Rechnen mit verschlüsselten Programmen und Daten

Der Fakultät für Elektrotechnik und Informatik der
Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades

Doktor der Naturwissenschaften
Dr. rer. nat.

genehmigte Dissertation

von
M. Sc. Michael Brenner
geboren am 7. August 1974 in Neustadt a. Rbge.

2012

Referent: Prof. Dr. Matthew Smith
Koreferent: Prof. Dr. Jörg Hähner

Tag der Promotion: 7. Dezember 2012

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit in der Einrichtung Leibniz Universität IT Services (ehemals Regionales Rechenzentrum für Niedersachsen) und am Forschungszentrum L3S in Hannover.

Ich danke besonders Herrn Prof. Dr. Matthew Smith für die Unterstützung bei der Erstellung dieser Arbeit und das entgegengebrachte Vertrauen. Ich danke weiterhin Herrn Prof. Dr. Jörg Hähner für die Übernahme des Koreferats und Frau Prof. Dr. Gabriele von Voigt für die Unterstützung und eine stets offene Tür.

Außerdem bedanke ich mich bei meinen Kollegen in der Distributed Computing & Security Group für eine fruchtbare und vertrauensvolle Arbeitsatmosphäre, insbesondere bei Henning Perl für die wertvolle Tätigkeit während seiner Master-Arbeit und die unzähligen, konstruktiven Diskussionen.

Bedanken möchte ich mich bei meiner Familie für den steten Zuspruch und bei meiner Frau Nadine für die aufgebrachte Geduld.

Neustadt a. Rbge. im Dezember 2012

Zusammenfassung

Verteilte IT-Infrastrukturen bestimmen längst den Lebensalltag der meisten Menschen. Während Unternehmen das mittlerweile etablierte Ressourcen-Paradigma des Cloud-Computing anwenden, um auf entfernten Miet-Ressourcen Rechenleistung auszulagern und somit die Kostenstruktur der eigenen IT-Infrastruktur dynamisch und bedarfsgerecht zu gestalten, kommen deren Kunden mit den gemieteten Ressourcen mittelbar dadurch in Kontakt, dass die zu verarbeitenden Daten auf die entfernten Infrastrukturen übertragen werden. Aber auch Privatpersonen können Cloud-Ressourcen direkt verwenden, indem sie die in letzter Zeit vermehrt angebotenen Speicherdienste für Endanwender buchen und ihre Daten, Digitalfotos und sonstige Informationen mit dem Vorteil der ortsungebundenen Abrufbarkeit dort sichern oder virtuelle Maschinen zur universellen Verwendung anmieten. In jedem Falle müssen möglicherweise vertrauliche Daten in eine potenziell ungeschützte Umgebung transferiert werden, wobei der Anwender die Kontrolle darüber verliert, mit welchen Mechanismen und mit welchem Sicherheitsniveau die Informationen verarbeitet werden. Mit den gegenwärtig verfügbaren Sicherheitstechnologien kann lediglich der Transport der Daten geschützt werden, die Verarbeitung selbst findet dagegen stets unverschlüsselt statt.

Ausgangspunkt für diese Arbeit ist die Herausforderung, dass auf entfernten Ressourcen verarbeitete Daten und Programme auch während des eigentlichen Verarbeitungsvorgangs verschlüsselt und damit geschützt bleiben können. Dies erlaubt es, die Verarbeitung auch vertraulicher Informationen unabhängig von der Vertrauenswürdigkeit und dem Sicherheitsniveau des Delegenten durchzuführen. Diese Dissertation beschreibt unterschiedliche Lösungen für die Problematik des vertraulichen Rechnens auf fremden und potenziell risikobehafteten Ressourcen und schlägt neue Konstruktionen zum Schutz der Vertraulichkeit

von Programmen und Daten während der Verarbeitung vor.

Die technische Grundlage für die in dieser Arbeit vorgestellten Methoden bildet die homomorphe Kryptografie. Diese gestattet es prinzipiell, elementare binärarithmetische Operationen auf verschlüsselten Bits auszuführen. Auf Basis dieser Eigenschaft werden in dieser Arbeit zunächst einige elementare digitale Schaltungselemente konstruiert, die bereits zur Lösung einzelner Problemstellungen verwendet werden können. Im weiteren Verlauf dienen diese Elemente zur Bildung einer vollständigen Abstraktion der unterliegenden Kryptografie in Gestalt eines verschlüsselten Computersystems, bestehend aus verschlüsseltem Prozessor und verschlüsseltem Arbeitsspeicher.

