gibt einen Algorithmus, der y in Polynomialzeit berechnen kann.
- Allerdings gibt es für jedes $y \in Y$ (bis auf vernachlässigbare viele) für die Berechnung der Umkehrung von $f^{-1}(y) = x$ keinen Algorithmus, der in Polynomialzeit läuft. Somit benötigt die Berechnung der Inverse von y einen hohen Rechenaufwand. Man sagt auch, die Umkehrung von f ist „schwer“ berechenbar.

Die Verschlüsselung mit Hilfe einer Einwegfunktion erfolgt mit öffentlichem Schlüssel. Es ist für einen Angreifer sehr schwierig aus der Einwegfunktion den verschlüsselten Klartext zu gewinnen, da die Berechnung der Umkehrung mit sehr hoher Laufzeit verbunden ist.

Eine Beispiel dafür ist das Multiplizieren von Primzahlen. Es ist kein Problem, die Multiplikation zu berechnen. Es ist jedoch kein effizienter Algorithmus bekannt, der in akzeptabler Zeit das Primzahlprodukt in seine Faktoren zu zerlegt. Man spricht von Faktorisierung und in dem Zusammenhang vom Faktorisierungsproblem.

Es ist nicht bekannt, ob es Funktionen gibt, die die Eigenschaften der Einwegfunktion erfüllen. Der Beweis für die Existenz von Einwegfunktionen würde bedeuten, dass $P \neq NP$ ist. Jedoch folgt aus $P \neq NP$ nicht, dass es Einwegfunktionen gibt. Es ist in den üblichen Definitionen erlaubt, die Umkehrung mit Hilfe eines probabilistischen Algorithmus zu bestimmen.

Ein wichtiges Beispiel für eine Einwegfunktion sind die so genannten *idealen Hashfunktionen*. Eine spezielle Art der Einwegfunktion ist die Obfuskation.

2 Formale Grundlagen

In dieser Arbeit werden folgenden formale Begriffe benötigt und eingeführt. Sie entstammen hauptsächlich aus dem Skript von Nickelsen [Nic07] und aus dem Buch von Vollmer [Vol13].

Definition 2.1 (Boole'sche Funktion)**.** *Eine* Boole'sche Funktion *ist eine Funktion* $f : \{0,1\}^n \to \{0,1\}^r$*, dabei sind* $n, r \in \mathbb{N}$.

Definition 2.2 (Familie Boole'scher Funktionen)**.** *Eine* Familie Boole'scher Funktionen *ist eine Folge* $f = (f^n)_{n\in\mathbb{N}}$*, dabei ist* f^n *eine Boole'sche Funktion.*

Definition 2.3 (Orakel-Turingmaschine)**.** *Eine* Orakel-Turingmaschine *(OTM) ist eine Turingmaschine mit einem speziellen Band, dem sogenannten* Orakel- *oder* Fragenband*. Eine OTM hat zwei spezielle Zustände* q_{frage} *und* $q_{antwort}$*. Eine OTM* M *arbeitet bei Eingabe* x *mit einer Orakelfunktion* $f : \{0,1\}^* \to \{0,1\}^*$ *wie folgt: Wenn* M *im Zustand* q_{frage} *ist und ein Wort* y *auf dem Orakelband steht, so enthält in der nächsten Konfiguration das Orakelband* $f(y)$ *und* M *ist im Zustand* $q_{antwort}$*.*

Für eine OTM M *und ein Orakel* f *gilt:* $M^f(x) = y$*, falls* M *bei Eingabe* x *mit Orakel* f *die Ausgabe* y *hat.*

Definition 2.4 (Boole'scher Schaltkreis)**.** *Ein Boole'scher Schaltkreis ist ein* 3*-Tupel* $C = (V, E, \beta)$*. Der gerichtete azyklische Graph* $G = (V, E)$ *zerfällt in drei disjunkte Teilmengen von Eingabe-, Gatter- und Ausgabeknoten. Die einzelnen Knotenmengen sind durchnummeriert:* $V = \{e_1, \ldots, e_n, g_1, \ldots g_s, a_1, \ldots, a_r\}$*. Dabei sind* $e_1, \ldots, e_n$ *Eingabeknoten,* $g_1, \ldots, g_s$ *Gatterknoten,* $a_1, \ldots, a_r$ *Ausgabeknoten.* $\beta(g_i)$ *ist stets eine Boole'sche Funktion* $\wedge, \vee$ *oder* $\neg$*.*
Ein Boole'scher Schaltkreis wird hier im Weiteren als Schaltkreis bezeichnet.

Definition 2.5 (Schaltkreisgröße und Tiefe)**.** *Für einen Schaltkreis* C *mit* $V = \{e_1, \ldots, e_n, g_1, \ldots g_s, a_1, \ldots, a_r\}$ *ist die Größe des Schaltkreises* $Size(C) = s$ *die Anzahl der Gatter und die Tiefe des Schaltkreises* $Depth(C)$ *die maximale Anzahl von Gattern auf einem Pfad von irgendeinem Eingabeknoten* e_i *zu irgendeinem Ausgabeknoten* a_j*.*

Definition 2.6 (Von einem Schaltkreis berechnete Funktion)**.** *Ein Schaltkreis* C *mit* n *Eingabeknoten und* r *Ausgabeknoten berechnet eine* $n-$*stellige Funktion*

$\Phi_C : \{0,1\}^n \to \{0,1\}^r$. *Sie ist definiert durch die rekursive Zuordnung von Boole'schen Werten zu Gattern bei gegebener Eingabe* $\vec{x} = x_1, \ldots, x_n$. *Für* $\Phi_C(x)$ *schreibt man auch* $C(x)$.

Definition 2.7 (Wahrscheinlichkeitsverteilung)**.** *Eine Wahrscheinlichkeitsverteilung über einer Menge* S *ist eine Funktion* $D : S \to [0,1]$, *so dass* $\sum_{s \in S} D(s) = 1$. *Hier schreiben wir statt einer Wahrscheinlichkeitsverteilung auch die unterliegende Menge und meinen die Verteilung, die jedem Element eine gleiche Wahrscheinlichkeit zuordnet (uniforme Wahrscheinlichkeitsverteilung).*

Definition 2.8 (Wahrscheinlichkeit)**.** *Sind* $D_1, \ldots, D_n$ *Wahrscheinlichkeitsverteilungen und* π *ein Prädikat, dann bezeichnet*

$$\Pr_{s_1 \leftarrow D_1, \ldots, s_n \leftarrow D_n}[\pi(s_1, \ldots, s_n)]$$

die Wahrscheinlichkeit für zufällig gezogene $s_1, \ldots, s_n$ *dafür, dass* π *gilt.*

Definition 2.9 (Probabilistische Turingmaschine)**.** *Eine probabilistische Turingmaschine (PTM) ist eine Turingmaschine mit einem zusätzlichen Band, dem* Zufallsband. *Für eine Eingabe* x *und ein genügend langes Wort* r *ist* $M(x,r)$ *die Ausgabe, die entsteht, wenn* M *mit Eingabe* x *und dem Wort* r *auf dem Zufallsband rechnet.*

Ein Wort r *ist genügend lang, wenn es so lang ist, wie der Zeitbedarf der Rechnung von* M *auf* x.

Die Zufallsbänder werden der Übersicht halber in der Arbeit weggelassen, wenn die Wahrscheinlichkeit angegeben wird:

$$\begin{matrix}\Pr[M(x)\cdots] := \\ \vdots\end{matrix} \quad \begin{matrix}\Pr \\ \vdots \\ r \leftarrow \{0,1\}^{t(|x|)}\end{matrix} \quad [M(x,r)\cdots]$$

Dabei ist t die Laufzeit von M.

Hat eine PTM zusätzlich einen Orakelzugriff auf eine Funktion f, dann wird sie OPTM genannt und die Ausgabe $M^f(x,r)$ bzw. $M^f(x)$ analog zu Definition 2.3 definiert.

Definition 2.10 (Probabilistische Polynomialzeit-Turingmaschine)**.** *Eine Probabilistische Polynomialzeit-Turingmaschine (PPT) bzw. Probabilistische Polynomialzeit-Orakelmaschine (OPPT) ist wie eine PTM bzw. OPTM definiert, läuft aber zusätzlich in Polynomialzeit.*

Definition 2.11 (Vernachlässigbare Funktion)**.** *Eine Funktion* $\nu : \mathbb{N} \to \mathbb{N}$ *gilt als* vernachlässigbar, *wenn sie langsamer wächst als der Kehrwert jedes Polynoms. Das heißt, für jedes positive Polynom* $p(\cdot)$ *existiert ein* $n_0 \in \mathbb{N}$, *sodass* $\nu(n) < 1/p(n)$ *für jedes* $n > n_0$, *also gilt* $\nu \in o(1/p)$ *für alle Polynome* p.

Definition 2.12 (Einwegfunktionen)**.** *Eine Einwegfunktion ist eine Funktion* $f : \{0,1\}^* \to \{0,1\}^*$*, die deterministisch in Polynomialzeit berechenbar ist, aber nicht effizient umkehrbar: Für jede PPT* I *(Invertierer) gilt, dass*

$$\Pr\left[f(I(f(x))) = f(x)\right]$$

für zufällige x *vernachlässigbar sein muss.*

3 Obfuskation

Die Theorie der Obfuskation hat ihren Ursprung in den frühen Arbeiten zu maschinellem Lernen, insbesondere im Konzept der *Lerntheorie*, etwa bei Leslie Valiant [Val84] und Dana Angluin [Ang92]. Die von Valiant [Val84] eingeführte Lerntheorie beschreibt das theoretische Problem der Lernbarkeit von so genannten *Konzepten* aus vorgegebenen Daten, etwa ein bestimmtes Element $x \in X$ einer vorgegebenen Menge.

Er definiert das grundlegende Modell der so genannten *Lernmaschinen*. Eine Lernmaschine besteht aus zwei Komponenten:

- Ein *Lernprotokoll* (engl. *learning protocol*) bestimmt die Art und Weise, wie Informationen aus der Welt gewonnen werden. Diese werden durch einen *Lehrer* bereitgestellt, der etwa ein Mensch oder ein anderes Programm sein kann.

- Ein *Ableitungsverfahren* (engl. *deduction procedure*) ist ein Algorithmus, der mit Hilfe des Protokolls das Zielkonzept ableitet.

Valiant betrachtete insbesondere binäre Konzepte P, die über einer bestimmten Datenmenge V definiert sind. Für diese Art von Konzepten legt Valiant folgendes Lernprotokoll fest. Es liefert zwei Arten von Informationen. Zum einen hat der Lernende Zugang zu einem Angebot von „typischen“ Daten. Ein Anruf der Routine Examples erfragt ein solches zufälliges positives Beispiel $v \in P$.
Die zweite Quelle der Informationen, die verfügbar ist, ist eine Routine Oracle. Ein Aufruf der Routine Oracle wertet aus, ob ein beliebiges übergebenes Datum v zu dem Konzept passt, das heißt, ob $v \in P$. Oracle gibt also die Werte „wahr“ oder „falsch“ zurück.

Valiants Lernprotokoll

Valiant betrachtete für ein solches Konzept Boole'sche Funktionen. Gegeben sei eine Wahrscheinlichkeitsverteilung D über einer Menge X von Boole'schen Funktionen F über den Variablen $p_1, \ldots, p_t$, die den Wert 1 oder 0 annehmen können. Ein *Datum* v ist eine Zuordnung eines Wertes $\{0, 1, *\}$ zu jeder der t Variablen, wobei $*$ für eine unbestimmte Variable steht. Ein Datum ist *total*,

wenn jeder Variable ein Wert aus $\{0, 1\}$ zugewiesen wurde. Damit wir Boole'sche Funktionen als Konzepte über solchen Daten auffassen können, definieren wir $F(v) = 1$, falls $F(v') = 1$ gilt für alle totalen v', die aus v durch Ersetzen von $*$ entstehen.

Ein Beispiel für ein Konzept ist $F = (a_1 \vee a_2) \wedge (a_4 \vee a_1)$. Ein möglicher Ablauf einer Lernmaschine könnte wie folgt sein.

$$
\begin{aligned}
&\textsc{Examples}() \\
&\quad \hookrightarrow \{a_1 = 0,\, a_2 = 1,\, a_3 = *,\, a_4 = 1\} \\
&\textsc{Oracle}(a_1 = 0,\, a_2 = 1,\, a_3 = 0,\, a_4 = 1) \\
&\quad \hookrightarrow \text{wahr} \\
&\textsc{Oracle}(a_1 = 0,\, a_2 = 1,\, a_3 = 1,\, a_4 = 1) \\
&\quad \hookrightarrow \text{wahr}
\end{aligned}
$$

Eine Klasse X von Konzepten ist *lernbar*, wenn es eine PPT A (Ableitungsalgorithmus) gibt, der für alle $F \in X$ bei Eingabe 1^n:

- Das obige Protokoll verwendet, das heißt Orakelzugriff auf die Routinen Examples und Oracle hat.
- F näherungsweise lernt, das heißt, ein $G \in X$ ausgibt, das F mit Wahrscheinlichkeit mindestens $1 - \frac{1}{n}$ entspricht (bei einem zufällig gewählten Eingabevektor v).

Man kann die zu lernenden Konzepte von Boole'schen Funktionen auf Programme erweitern, die diese berechnen. Da bei der Obfuskation Programmcode verschleiert werden soll, sind Lernbarkeit und Obfuszierbarkeit widersprüchliche Eigenschaften in folgendem Sinne. Ist eine Klasse von Programmen lernbar, bedeutet dies, dass keine Informationen über die Programme v geheim gehalten werden können. Damit ist es sinnlos, nach einer Obfuskation für diese Programme zu suchen.

Valiant stellt einige Klassen von lernbaren „Programmen“ vor. Dazu gehören einige Klassen von aussagenlogischen Formeln, wie k-CNF, monotone DNF und so genannte μ-Ausdrücke [Val84].

3.1 Moderne Obfuskationsmodelle

Die Suche nach einem Modell für Obfuskation, das auch moderne kryptographische Erkenntnisse berücksichtigt, ist immer noch Gegenstand aktueller Forschung.

Im Allgemeinen Fall ist ein Obfuskator ein Algorithmus O, der als Eingabe ein Programm P bekommt. Dieses wird zu einem Programm $O(P)$ modifiziert, so dass folgende Anforderungen erfüllt sind:

Funktionalität: Die Funktionalität der Programme P und $O(P)$ ist äquivalent, das heißt, sie berechnen die gleiche Funktion.

Effizienz: Sowohl die Größe als auch die Ausführungszeit von $O(P)$ weicht nicht zu stark von P ab.

Robustheit: Das generierte Programm $O(P)$ muss so unverständlich wie möglich sein.

Das Programm $O(P)$ wird Obfuskation von P genannt.

Der Schwerpunkt der Forschung im Gebiet der Obfuskation lag in den letzten Jahren darin, Modelle zu entwerfen, die verschiedenen Vorstellungen der obigen Punkte umsetzen, bzw. darin, ob es diese Obfuskatoren in der Realität geben kann. Die grundlegende Definition von Obfuskation stammt von Barak u. a. [Bar+12], der die drei Anforderungen mit Hilfe von probabilistischen Maschinen formalisiert hat. Die obfuszierten Programme werden im folgenden als deterministische Turingmaschinen oder als Bool'sche Schaltkreise betrachtet.

Definition 3.1 (Turingmaschinenobfuskator:)**.** *Eine PTM O ist ein* Obfuskator *für TMs, wenn folgende Eigenschaften erfüllt sind:*

- *Funktionalität: Für jede TM M beschreibt jede Ausgabe $O(M)$ eine TM, die die gleiche Funktion wie M berechnet.*
- *Effizienz der Obfuskation: Die Programmlänge und die Ausführungszeit von $O(M)$ ist höchstens polynomiell bezüglich M. Das heißt, es gibt ein Polynom p, sodass für jede TM M und für jedes $O(M)$ gilt: $|O(M)| \leq p(|M|)$, und wenn M bei Eingabe x nach t Schritten hält, dann hält $O(M)$ auf x nach nicht mehr als $p(t)$ Schritten.*

Definition 3.2 (Schaltkreisobfuskator:)**.** *Eine PTM O ist ein Obfuskator für Boole'sche Schaltkreise, wenn folgende Eigenschaften erfüllt sind:*

- *Funktionalität: Für jeden Boole'schen Schaltkreis C beschreibt die Ausgabe $O(C)$ einen Boole'schen Schaltkreis, der die gleiche Funktion berechnet.*
- *Effizienz der Obfuskation: Es gibt ein Polynom p, so dass für jeden Boole'schen Schaltkreis C gilt: $|O(C)| \leq p(|C|)$.*

Die Definition der Robustheit wurde von vielen Autoren variiert und abgeschwächt, wie z. B. von Canetti, Goldreich und Halevi, Goldwasser und Rothblum, Kuzurin u. a. und Garg u. a. [CGH04; GR07; Kuz+07; Gar+16]. Die Robustheit einer Obfuskation hängt von den konkreten Anwendungsfällen ab. Sie wird jedoch meistens durch ein „Spiel“ zwischen einem so genannten *Angreifer* und einem *Simulator* dargestellt. Der Angreifer berechnet irgendeine Eigenschaft eines obfuszierten Programms $O(P)$. Der Simulator versucht die Rechnung des Angreifers nachzuvollziehen, erhält aber statt $O(P)$ nur Orakelzugriff auf P. Die Idee ist, dass, wenn der Simulator nur durch Orakelzugriffe die gleiche Berechnung durchführen kann, dann der Angreifer A offensichtlich nur aus den Ausgaben des Programms Informationen gewinnen kann.

3.2 Klassifikation der Obfuskationsmodelle

In der Arbeit von Barak u. a. [Bar+12] werden verschiedene Definitionen der Robustheit der Obfuskation vorgestellt. Es kann keine universelle Definition geben, da es verschiedene Anforderungen an Obfuskation gibt.

Es muss nicht immer der gesamte Programmcode obfusziert werden. Wenn etwa ein Verschlüsselungsalgorithmus mit privaten Schlüssel obfusziert werden soll, dann muss nicht das gesamte Programm, sondern nur der private Schlüssel verschleiert werden [Kuz+07]. Für folgende und andere Anwendungen wurden Obfuskationsmodelle entworfen:

- Software-Schutz
- Digitale Wasserzeichen
- Obfuskation von Konstanten wie Schlüsseln
- Obfuskation von Prädikaten
- Totale Obfuskation

3.2.1 Obfuskation von Algorithmen für Softwareschutz

Um Obfuskation in der realen Welt anzuwenden, um (kommerzielle) Software zu schützen, ist eine bestimmte Definition von Robustheit erforderlich.

Wenn jemand eine Software mit einer komplizierten Funktion geschrieben hat, die vermarktet werden soll, dann ist es wichtig, dass diese Funktion so obfusziert wird, dass keine Informationen über sie gewonnen werden können. Im Black-Box-Modell werden Programme so obfusziert, dass alle Eigenschaften eines Programms

versteckt werden. Ein Black-Box-Obfuskator ist jedoch hier nicht geeignet, da ein Käufer über das Wissen funktioneller Eigenschaften einer Software verfügen will.