Zur Verbesserung der Leistungsfähigkeit homomorph verschlüsselter Algorithmen werden in dieser Arbeit hybride Systeme eingeführt, bei denen nur ein geringer Teil tatsächlich verschlüsselt abläuft. Die Herausforderung bei dieser Vorgehensweise ist die Formulierung entsprechender Protokolle, welche die verschlüsselten und unverschlüsselten Komponenten so zusammenführen, dass die Vertraulichkeit der Verarbeitung dennoch gewährleistet bleibt.

Die vorgestellten Konzepte werden mit prototypischen Implementierungen überprüft und in ihrer Leistungsfähigkeit untersucht. Die Basis hierzu bildet die erste frei verfügbare Implementierung des homomorphen Smart-Vercauteren Kryptosystems, welche im Rahmen dieser Arbeit entstand. Aufbauend darauf werden die verschlüsselten Prozessor- und Speicherkomponenten auf unterschiedlichen Plattformen realisiert, um die Übertragbarkeit und universelle Einsetzbarkeit nachzuweisen. Die weiteren Konstruktionen und Hybrid-Systeme in dieser Arbeit werden mit unterschiedlichen homomorphen Kryptosystemen evaluiert. Die so validierten Mechanismen können die Sicherheit des Rechnens in verteilten Systemen wie dem Cloud-Computing signifikant verbessern und so zur Erschließung sicherheitsrelevanter Anwendungsgebiete für das verteile Rechnen in modernen Ressourcen-Paradigmen beitragen.

Schlagwörter: Computersicherheit, Homomorphe Kryptografie, Verschlüsseltes Rechnen

Abstract

Networked information systems permeate modern life. Companies apply the widely accepted cloud computing paradigm to delegate computation to hired remote resources for the purpose of cost structure controlling and to dynamically shape the infrastructure size according to actual requirements. The customers of these companies are affected by this strategy, because their data is uploaded to the remote resources in order to be processed. But also individuals are directly confronted with cloud computing when accepting the various offers to save data, pictures and other information in one of the cloud storage spaces, a service that recently has become increasingly popular. The advantage for the user is high availability of the data and accessibility from any location. In either case, customer or individual, private information has to be transferred into potentially insecure environments, where the user can no longer control under what mechanisms and security levels the information is being processed. Using the currently available encryption technologies, only transport and storage of data can be secured. However, data processing is usually still unencrypted.

The starting point for this thesis is the challenge to safely process data on remote resources by protecting programs and data even during processing. This allows to outsource the computation of confidential information independently from the trustworthiness or the security level of the remote delegate. This thesis describes different solutions for the problem of confidential computation in potentially insecure remote environments and proposes novel constructions to protect the confidentiality of programs and data during the processing step.

The technological basis for the methods presented in this thesis is homomorphic cryptography which allows to compute elementary binary operations on encrypted bits. Taking advantage of this property, this thesis first describes

a number of basic digital circuits that can already be used to solve different fundamental problems of limited complexity. These elements are then applied to construct an encrypted computer system with encrypted memory and encrypted processor circuits. The result is an abstraction of the underlying homomorphic cryptography.

To improve the performance of homomorphically encrypted algorithms, this thesis introduces the concept of hybrid systems that are only partly encrypted and most elements process information in plain or non-homomorphically encrypted. The challenge of this approach is to formulate protocols that combine encrypted and unencrypted components in a way that still guarantees the confidentiality of the processed information.

The presented concepts are evaluated with prototypic implementations that provide performance figures. The foundation is the first open-source implementation of the fully homomorphic Smart-Vercauteren cryptosystem which has been realized during the course of this thesis. On top of this, the encrypted processor- and memory circuits are implemented for different platforms to prove portability and to sketch a universal deployment model. The further constructions and hybrid system examples are evaluated with different homomorphic cryptosystems and show, how limited homomorphic schemes can be practically applied. The validation of the mechanisms significantly improves the security of computation in distributed environments like the cloud. This enables access to new security-relevant use cases for distributed computing in modern resource paradigms.