Ein Angreifer kennt also hier stets die Funktion, die von einem Programm berechnet wird, soll aber nicht den Originalquellcode ableiten können. Die Idee ist, den Angreifer mit zwei Informationen auszustatten: Dem obfuszierten Programm $O(M)$ und einem (ineffizienten) Referenzprogramm $\widetilde{M}$, das die gleiche Funktion wie M berechnet [Kuz+07].

Definition 3.3. *Ein Obfuskator für TMs ist* Algorithmen-robust*, wenn folgendes gilt:*

Für jede OPPT A gibt es eine PPT S und eine vernachlässigbare Funktion α, so dass für alle Paare von TMs $(M, \widetilde{M})$ mit $M \equiv \widetilde{M}$ (Mist äquivalent zu $\widetilde{M}$)

$$\left|\Pr\left[A(O(M), \widetilde{M}) = 1\right] - \Pr\left[S^M(1^{|M|}, \widetilde{M}) = 1\right]\right| \leq \alpha(|M|).$$

gilt.

Kuzurin u. a. konstruierten einen Obfuskator für deterministische endliche Automaten (DEAs), die wir als nur nach rechts laufende TM auffassen können.

Satz 3.4. *Es gibt einen Algorithmen-robusten Obfuskator für deterministische endliche Automaten.*

Beweis. Wir können jeden effizienten DEA-Minimierungsalgorithmus O als Turingmaschinen-Obfuskator betrachten. Jeder DEA M kann zu einem eindeutigen DEA M_0 minimiert werden, so dass $M \equiv M_0$ ist [Mei13]. Also kann ein Simulator S, dem nur eine TM $\widetilde{M}$ gegeben wird, durch Anwendung eines effizienten Minimierungsalgorithmus für $\widetilde{M}$ leicht $O(\widetilde{M}) = O(M)$ berechnen und damit einen Angreifer $A(O(M), \widetilde{M})$ für das Paar $(M, \widetilde{M})$ simulieren.

□

Einfache Berechnungsmodelle wie DEAs sind nach Kuzurin u. a. [Kuz+07] zwar obfuszierbar, aber sie sind auch sehr schwach. Für die Klasse aller Turingmaschinen konnten Varnovsky u. a. [Var+09] zeigen, dass es keinen Algorithmen-robusten Obfuskator gibt:

Satz 3.5 (Varnovsky u. a. [Var+09])**.** *Es gibt keine Algorithmen-robusten Obfuskatoren für Turingmaschinen.*

3.2.2 Obfuskation von digitalen Wasserzeichen

Das *digitale Wasserzeichen* ist ein Analogon zu einer verborgenen Nachricht, eine technische Markierung von z. B. Ton-, Bild-, Videoträgern oder Programmcode. Wasserzeichen sind spezifische Daten, die durch den Eigentümer einer Nachricht in elektronischer Form hinzugefügt werden. Digitale Wasserzeichen in Dateien dienen für Dritte als Beweis dafür, dass diese geistiges Eigentum der Person sind, die dieses Wasserzeichen hinzugefügt hat, meist für kommerzielle Zwecke. Sie können individualisiert sein. So kann ein Verkäufer, der Kunden mit Dateien versorgt, zur Identifizierung des Kunden, an den verkauft wird, in jeder Kopie dieser Datei digitale Wasserzeichen platzieren. Sollten dann Raubkopien seiner Datei entdeckt werden, wird der Verkäufer in der Laage sein, den Kunden zu identifizieren. Mit digitalen Wasserzeichen können also Dateien authentifiziert und zurückverfolgt werden.

Wir betrachten hier digitale Software-Wasserzeichen als Obfuskation von Programmcode [Bar+01]. Zur Formalisierung des Konzepts der digitalen Wasserzeichen betrachten wir die Definition von Barak u. a. [Bar+01]. Ein Wasserzeichenschema besteht aus einem Markierungsalgorithmus, der ein digitales Wasserzeichen m in ein gegebenes Programm einflicht und einem Extraktionsalgorithmus, der dieses aus einem markierten Programm extrahiert. Als Programme werden hier Boole'sche Schaltkreise betrachtet. Ein Programm wird als *markiert* bezeichnet, wenn es mit einem digitalen Wasserzeichen versehen ist.

Definition 3.6 (Software-Wasserzeichen:)**.** *Ein Software-Wasserzeichen-Schema ist ein Paar von PTMs (Mark, Extract), die folgende Eigenschaften erfüllen:*

- *Funktionalität: Für alle Schaltkreise C, Schlüssel K, Wasserzeichen m, beschreibt die Zeichenkette $Mark_K(C, m)$ einen Schaltkreis, der die selbe Funktion wie C berechnet.*
- *Polynomielle Laufzeit: Es gibt ein Polynom p, sodass für jeden Schaltkreis C gilt: $|Mark_K(C, m)| \leq p(|C| + |m| + |K|)$.*
- *Extrahierbarkeit: Für alle Schaltkreise C, einen Schlüssel K und ein Wasserzeichen m gilt: $Extract_K(Mark_K(C, m)) = m$.*
- *Aussagekraft: Es gibt eine vernachlässigbare Funktion ν, so dass gilt: $\Pr_K[Extract_K(C) \neq \varepsilon] \leq \nu(|C|)$. Dabei ist ε eine leere Zeichenkette. Das heißt, zufällige Schaltkreise sollen mit hoher Wahrscheinlichkeit unmarkiert sein.*

- *Robustheit: Für jede PPT A gibt es eine PPT S und eine vernachlässigbare Funktion α, so dass für jeden Schaltkreis C und jedes Wasserzeichen m gilt:*

$$\begin{aligned}&\Big|\Pr\Big[A(Mark_K(C,m)) = C' : C \equiv C' \text{ und } Extract_K(C') \neq m\Big]\\ &-\Pr\Big[S^C(1^{|C|}) = C' : C \equiv C'\Big]\Big|\\ &\leq \nu(|C|).\end{aligned}$$

Dabei wird der Schlüssel K zufällig aus $\{0,1\}^{max(|C|,|m|)}$ gewählt.

Das Schema heißt *effizient*, wenn die Funktionen *Mark* und *Extract* in Polynomialzeit laufen. Für die meisten Schlüssel k gibt $Extract_K(C)$ eine leere Zeichenkette aus.

Wenn digitale Wasserzeichen verwendet werden, hat ein Angreifer Zugriff auf den markierten Schaltkreis. Wenn bereits Informationen über Ausgaben des Schaltkreises ausreichen, ein zum Ausgangsprogramm äquivalentes Programm zu erstellen, dann ergeben digitale Wasserzeichen als Softwareschutz keinen Sinn. Genau dies wird in der Formulierung der Robustheit berücksichtigt, in der die Wahrscheinlichkeit dafür, dass ein Angreifer das Wasserzeichen aus einem markierten Programm $Mark_K(C,m)$ entfernen kann, vergleichbar mit der Wahrscheinlichkeit ist, dass durch ein Orakelzugriff ein zum Ausgangsprogramm äquivalentes Programm erstellt werden kann.

Unter der Annahme, dass es Einwegfunktionen gibt, wurde gezeigt, dass es kein digitales Wasserzeichen-Schema im Sinne der Definition gibt.

Satz 3.7 (Barak u. a. [Bar+01])**.** *Wenn es Einwegfunktionen gibt, dann gibt es kein Wasserzeichen-Schema.*

Außerdem gilt für effiziente Wasserzeichen-Schemata:

Satz 3.8 (Barak u. a. [Bar+01])**.** *Es gibt keine effizienten Wasserzeichen-Schemata.*

3.2.3 Obfuskation von Konstanten

Obfuskation von Konstanten dient dem Schutz von Software. In vielen kryptographischen Anwendungen ist der Zweck von Obfuskation das Verschleiern einer zufällig ausgewählten Konstante c_0, die im Programm M enthalten ist, wie einem privaten Schlüssel. Bis auf die Konstante c_0 ist das Programm dem Angreifer bekannt.

Wir betrachten eine Menge von parametrisierten Turingmaschinen $P = \{M_c : c \in \{0,1\}^n, n \geq 1\}$, die sich voneinander nur in einer Konstante c unterscheiden.

Das Ziel der Obfuskation ist, dass kein Angreifer Informationen über die Konstante c aus M_c gewinnen kann [Kuz+07].

Definition 3.9 (Konstanten-Robustheit:)**.** *Ein Obfuskator O für eine parametrisierte Familie P von TMs ist* Konstanten-robust*, wenn folgendes erfüllt ist:*

Für jede PPT A gibt es eine OPPT S und eine vernachlässigbare Funktion ν, so dass für jedes zufällig gewählte Paar von Konstanten (c, c_0) gilt:

$$\left| \Pr\left[A(O(M_c), M_{c_0}) = 1\right] - \Pr\left[S^{M_c}(1^{|M_c|}, M_{c_0}) = 1\right] \right| \leq \nu(|M|).$$

Ob ein robuster Konstanten-Obfuskator konstruierbar ist, hängt von der Menge der parametrisierte TM's ab. Es ist offen, ob es für beliebige Familien von TM's Konstanten-robuste Obfuskatoren gibt [Kuz+07].

3.2.4 Obfuskation der Prädikate

Die einfachste Art von Obfuskation ist, dass nur eine Eigenschaft (Prädikat) eines Programms verschleiert wird. Diese Prädikate können zum Beispiel versteckte Fähigkeiten oder der Befall durch einen Computervirus sein [Kuz+07].

Obfuskation von Prädikaten ist dafür konzipiert, um eine bestimmte Eigenschaft eines Programms zu verstecken. Wir betrachten ein Prädikat π, das für eine Familie von TMs P definiert wird.

Definition 3.10 (π-Robustheit)**.** *Sei π ein Prädikat. Ein Obfuskator O ist π-robust für eine Familie von TMs Φ,falls es für jede PPT A eine OPPT S und eine vernachlässigbare Funktion α gibt, so dass für jede TM $M \in \Phi$ und deren Obfuskation $O(M)$ gilt:*

$$\left| \Pr\left[A(O(M)) = \pi(M)\right] - \Pr\left[S^M(1^{|M|}) = \pi(M)\right] \right| \leq \alpha(|M|).$$

Obfuskation einer Eigenschaft eines Programms ist einfacher als die Verschleierung des gesamten Programms. Es gibt für einige Klassen von Programmen π-robuste Obfuskatoren. Beispielsweise wurde in [Wee05] gezeigt, dass es für die Klasse von Programmen, die Konstanten und Punkt-Funktionen berechnen, robuste Obfuskatoren gibt.

3.2.5 Totale Obfuskation

Obfuskation im sogenannten *„Black-Box-Modell“* und im *„Grey-Box-Modell“* kann als Verschleierung aller Prädikate eines Programms angesehen werden. An der Verschleierung von Programmen im Black-Box-Modell arbeiteten unter anderen Barak u. a. [Bar+12] und an der im Grey-Box-Modell Kuzurin u. a. [Kuz+07]. Diese beiden Obfuskationsmodelle werden in den Kapiteln 5 und 6 ausführlich vorgestellt.

4 Obfuskation im Black-Box-Modell

Idealerweise ist ein obfusziertes Programm $O(P)$ eine virtuelle Black-Box, in dem Sinn, dass man alle Informationen, die man aus $O(P)$ über P gewinnen kann, bereits aus dem Eingabe-Ausgabe-Verhalten erfahren kann. Daher kommt der Name. Barak u. a. [Bar+01] formulierten 2001 eine eindeutige Definition von Obfuskation, das so genannte *virtuelle Black-Box-Modell*.

Programmobfuskation wird im virtuellen Black-Box Modell als robust definiert, wenn alle Informationen, die ein Angreifer aus der Obfuskation *O(P)* eines Programms P berechnen kann, nur das ist, was er mit einem Orakelzugriff berechnen kann. Jeder Angreifer kann das obfuszierte Programme starten, also für Eingaben Ausgaben erhalten. Das simuliert der Black-Box-Zugriff.

Der Hauptunterschied zwischen einem Schaltkreisobfuskator und einem Turingmaschinenobfuskator ist, dass eine Schaltung eine Funktion in einem endlichen Bereich (alle Eingänge einer bestimmten Länge) berechnet, während eine TM eine Funktion im unendlichen Bereich berechnet. Wäre die Größe der verschleierten Schaltkreise nicht beschränkt, so würde die Auflistung aller Werte der Schaltung eine gültige Obfuskation ergeben.

Definition 4.1 (Black-Box-Robustheit für Turingmaschinen)**.** *Ein Obfuskator O ist Black-Box-robust, wenn es für jede PPT A eine OPPT S und eine vernachlässigbare Funktion ν gibt, so dass für jede TM M gilt:*

$$\left| \Pr\left[A(O(M)) = 1\right] - \Pr\left[S^M(1^{|M|}) = 1\right] \right| \leq \nu(|M|).$$

Wir sagen, dass ein Turingmaschinen-Obfuskator effizient ist, wenn er in Polynomialzeit läuft. Die PPT A soll hier den Angreifer und S den Simulator darstellen. Dabei wird die erste Wahrscheinlichkeit für zufällige Größen berechnet, die von dem Obfuskator O und dem Angreifer A genutzt werden, und die zweite Wahrscheinlichkeit wird für zufällige Größen berechnet, die der Simulator S nutzt.

Jede beliebige Eigenschaft von obfuszierten Programme, die A berechnen kann, muss von S simuliert werden können.

Satz 4.2. *Wenn es einen Black-Box-robusten Turingmaschinenobfuskator gibt, dann gibt es auch einen Schaltkreisobfuskator [Bar+12].*

Es ist also schwerer Turingmaschinen zu obfuszieren als Schaltkreise.

Wir benötigen folgendes Lemma für den nächsten Satz. Es sagt aus, dass Obfuskatoren die Programme nicht wesentlich verkürzen können.

Lemma 4.3. *Sei O ein beliebiger Turingmaschinenobfuskator. Sei P eine beliebige Menge von 2^n Turingmaschinen, die paarweise verschiedene Funktionen berechnen. Dann gibt es mindestens $\frac{2^n}{2}$ Turingmaschinen $M \in P$, so dass $|y| \geq n-1$ für alle $y \in O(M)$.*

Beweis. Wir nehmen für einen Widerspruch an, für mehr als $\frac{2^n}{2}$ Maschinen M aus P wäre $|y| < n-1$ für irgendein $y \in O(M)$. Für jede mögliche Länge $m < n-1$ gibt es höchstens 2^m verschiedene Maschinen. Damit gibt es höchstens $\sum_{m=0}^{n-2} 2^m = 2^{n-1} - 1$ Maschinen mit Länge $< n-1$. Da M und $O(M)$ stets dieselbe Funktion berechnen, muss es aber mindestens 2^{n-1} verschiedene Maschinen der Länge $< n-1$ geben. Widerspruch.

□

Satz 4.4. *Es gibt keine robusten Turingmaschinenobfuskatoren im Black-Box-Modell.*

Beweis. Es wird gezeigt, dass es eine Menge von nichtobfuszierbaren Programmen gibt, das bedeutet, dass kein Simulator einen Angreifer simulieren kann. Angenommen, es gäbe einen robusten Obfuskator. Wir definieren Turingmaschinen $C_{\alpha,\beta}$, $D_{\alpha,\beta}$ und Z_n. Die TM Z_n gibt immer 0^n aus. $C_{\alpha,\beta}$, $D_{\alpha,\beta}$ arbeiten wie folgt:
Für alle Paare von Zeichenketten α, β gilt:

$$C_{\alpha,\beta}(x) = \begin{cases} \beta, & \text{wenn } x = \alpha \\ 0^n, & \text{sonst} \end{cases}$$

Bei jeder Eingabe x hält die TM $C_{\alpha,\beta}$ stets innerhalb von $2n$ Schritten. Die Maschine $D_{\alpha,\beta}$ arbeitet in Polynomialzeit. $D_{\alpha,\beta}$ erhält als Eingabe die Gödelisierung von irgendeiner TM.

$$D_{\alpha,\beta}(\langle M \rangle) = \begin{cases} 1, & M(\alpha) \text{ hält in } p(|M|+1) \text{ Schritten und } M(\alpha) = \beta \\ 0, & \text{sonst} \end{cases}$$

Dabei ist p ein Polynom, das im Folgenden erklärt wird.
Sei $M_0 \# M_1$ eine TM, welche die Paare $(\delta, x), \delta \in \{0,1\}$ als Eingaben erhält. Sie führt M_δ auf x aus. Wir definieren $F_{\alpha,\beta} = C_{\alpha,\beta} \# D_{\alpha,\beta}$ und $G_{\alpha,\beta} = Z_{|\alpha|} \# D_{\alpha,\beta}$.
Für beliebige Maschinen M, die mit Eingaben der Form (δ, x) arbeiten, definieren wir umgekehrt $M_b(x)$, als eine TM, die $M(b, x)$ ausführt. Wir definieren nun

eine $PPTA$, die als Eingaben Codierungen von solchen Maschinen N erhält, die Eingaben der Form (δ, x) akzeptieren. A simuliert $N_1(N_0)$ für $q(n)$ Schritte, wobei das Polynom q ebenfalls im weiteren Beweis definiert wird. Das Ziel hierbei ist, dass für Eingaben $O(F_{\alpha,\beta})$ die Funktion $D_{\alpha,\beta}(\langle C_{\alpha,\beta}\rangle)$ berechnet wird. Dafür müssen aber $p(n)$ und $q(n)$ groß genug sein, da $O(F_{\alpha,\beta})_0$ und $O(F_{\alpha,\beta})_1$ zwar dieselbe Funktion berechnet wie $C_{\alpha,\beta}$ und $D_{\alpha,\beta}$, aber eine höhere Laufzeit besitzen können. Es gilt:

$$
\begin{aligned}
& C_{\alpha,\beta}(x) \text{ ist eine Polynomialzeitmaschine} \\
\Rightarrow\ & \text{auf Eingaben } (0,x) \text{ hat } F_{\alpha,\beta} \text{ Polynomiallaufzeit } p_1(|x|) \\
\Rightarrow\ & \text{auf Eingaben } (0,x) \text{ hat } O(F_{\alpha,\beta}) \text{ Polynomiallaufzeit } p_2(|x|) \\
\Rightarrow\ & \text{auf allen Eingaben } x \text{ hat } N_0 = O(F_{\alpha,\beta})_0 \text{ Polynomiallaufzeit } p_3(|x|)
\end{aligned}
$$

Diese Funktion $p_3(n)$ verwenden wir für $p(n)$. Damit ist $D_{\alpha,\beta}$ eine Polynomialzeitmaschine. Sie arbeitet im Allgemeinen nur für solche Eingaben $\langle N_0\rangle$ korrekt, wenn $|N_0| + 1 \geq |\alpha|$ gilt, da nur dann $D_{\alpha,\beta}$ die Maschine N_0 genügend lange simuliert. Wir können annehmen, dass $|C_{\alpha,\beta}| \geq |\alpha|$, damit ist $|C_{\alpha,\beta}\#D_{\alpha,\beta}|$ auch $\geq \alpha$. Da für alle $\alpha, \beta \in \{0,1\}^n$ die Maschinen $C_{\alpha,\beta}$ paarweise verschiedene Funktionen berechnen, tun dies auch die 2^{2n} Maschinen $F_{\alpha,\beta}$. Nach Lemma 4.3 gilt aber, dass mindestens die Hälfte der Maschinen $O(F_{\alpha,\beta})$ Länge mindestens $2n - 1 \geq |\alpha| - 1$ hat und damit auch die Hälfte der N_0.
Für diese N_0 gilt, dass $D_{\alpha,\beta}(N_0) = 1 \Leftrightarrow N_0(\alpha) = \beta \Leftrightarrow C_{\alpha,\beta}(\alpha) = \beta$. Die erste Äquivalenz gilt, weil N_0 nach höchstens $p(|N_0| + 1)$ Schritten hält und $D_{\alpha,\beta}$ die Maschine N_0 auf α erfolgreich simulieren kann. Die zweite gilt, weil die Obfuskation die Funktionalität erhält.