Keywords: computer security, homomorphic cryptography, encrypted computing

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

AES ..... Advanced Encryption Standard
ALU ..... Arithmetic-Logical Unit
BIOS ..... Basic Input/Output System
BMWi ..... Bundesministerium für Wirtschaft und Technologie
CCA ..... Chosen Ciphertext Attack
CLA ..... Carry-Look-Ahead-Adder
CPA ..... Chosen Plaintext Attack
CSS ..... Content Scrambling System
CU ..... Control Unit
CVP ..... Closest Vector Problem
DES ..... Data Encryption Standard
DH ..... Diffie-Hellman
DNA ..... Deoxyribonucleic Acid
DNF ..... Disjunktive Normalform
DPA ..... Differential Power Analysis
DRM ..... Digital Rights Management
EMSCB ..... European Multilaterally Secure Computing Base
FE ..... Funktionseinheit
FHE ..... Fully Homomorphic Encryption
FPGA ..... Field Programmable Gate Array
HNF ..... Hermite Normalform
IND ..... Indistinguishable
IPSec ..... Internet Protocol Security
ISA ..... Instruction Set Architecture
ISS ..... Instruction Set Simulator
JAR ..... Java Archive

JDK ..... Java Development Kit
JTAG ..... Joint Test Action Group
LPC ..... Low Pin Count
LSB ..... Least Significant Bit
LSF ..... Load Sharing Facility
MSB ..... Most Significant Bit
MU ..... Memory Unit
NIST ..... National Institute of Standards and Technology
ORAM ..... Oblivious Random Access Machine
PBS ..... Portable Batch System
PDA ..... Personal Digital Assistant
PKI ..... Public Key Infrastructure
PRAM ..... Parallel Random Access Machine
RCA ..... Ripple-Carry-Adder
ROM ..... Read Only Memory
RSA ..... Rivest-Shamir-Adleman
SFE ..... Secure Function Evaluation
SHA ..... Secure Hash Algorithm
SHE ..... Somewhat Homomorphic Encryption
SPA ..... Simple Power Analysis
SSL ..... Secure Sockets Layer
TCG ..... Trusted Computing Group
TCPA ..... Trusted Computing Platform Alliance
TPC ..... Trusted Platform Computing
TPM ..... Trusted Platform Module
VHDL ..... Very High Speed Integrated Circuit Hardware Description Language

Kapitel 1

Einleitung

Moderne IT-Infrastrukturen bestimmen längst unseren Lebensalltag. Sie kontrollieren sprichwörtlich alles von der Kommunikation bis hin zu Krankenhaussystemen, vom elektronischen Personalausweis, der Kreditkarte bis hin zu internationalen Finanzströmen. Ihre Omnipräsenz macht diese Systeme zunehmend zum Ziel gut geplanter und durchgeführter Angriffe, wie die Vorfälle um *Stuxnet* oder zuletzt *Flame* beweisen. Diese Angriffe betreffen längst nicht mehr nur Einzelpersonen, sondern bedrohen durch ihr Ausmaß Wirtschaftsunternehmen, geografische Regionen und sogar die nationale Sicherheit von Staaten. Nicht zuletzt die Tatsache, dass der *Cyberwar* jüngst mit konventioneller Verteidigungspolitik verknüpft wurde[1], zeigt, dass man auf staatlicher Ebene sogar dazu bereit ist, diesen neuralgischen Punkt des wirtschaftlichen und gesellschaftlichen Systems auch mit militärischen Mitteln zu verteidigen. Dies unterstreicht die Wichtigkeit sicherer IT-Systeme.

Im Fokus moderner Angriffe stehen einerseits Privatpersonen, denen mit unterschiedlichen Mitteln und Technologien Software auf den eigenen Computer regelrecht untergeschoben werden. Die eingeschleusten Programme können dann relativ unbemerkt aus dem Handlungsumfeld des betroffenen Benutzers

[1] *Guardian (UK), 30.05.2011*: '[Cyberwarfare] is an integral part of the country's armory', *Wallstreet Journal (USA), 31.05.2011*: 'If you shut down our power grid, maybe we will put a missile down one of your smokestacks.', *Rasmussen Reports (USA), 02.06.2011*: '53% [of likely U.S. voters] Say Major Cyberattack Should Be Viewed As Act of War', *Col. David Laplan (Pentagon, USA)*, 31.05.2011: '[A Defense Department strategy for cybersecurity, to be released in June, points to] the idea that attacks in cyber would be viewed the same way that attacks in a kinetic form are now'

z. T. kriminelle, in jedem Falle aber ungewollte Aktionen durchführen. Dies reicht von der Einrichtung von Kommunikationsknotenpunkten (z. B. Mail-Relays[2] oder Web-Proxies[3]) bis hin zur Ausspähung von Kreditkarten- oder sonstigen Kontoinformationen. Neben der rein finanziellen Schädigung des Benutzers besteht also zusätzlich die Gefahr, als vermeintlicher Urheber rechtlich problematischer Handlungen ins Visier der Strafverfolgungsbehörden zu geraten.