Es gilt, dass N_1 und $D_{\alpha,\beta}$ dieselbe Funktion berechnen. Wir müssen also q so wählen, dass A die Maschine N_1 vollständig simulieren kann. Die Existenz dieses q folgt daher, dass $D_{\alpha,\beta}$ eine Polynomialzeitmaschine ist und die Obfuskation die Berechnung nur polynomiell verlangsamt. Insgesamt ist A also eine PPT, und es gilt:

$$
\begin{aligned}
& \Pr_{\alpha,\beta\leftarrow\{0,1\}^n}[\forall y \in O(F_{\alpha,\beta}) : |y| \geq |\alpha| - 1] \geq \frac{1}{2} \\
\Longrightarrow\quad & \Pr_{\alpha,\beta\leftarrow\{0,1\}^n}[A(O(F_{\alpha,\beta})) = 1] \geq \frac{1}{2}
\end{aligned}
$$

Eingabe von der Form $O(G_{\alpha,\beta}) = O(Z_{|\alpha|}\#D_{\alpha,\beta})$ werden immer abgelehnt, außer $\beta = 0^{|\alpha|}$. Es gilt also:

$$
\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}[A(O(G_{\alpha,\beta})) = 1] \leq 2^{-n}
$$

Nach Definition 4.1 gilt für alle DTMs M, dass

$$\begin{aligned} &\Pr[S^M(1^{|M|}) = 1] - \nu(|M|) \\ \leq &\Pr[A(O(M)) = 1] \\ \leq &\Pr[S^M(1^{|M|}) = 1] + \nu(|M|) \end{aligned}$$

und damit

$$\begin{aligned} &\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}[S^{F_{\alpha,\beta}}(1^{|F_{\alpha,\beta}|}) = 1] - \nu(|M|) \\ \leq &\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}[A(O(F_{\alpha,\beta})) = 1] \\ \leq &\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}[S^{F_{\alpha,\beta}}(1^{|F_{\alpha,\beta}|}) = 1] + \nu(|M|). \end{aligned}$$

Die Ungleichung gilt analog für $G_{\alpha,\beta}$ für eine vernachlässigbare Funktion ν.

Nun wird gezeigt, dass A nicht von S simuliert werden kann. Die Wahrscheinlichkeit dafür, dass die Maschine S Orakel der Form $F_{\alpha,\beta}$ wie von der Form $G_{\alpha,\beta}$ akzeptiert, ist etwa gleich groß.

$$\left|\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}\left[S^{F_{\alpha,\beta}}(1^n) = 1\right] - \Pr_{\alpha,\beta\leftarrow\{0,1\}^n}\left[S^{G_{\alpha,\beta}}(1^n) = 1\right]\right| \leq \nu(n) \tag{4.1}$$

Dies liegt daran, dass die Wahrscheinlichkeit für eine Orakelantwort von $C_{\alpha,\beta}$ oder $D_{\alpha,\beta}$, die nicht Null ist, vernachlässigbar ist. Somit ist auch der Anteil der Zufallsbänder von S, auf denen eine der polynomiell vielen Fragen eine solche Antwort erhält, vernachlässigbar.

$$\begin{aligned} \frac{1}{2} - \nu(|F_{\alpha,\beta}|) \leq &\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}[A(O(F_{\alpha,\beta})) = 1] - \nu(|F_{\alpha,\beta}|) \\ \leq &\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}[S^{F_{\alpha,\beta}}(1^{|F_{\alpha,\beta}|}) = 1] \end{aligned}$$

$$\begin{aligned} 2^{-n} + \nu(|G_{\alpha,\beta}|) \geq &\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}[A(O(G_{\alpha,\beta})) = 1] + \nu(|G_{\alpha,\beta}|) \\ \geq &\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}[S^{G_{\alpha,\beta}}(1^{|G_{\alpha,\beta}|}) = 1] \end{aligned}$$

Allerdings ist $(\frac{1}{2} - \nu(|F_{\alpha,\beta}|)) - (2^{-n} + \nu(|G_{\alpha,\beta}|))$ nicht vernachlässigbar, da die Größen von $F_{\alpha,\beta}$ und $G_{\alpha,\beta}$ polynomiell in $|\alpha|$ sind. Das ist eine Widerspruch zu der Gleichung (4.1).

□

Analog zu Obfuskatoren für Turingmaschinen definieren wir Obfuskatoren für Schaltkreise.

Definition 4.5 (Black-Box-Robustheit bei Schaltkreisen)**.** *Ein Obfuskator O ist robuster Obfuskator für Boole'sche Schaltkreise, wenn es für jede PPT A eine OPPT S und eine vernachlässigbare Funktion ν gibt, so dass für alle Boole'schen Schaltkreise C gilt:*

$$\left| \Pr\left[A(O(C)) = 1\right] - \Pr\left[S^C(1^{|C|}) = 1\right] \right| \leq \nu(|C|).$$

Analog zu dem Turingmaschinenobfuskator wurde mit Hilfe kryptographischer Elemente bewiesen, dass es für Bool'sche Schaltkreise keine effiziente robuste Obfuskation gibt [Bar+12]. Dafür werden Einwegfunktionen benötigt, also effizient berechenbare Funktionen, deren Umkehrung nicht effizient berechenbar ist.

Lemma 4.6. *Wenn es Einwegfunktionen gibt, dann gibt es keine Black-Box-robusten Schaltkreisobfuskatoren.*

Für effiziente Obfuskatoren wurde folgendes Lemma bewiesen:

Lemma 4.7. *Wenn es effiziente Black-Box-robuste Obfuskatoren gibt, dann gibt es Einwegfunktionen.*

Satz 4.8. *Es gibt keine effizienten Black-Box-robusten Schaltkreisobfuskatoren.*

Beweis. Angenommen, es gibt effiziente Black-Box-robuste Schaltkreisobfuskatoren. Dann muss es nach Lemma 4.7 Einwegfunktionen geben. Wenn es Einwegfunktionen gibt, dann gibt es nach Lemma 4.6 keine effizienten Black-Box-robusten Schaltkreisobfuskatoren. Widerspruch zur Annahme. □

Aus der Arbeit von Barak u. a. [Bar+12] geht insgesamt hervor, dass es keine einfache Lösung für eine robuste Obfuskation gibt. Aus diesem Grund haben die Forscher im Weiteren ihre Arbeit den Fragestellungen gewidmet:

- ob es andere Definitionen für Obfuskatoren gibt, deren Anforderungen an die Robustheit schwächer, als die im Black-Box Modell sind, und sich dadurch ein Weg zur robusten Obfuskation finden lässt.
- ob es Klassen von Programmen gibt, für die Obfuskatoren entworfen werden können, die den einen oder den anderen Anforderungen an die Robustheit entsprechen.

Alternative Definitionen der Robustheit im Black-Box-Modell

Die Anforderungen an die Robustheit in der Definition der Obfuskation im virtuellen Black-Box-Modell können abgeschwächt werden, indem die Anforderungen an den Angreifer und an den Simulator variiert werden oder die Klassen der zu obfuszierenden Programme eingeschränkt werden.

4.1 Ununterscheidbare Obfuskation nach Barak

Eine weitere Definition von Robustheit kommt von Barak u. a. [Bar+01]. Die Idee dabei ist folgende: Ein *Ununterscheidbarkeits-Obfuskator* ist intuitiv gesprochen ein Obfuskator, bei dem es unmöglich ist die obfuszierten Programme $O(P_1)$ und $O(P_2)$ von zwei verschiedene Programmen P_1 und P_2 mit gleicher Funktionalität von einander zu unterscheiden. Man kann nicht sagen, welche Verschleierung zu welchem der ursprünglichen Programme gehört, also ob $O(P_1)$ eine Obfuskation von P_1 oder von P_2 ist.

Definition 4.9 (Ununterscheidbarkeits-Robustheit)**.** *Ein Schaltkreisobfuskator ist ein Ununterscheidbarkeitsobfuskator, wenn es für jede PPT A eine vernachlässigbare Funktion ν gibt, so dass für jedes Schaltkreispaar C_1, C_2, die dieselbe Funktion berechnen und die gleiche Größe k haben, gilt:*

$$\left| \Pr\left[A(O(C_1))\right] - \Pr\left[A(O(C_2))\right] \right| \leq \nu(k).$$

Satz 4.10. *Es gibt Ununterscheidbarkeitsobfuskatoren.*

Beweis. Sei $O(C)$ der lexikographisch größte Schaltkreis der Größe $|C|$, der die gleiche Funktion wie C berechnet. □

Dieser Obfuskator ist jedoch ineffizient, denn: Erstens gibt es $2^{|C|}$ viele Schaltkreise C' der gleichen Größe. Außerdem muss man bis zu $2^{|C|}$ viele Eingaben auf beiden Schaltkreisen C und C' testen, um zu prüfen, ob sie die gleiche Funktion berechnen. Drittens ist er wegen der Berechnung des lexikographischen Codes langsam.

Diese Idee wurde von Garg u. a. [Gar+16] weiterentwickelt. Sie führte zu einem großen positiven Ergebnis, einer Konstruktion für einen Ununterscheidbarkeitsobfuskator für die Klasse aller Schaltkreise, der in Polynomialzeit läuft.

4.2 Das Black-Box-Modell mit schwach beschränktem Simulator

Wir betrachten folgende Variante der Definition der Robustheit eines Obfuskators, in der dem Simulator grenzenlose Rechenzeit erlaubt wird, während er nur polynomiell viele (in Abhängigkeit von der Größe des Programms) Orakelzugriffe durchführen darf [CRV10]. Dieser wird hier als *„Obfuskator mit schwach beschränktem Simulator“* bezeichnet. In derselben Arbeit [CRV10] wurde gezeigt, dass die Anforderungen an die Robustheit eines Obfuskators im virtuellen Black-Box-Modell mit schwach beschränktem Simulator lockerer sind, als die des „klassischen“ virtuellen Black-Box-Modells.

Canetti, Rothblum und Varia [CRV10] haben gezeigt, dass es eine Familie von Funktionen gibt, die im virtuellen Black-Box-Model mit schwach beschränktem Simulator obfuszierbar sind, aber nicht im „klassischen“ Black-Box-Modell. Der Beweis dafür basiert auf der gleichen Idee, wie der vom „klassischen“ Black-Box Modell. Sie zeigten auch, dass Familien von Funktionen existieren, für die es keine Obfuskation mit schwach beschränktem Obfuskator gibt.

4.3 Der differing-inputs-Obfuskator

Eine weitere Variante der Definition der Robustheit der Obfuskation wurde von Barak u. a. [Bar+12] eingeführt, der *differing-inputs-Obfuskator*.

Für je zwei Schaltungen C_0 und C_1 garantiert ein differing-inputs-Obfuskator, dass es keinen Angreifer geben kann, der eine Eingabe findet, bei dem sich C_0 und C_1 unterscheiden. Das bedeutet, dass $O(C_0)$ und $O(C_1)$ sich rechnerisch nicht unterscheiden.

Die Eigenschaften der Funktionalität und der Effizienz des differing-inputs-Obfuskators werden genau so definiert, wie die des Ununterscheidbarkeitsobfuskators. Aber die Anforderungen an die Robustheit sind wie folgt:

Definition 4.11 (differing-inputs-Robustheit)**.** *Ein Obfuskator O ist differing-inputs-robust, wenn es für jede PPT A eine PTM D und eine vernachlässigbare Funktion ν gibt, so dass für Schaltkreise C_1 und C_2 der Größe k folgendes gilt:*

Entweder ist

$$\epsilon =^{def} \left| \Pr\left[A(O(C_1)) = 1\right] - \Pr\left[A(O(C_2)) = 1\right] \right| \leq \nu(k),$$

oder $D(C_1', C_2')$ berechnet für alle $C_1 \equiv C_1', C_2 = C_2'$ eine Eingabe, für die sich C_1 und C_2 unterscheiden und läuft dabei in Zeit polynomiell in k und $\frac{1}{\epsilon - \nu(k)}$.

Diese Definition der Robustheit eines differing-inputs-Obfuskators ist stärker als die des Ununterscheidbarkeitsobfuskators, denn wenn C_1 und C_2 die gleiche Funktion berechnen, dann kann D niemals eine Eingabe finden, in der sie sich unterscheiden. In diesem Fall muss ϵ vernachlässigbar sein.

4.4 Obfuskator im Black-Box-Modell mit zusätzlicher Eingabe

Eine weitere Variante der Definition der Obfuskation, die der Definition des „klassischen“ Black-Box-Modells ähnelt, gibt es von Goldwasser und Kalai [GK05].

In der Praxis kann der Angreifer zusätzlich zu dem obfuszierten Programm $O(P)$ Informationen besitzen, die für die Deobfuskation von $O(P)$ relevant bzw. unterstützend sein könnten. Beispielsweise könnte er außer dem obfuszierten Programm $O(P)$ auch eine Demoversion, also eine verkleinerte Version $O(P)'$ verfügen. Dem entsprechend entstand die Definition für das „virtuelle Black-Box-Modell mit zusätzlicher Eingabe“ [GK05].

Definition 4.12 (Black-Box-Robustheit des Obfuskators mit zusätzlicher Eingabe)**.** *Ein Obfuskator O ist* robust mit zusätzlicher Eingabe *für eine Klasse von Schaltkreisen $C = \{C_n\}$, $n \in \mathbb{N}$, wenn gilt:*

Für jede PPT A (Angreifer) gibt es eine OPPT S (Simulator) und eine vernachlässigbare Funktion $\nu(\cdot)$, so dass für jedes Wort $z \in \{0,1\}^*$ und jedes $C \in \mathcal{C}$ gilt:

$$\left| \Pr\left[A(O(C), z) = 1\right] - \Pr\left[S^C(1^{|C|}, z) = 1\right] \right| \leq \nu(|C|).$$

Im Gegensatz zu Barak u. a. [Bar+12], zeigten Goldwasser und Kalai [GK05], dass es nicht nur eine Klasse von Schaltkreisen, sondern viele nichtobfuszierbare Schaltkreisfamilien gibt. Obfuskation mit zusätzlicher Eingabe ist stärker als die des klassische Black-Box-Modells von Barak.

Goldwasser und Kalai [GK05] fanden eine Familie Boole'scher Schaltkreise, für die es unmöglich ist, einen Obfuskator sowohl ohne als auch mit zusätzlicher Eingabe zu konstruieren.

Also zeigen die Resultate, dass die Lockerungen der Anforderungen an die Robustheit von Obfuskatoren im virtuellen Black-Box-Modell keinen wesentlichen Erfolg für die Obfuskation im Black-Box Modell bewirken. Denn entweder führen

diese auch dazu, dass die Obfuskatoren mit gelockerten Anforderungen an die Robustheit nicht existieren oder sie sind ineffizient.

Kuzurin u. a. [Kuz+07] forschten ebenfalls daran, ob es einen Obfuskator mit zusätzlicher Eingabe gibt, nur dass sie nicht irgendwelche Eingabe in Betracht gezogen haben, sondern den Berechnungsweg π des Programms $O(P)$ als zusätzliche Eingabe. Diese Art der Obfuskation wird *Grey-Box-Modell* genannt. Im nächsten Kapitel wird es ausführlich vorgestellt.

5 Obfuskation im Grey-Box-Modell

Die Anforderungen an die Robustheit einer Obfuskation können auch geschwächt werden, indem dem Simulator S außer der Information, die ihm im Black-Box-Modell zur Verfügung steht, eine zusätzliche Information gegeben wird, wie auch im Kapitel 4.4. Im Gegensatz zu diesem Ansatz erhält der Simulator hier aber eine bestimmte, von der obfuszierten Maschine abhängige, Eingabe. Eine der hier möglichen Varianten wurde 2007 von Kuzurin u. a. [Kuz+07] und 2009 von Varnovsky u. a. [Var+09] erforscht und wird von diesen als *Grey-Box-Modell* bezeichnet.

Im Black-Box-Modell zielt der Angreifer darauf ab, irgendeine Information des ursprünglichen nichtobfuszierten Programms zu erhalten. Das virtuelle Grey-Box-Modell unterscheidet sich vom virtuellen Black-Box-Modell darin, dass das Orakel O auf die Anfrage x nicht nur die Ausgabe des obfuszierten Programms M zurück gibt, sondern auch den dazugehörigen Berechnungspfad von $M(x)$. Der Berechnungspfad von $M(x)$ wird also nicht geheim gehalten. Der Simulator hat mehr Ressourcen als im Black-Box-Modell, somit können Funktionen, die nicht black-box-obfuszierbar sind, noch grey-box-obfuszierbar sein.

Die Zeichenkette $tr_M(x)$ ist als Verkettung aller aufeinanderfolgenden ausgeführten Einträge der Delta-Funktion (Anweisungen) definiert, die von M für x ausgeführt werden. $Tr(M)$ ist die Funktion, die x auf $tr_M(x)$ abbildet.

Definition 5.1 (Trace-Robustheit)**.** *Ein Obfuskator O ist* Trace-robust*, falls es für jede PPT A (Angreifer) eine OPPT S und eine vernachlässigbare Funktion ν gibt, so dass für alle Maschinen M gilt:*

$$\left| \Pr\left[A(O(M)) = 1\right] - \Pr\left[S^{Tr(M)}(1^{|M|}) = 1\right] \right| \leq \nu(|M|)$$

Es ist nicht bekannt, ob es Obfuskatoren gibt, die Trace-robust sind. Kuzurin u. a. [Kuz+07] haben deswegen die Maschinen aus dem Black-Box-Gegenbeispiel (siehe Satz 4.4 und [Bar+12]) zu sogenannten *reaktiven Programmen* erweitert, mit dem Ziel, die Zahl der Orakelfragen zu beschränken. Stattdessen werden hier gewöhnliche TMs als Orakel betrachtet. Um die Orakelfragen des Simulators zu beschränken, werden diese jedoch parallel (*nicht-adaptiv*) gestellt.

Definition 5.2 (Orakel-Turingmaschine der Truth-Table-Art)**.** *Eine tt-Orakel-PPT wird definiert wie eine Orakel-PPT, außer, dass alle Orakelfragen $a_1, \ldots, a_n$ parallel in Form eines Wortes $a_1\# \ldots \#a_n$ gestellt werden und nur eine solche Frage gestellt werden darf. Die Antwort ist dann ein Wort $b_1\# \ldots \#b_n$, wobei b_i die Antwort auf die Frage a_i ist.*

Definition 5.3 (Schwache Trace-Robustheit)**.** *Ein Obfuskator O ist* Schwach Trace-robust*, falls es für jede PPT A (Angreifer) eine tt-Orakel-PPT S gibt, so dass für alle Maschinen M gilt:*

$$\left| \Pr\left[A(O(M)) = 1\right] - \Pr\left[S^{Tr(M)}(1^{|M|}) = 1\right] \right| \leq \nu(|M|)$$

Auch hier bedeutet diese Ungleichung, dass jede Eigenschaft, die ein beliebiger Angreifer A über ein M ausrechnen könnte, auch der Simulator S ausrechnen kann. Der Betrag der Differenz dieser Wahrscheinlichkeiten wächst insgesamt langsamer, als $\nu(|M|)$.

Satz 5.4. *Wenn es Einwegfunktionen gibt, dann gibt es keine schwach Trace-Robusten Obfuskatoren.*

Der Berechnungspfad von $O(M)$ darf keine zusätzliche nützliche Informationen zur Verfügung stellen. Ein Simulator, der Zugriff zu den Berechnungspfaden der Ausführung von M hat, bekommt zusätzlich zu dem Tupel $(Eingabe, Ausgabe)$ auch den Berechnungspfad. Um gegen einen solchen mächtigen Simulator zu gewinnen, werden Einwegfunktionen eingesetzt.

Es reicht aus, die Konstruktion des Beweises für den Black-Box-Obfuskator so zu modifizieren.

Und zwar, anstatt die Eingabe x der Maschine $C_{\alpha,\beta}$ mit der festen Zeichenkette α zu vergleichen (siehe Satz 4.4), wird zuerst geprüft, ob $f(x) = f(\alpha)$, wobei f eine Einwegfunktion ist. Nur wenn diese Gleichung erfüllt ist, wird geprüft, ob $x = \alpha$ ist. Die Idee dahinter ist, dass ein Simulator mit einer Frage x, für die $f(x) = f(\alpha)$ ist, gleichzeitig ein Urbild von $f(\alpha)$ liefert.

Beweis. Wir definieren eine nichtobfuszierbare Familie von TMs, das heißt, dass auch hier wie im Black-Box-Modell gilt, dass kein Simulator einen Angreifer simulieren kann. Das Gegenbeispiel aus dem Black-Box-Modell (Satz 4.4) besteht aus zwei Familien von Turingmaschinen: $C_{\alpha,\beta}(x)$ und $D_{\alpha,\beta}(C)$, die auch hier genutzt werden. Für alle Paare von Zeichenketten $\alpha, \beta \in \{0,1\}^n$ gilt:

$$C_{\alpha,\beta}(x) = \begin{cases} \beta, & \text{wenn } x = \alpha \\ 0, & \text{sonst} \end{cases}$$

$$D_{\alpha,\beta}(C) = \begin{cases} 1, & \text{wenn } C(\alpha) = \beta \\ 0, & \text{sonst} \end{cases}$$

Sei $f : \{0,1\}^* \to \{0,1\}^*$ eine Einwegfunktion, die nach Voraussetzung des Satzes existiert.

Nun wird eine TM $C^f_{\alpha,\beta}$ wie folgt definiert:
Für die Eingabe x berechnet $C^f_{\alpha,\beta}$ die Funktion $f(x)$ und vergleicht das Ergebnis mit $f(\alpha)$. Wenn $f(x) \neq f(\alpha)$, dann hält $C^f_{\alpha,\beta}$ mit der Ausgabe 0^n. Sollte der Fall eintreten, dass $f(x) = f(\alpha)$, dann prüft $C^f_{\alpha,\beta}$, ob $x = \alpha$ ist. Wenn diese Gleichheit erfüllt ist, wird β ausgegeben, ansonsten 0^n. Die Maschine $Z^f_{\alpha,\beta}$ ist wie $C^f_{\alpha,\beta}$ definiert, außer, dass sie bei Eingabe x auch dann 0^n ausgibt, wenn $f(x) = f(\alpha)$ ist. Die Folge dabei ist, dass die Traces der beiden Maschinen gleich sind, außer x ist ein Urbild von $f(\alpha)$.

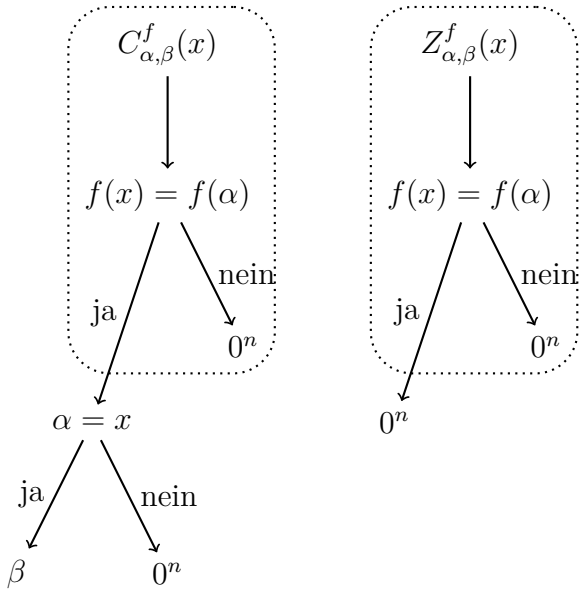

Abbildung 5.1: Berechnungsbäume von $C^f_{\alpha,\beta}$ und $Z^f_{\alpha,\beta}$

Das Paar $C^f_{\alpha,\beta}$ und $Z^f_{\alpha,\beta}$ ersetzen $C_{\alpha,\beta}$ und Z_n aus dem Beweis von Satz 4.4, in dem ein nicht simulierbarer Angreifer A konstruiert wurde. Diesen verwenden wir hier ebenfalls.

Der Beweis muss allerdings im folgenden Punkt modifiziert werden.

Wir definieren die TMs $F^f_{\alpha,\beta} := C^f_{\alpha,\beta}\#D_{\alpha,\beta}$ und $G^f_{\alpha,\beta} := Z^f_{\alpha,\beta}\#D_{\alpha,\beta}$. Es muss gezeigt werden, dass kein Simulator S die beiden Orakel $Tr(F^f_{\alpha,\beta})$ und $Tr(G^f_{\alpha,\beta})$ voneinander unterscheiden kann, als Gleichung ausgedrückt gibt es für alle tt-Orakel-PPTs S eine vernachlässigbare Funktion ν, so dass für alle n gilt:

$$\left| \Pr_{\alpha,\beta\leftarrow\{0,1\}^n}\left[S^{Tr(F^f_{\alpha,\beta})}(1^n)=1\right] - \Pr_{\alpha,\beta\leftarrow\{0,1\}^n}\left[S^{Tr(G^f_{\alpha,\beta})}(1^n)=1\right]\right| \leq \nu(n),$$

Dafür reicht es zu zeigen, dass folgende Ungleichung gilt:

$$\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}\left[S^{Tr(F^f_{\alpha,\beta})}(1^n) \neq S^{Tr(G^f_{\alpha,\beta})}(1^n)\right] \leq \nu(n),$$

Das heißt, dass die Wahrscheinlichkeit dafür, dass ein Simulator bei den Orakeln verschiedene Ausgaben hat, vernachlässigbar ist.

Dafür muss die Antwort auf die einzig gestellte Orakelanfrage $(\delta_1, x_1), \ldots, (\delta_s, x_s)$ verschieden ausfallen, weil im tt-Orakel-Modell die Orakelanfragen (δ_i, x_i) parallel gestellt werden. Die Traces vom zweiten Orakel $D_{\alpha,\beta}$ sind gleich, da $F^f_{\alpha,\beta}$ und $G^f_{\alpha,\beta}$ beide $D_{\alpha,\beta}$ als zweite Komponente beinhalten. Daher muss das erste Orakel verschiedene Traces von $C^f_{\alpha,\beta}$ bzw. $Z^f_{\alpha,\beta}$ als Antworten ausgeben. Dafür muss jedoch eines der x_i ein Urbild von $f(\alpha)$ sein. Wir zeigen nun, dass die Wahrscheinlichkeit dafür vernachlässigbar ist.

Angenommen, sie wäre nicht vernachlässigbar. Dann gibt es eine PPT T, die f invertiert. Dies würde dem widersprechen, dass f eine Einwegfunktion ist.

Wir konstruieren nun T. T erhält als Eingabe ein Wort y und soll ein z ausgeben, so dass gilt: $f(z) = y$. T arbeitet wie folgt. T simuliert $S(1^{|y|})$ bis S Orakelfragen $(\delta_1, x_1), \ldots, (\delta_s, x_s)$ stellt. Dann vergleicht T, ob $|x_i| = |y|$ und $y = f(x_i)$, was bedeuten würde, dass x_i ein Urbild von y ist. In diesem Fall hat T die Ausgabe x_i, sonst ist die Ausgabe beliebig. Deshalb gilt:

$$\begin{aligned} &\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}\left[S(1^n) \text{ stellt Frage } x \in f^{-1}(\alpha), |x| = n\right] \\ \leq &\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}\left[T(f(\alpha)) \in f^{-1}(\alpha)\right] \end{aligned}$$

Da die untere Wahrscheinlichkeit vernachlässigbar ist, ist dies auch die obere. Die Folge ist, dass auch die Wahrscheinlichkeit dafür, dass der Simulator die Traces von $F^f_{\alpha,\beta}$ und $G^f_{\alpha,\beta}$ unterscheiden kann, ebenfalls vernachlässigbar ist. Insgesamt folgt

$$
\begin{aligned}
&\left| \Pr_{\alpha,\beta\leftarrow\{0,1\}^n}\left[S^{Tr(F^f_{\alpha,\beta})}(1^n)=1\right] - \Pr_{\alpha,\beta\leftarrow\{0,1\}^n}\left[S^{Tr(G^f_{\alpha,\beta})}(1^n)=1\right]\right| \\
\leq &\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}\left[S^{Tr(F^f_{\alpha,\beta})}(1^nS)\neq S^{Tr(G^f_{\alpha,\beta})}(1^n)\right] \\
\leq &\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}\left[S(1^n) \text{ stellt Frage } x\in f^{-1}(\alpha), |x|=n\right] \\
\leq &\Pr_{\alpha,\beta\leftarrow\{0,1\}^n}\left[T(f(\alpha))\in f^{-1}(\alpha)\right] \leq \nu(n)
\end{aligned}
$$

für eine vernachlässigbare Funktion ν. Damit ist der Satz bewiesen. □

Daraus folgt, dass es, unter der Annahme der Existenz von Einwegfunktionen, keine schwach trace-robusten TM-Obfuskatoren gibt.

6 Zusammenfassung und Ausblick

Zusammenfassung

Die Aufgabe der Obfuskation ist Programme so zu transformieren, dass die funktionellen Eigenschaften des Programms beibehalten werden, aber eine Extraktion wichtiger Informationen des Ausgangsporogramms dabei unmöglich oder sehr langwierig sein muss. In dieser Arbeit wurden bestehende Obfuskationsmodelle vorgestellt. Insbesondere wurden das Black-Box-Modell und dessen Varianten so wie das Grey-Box-Modell ausführlich behandelt. Die Hauptbetonung der Merkmale der Obfuskation liegt auf der Robustheit der Obfuskatoren. Ein Obfuskator ist robust, wenn ein Angreifer eines obfuszierten Programms keine Informationen über das Ausgangsprogramm gewinnen kann. Diese Robustheit wird je nach Modell unterschiedlich definiert. Die Frage nach der Existenz von Obfuskatoren, welche den verschiedenen formale Anforderungen an die Robustheit entsprechen, ist das Hauptthema dieser Arbeit.

Die wichtigsten Forschungsergebnisse nach robusten Obfuskatoren sind jedoch negativ. Barak u. a. [Bar+12] zeigten mit Hilfe kryptographischer Mittel, dass es keine robusten und effizienten Obfuskatoren für alle Klassen von Turingmaschinen oder Schaltkreisen im Black-Box-Modell gibt.

Einige Forscher versuchten deshalb, die Definition der Robustheit eines Obfuskators abzuschwächen. So entstanden Varianten im Black-Box-Modell, wie z. B. der Obfuskator mit zusätzlicher Eingabe. Jedoch wurde bewiesen, dass es auch für all diese Varianten entweder keine robusten oder effizienten Obfuskation gibt.

Kuzurin u. a. [Kuz+07] haben das Modell mit zusätzlicher Eingabe spezialisiert, so dass die zusätzliche Eingabe eine bestimmte ist, nämlich der Berechnungspfad oder der Trace von dem obfuszierten Programm. So entstand das Grey-Box-Modell. Auch sie zeigten mittels kryptographischer Werkzeuge, dass es auch im Grey-Box-Modell ein Programmfamilie, die nicht im Grey-Box-Modell obfuszierbar ist, sofern die Orakelfragen des Simulators beschränkt werden.

Positive Resultate existieren hauptsächlich für kleinere Klassen von Programmen wie DEAs (Abschnitt 3.2.1) oder für Schaltkreise [Gar+16].

Ausblick

Eine weitere Herangehensweise an die Obfuskation ist die Einschränkung der obfuszierten Programme. Anstatt alle Turingmaschinen und Schaltkreise zu obfuszieren, kann man versuchen einen Obfuskator für beschränkte Maschinen- oder Schaltkreisklassen zu entwerfen.
Allerdings gibt es nicht nur beliebige unobfuszierbare Familien von Schaltkreisen. Naor und Reingold [NR04] zeigten, dass sogar eine nichtobfuszierbare Familie von Schaltkreisen existiert, die in TC^0 berechenbar ist.

Ein weiterer möglicher Ansatz ist, eine Klasse nicht hinsichtlich ihrer Berechnungskomplexität, sondern hinsichtlich ihrer Semantik einzuschränken, so dass etwa nur Programme enthalten sind, die einen bestimmten Zweck haben.

Ein Beispiel ist eine Klasse von Programmen, die asymmetrische Verschlüsselungsalgorithmen implementieren. Barak u. a. [Bar+01] konnten allerdings zeigen, dass solche Programme nur dann existieren, wenn auch unobfuszierbare Varianten existieren.

Es gibt noch viele weitere engeführte Obfuskationsbegriffe. Goldwasser und Rothblum untersuchten im Jahr 2007 [GR07] die so genannte *bestmögliche Obfuskation*. Ein Obfuskator gilt als *„bestmöglicher“*, wenn man aus einem obfuszierten Programm nicht mehr Informationen gewinnen kann, als aus jedem anderen Programm mit der gleichen Funktionalität und gleicher Größe. Es wurden bestmögliche Obfuskatoren für Klassen von Programmn konstruiert. Dieser Ansatz umgeht das Unmöglichkeitsresultat von Barak, indem mehr Informationen preisgegeben werden dürfen, als in der Black-Box-Robustheit.

Garg u. a. [Gar+16] haben im Bereich der Obfuskation einen Durchbruch erzielt. Sie griffen die Idee der *Indistinguishability Obfuscation (iO)* von Barak u. a. [Bar+01] auf (siehe Abschnitt 4.1). Auf dieser Grundlage ist ein Unterscheidbarkeits-Obfuskator für alle Klassen von Boole'schen Schaltkreisen konstruiert worden.

Neuere Arbeiten untersuchen Obfuskatoren unter Benutzung von kryptographischen Werkzeugen, wie Hashfunktionen, Zufallsorakel und so genannte *Universal Computational Extractors (UCE)* [BFM14].

Literatur

[Ang92] Dana Angluin. „Computational Learning Theory: Survey and Selected Bibliography“. In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*. 1992, S. 351–369. DOI: 10.1145/129712.129746. URL: http://doi.acm.org/10.1145/129712.129746.

[Bar+01] Boaz Barak u. a. „On the (Im)possibility of Obfuscating Programs“. In: *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*. 2001, S. 1–18. DOI: 10.1007/3-540-44647-8_1. URL: http://dx.doi.org/10.1007/3-540-44647-8_1.

[Bar+12] Boaz Barak u. a. „On the (im)possibility of obfuscating programs“. In: *J. ACM* 59.2 (2012), S. 6. DOI: 10.1145/2160158.2160159. URL: http://doi.acm.org/10.1145/2160158.2160159.

[BFM14] Christina Brzuska, Pooya Farshim und Arno Mittelbach. „Indistinguishability Obfuscation and UCEs: The Case of Computationally Unpredictable Sources“. In: *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*. 2014, S. 188–205. DOI: 10.1007/978-3-662-44371-2_11. URL: http://dx.doi.org/10.1007/978-3-662-44371-2_11.

[CGH04] Ran Canetti, Oded Goldreich und Shai Halevi. „The random oracle methodology, revisited“. In: *J. ACM* 51.4 (2004), S. 557–594. DOI: 10.1145/1008731.1008734. URL: http://doi.acm.org/10.1145/1008731.1008734.

[CRV10] Ran Canetti, Guy N. Rothblum und Mayank Varia. „Obfuscation of Hyperplane Membership“. In: *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*. 2010, S. 72–89. DOI: 10.1007/978-3-642-11799-2_5. URL: http://dx.doi.org/10.1007/978-3-642-11799-2_5.

[CTL97] Christian Collberg, Clark Thomborson und Douglas Low. *A taxonomy of obfuscating transformations*. Techn. Ber. Department of Computer Science, The University of Auckland, New Zealand, 1997.

[Gar+16] Sanjam Garg u. a. „Candidate Indistinguishability Obfuscation and Functional Encryption for All Circuits“. In: *SIAM J. Comput.* 45.3 (2016), S. 882–929. DOI: 10.1137/14095772X. URL: http://dx.doi.org/10.1137/14095772X.

[GK05] Shafi Goldwasser und Yael Tauman Kalai. „On the Impossibility of Obfuscation with Auxiliary Input“. In: *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*. 2005, S. 553–562. DOI: 10.1109/SFCS.2005.60. URL: http://dx.doi.org/10.1109/SFCS.2005.60.

[GR07] Shafi Goldwasser und Guy N. Rothblum. „On Best-Possible Obfuscation“. In: *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*. 2007, S. 194–213. DOI: 10.1007/978-3-540-70936-7_11. URL: http://dx.doi.org/10.1007/978-3-540-70936-7_11.

[Kuz+07] Nikolay Kuzurin u. a. „On the concept of software obfuscation in computer security“. In: *International Conference on Information Security*. Springer. 2007, S. 281–298.

[Mei13] Arne Meier. *Formale Sprachen*. 2013.

[Nic07] Arfst Nickelsen. *Komplexitätstheorie*. 2007.

[NR04] Moni Naor und Omer Reingold. „Number-theoretic constructions of efficient pseudo-random functions“. In: *Journal of the ACM (JACM)* 51.2 (2004), S. 231–262.

[Val84] Leslie G. Valiant. „A Theory of the Learnable“. In: *Commun. ACM* 27.11 (1984), S. 1134–1142. DOI: 10.1145/1968.1972. URL: http://doi.acm.org/10.1145/1968.1972.

[Var+09] NP Varnovsky u. a. „On the secure obfuscation of computer programs“. In: *Scientific statements Belgorod State University. Series: Economy. Computer science* 12.15 (70) (2009).

[Vol13] Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer Science & Business Media, 2013.

[Wee05] Hoeteck Wee. „On obfuscating point functions“. In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*. 2005, S. 523–532. DOI: 10.1145/1060590.1060669. URL: http://doi.acm.org/10.1145/1060590.1060669.

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 01.12.2016

Maria Buling