Neben den bereits erwähnten Angriffen auf Industrieanlagen mit eher politischem Hintergrund sind solche Anlagen aber auch der Sabotage vor dem Hintergrund der Wirtschaftskriminalität ausgesetzt. Vor allem bei der Wirtschaftsspionage spielen IT-Systeme mittlerweile eine herausgehobene Rolle, da hier nahezu das gesamte Repertoire an unternehmensspezifischen und kritischen Daten gespeichert und verarbeitet wird. Neben den rein buchhalterischen oder operativen Informationen werden meist auch Informationen zu geistigem Eigentum, Patenten und anderen Geschäftsgeheimnissen elektronisch vorgehalten.

Durch die zunehmende Mobilisierung von elektronischen, technisch und funktional hochgradig ausgereiften Geräten, spielt insbesondere im Endverbrauchermarkt der Schutz geistigen Eigentums eine wichtige Rolle. Dies trifft dabei nicht nur auf Software für beispielsweise Spielkonsolen oder Computer zu, sondern auch auf die Betriebssoftware von Videorecordern, Sat-Receivern, MP3-Playern und Mobiltelefonen. Hier gilt es, effizient implementierte Funktionen elektronischer Geräte vor dem Zugriff durch Dritte zu schützen und so einen Wettbewerbsvorteil, der durch diese Funktionen erzielt worden ist, zu sichern.

Der Schutz der genannten Systeme wird derzeit teils mit kryptografischen, teils mit physikalischen Mechanismen durchgesetzt. Jedoch eignen sich die *klassische* und die bisherige *moderne* Kryptografie nur bedingt zur Absicherung aller genannter Aspekte moderner elektronischer Systeme und Komponenten.

[2]ein Mail-Relay leitet e-Mails weiter und ist in der Lage, die Herkunft der Originalnachricht zu verschleiern; ein Angreifer mit Zugriff auf ein solches Relay kann massenhaft Spam unter der Identität des Opfers versenden

[3]ein Web-Proxy leitet Internet-Verkehr weiter und verwendet dazu seine eigene Identität; damit lassen sich lokationsbasierte Beschränkungen und Sicherheitsmechanismen umgehen oder Kommunikation unter falscher Netzwerkidentität führen

Der folgende Abschnitt führt in den gegenwärtigen Stand der Kryptografie ein.

1.1 Einführung: Stand der Kryptografie

Die Kryptologie, bestehend aus der eigentlichen Kryptografie sowie der Kryptanalyse, blickt auf eine recht lange Geschichte zurück. Die *klassischen* Verfahren reichen von altägyptischen Verschlüsselungsmethoden um 3000 v. Chr. bis hin zu den Verfahren von Cäsar, Vigenere und weiteren aus dem ersten und zweiten Jahrtausend und dienten vornehmlich der Chiffrierung von militärischen oder diplomatischen Botschaften. Aus technischer Sicht handelte es sich vorwiegend um einfache Transpositions- und Substitutionschiffren, die im Zeitalter der computergestützten Kryptanalyse keinen wirklichen Schutzeffekt mehr besitzen. Den letzten Meilenstein der klassischen Kryptografie bilden die Verfahren des Zweiten Weltkrieges, deren Sicherheit noch wesentlich darauf beruhte, dass die elektromechanischen Chiffriermaschinen nach geheimen Bauplänen gefertigt wurden.

Die *moderne* Kryptografie fußt ganz allgemein auf zwei Erkenntnissen. Zum einen sollte die Sicherheit eines kryptografischen Verfahrens nicht auf der Geheimhaltung des zugrundeliegenden Algorithmus' basieren, sondern allein auf der Qualität und der Geheimhaltung des Schlüssels. Dieser Grundsatz wurde bereits 1883 vom Niederländer Auguste Kerckhoffs formuliert und ist bis heute gültig. Ein anderes Grundprinzip, das bis heute Bestand hat, wurde 1949 vom amerikanischen Mathematiker Claude Elwood Shannon formuliert. Shannon gilt als Begründer der Informationstheorie, und in seinen Arbeiten setzte er ein starkes mathematisches Fundament für eine gleichermaßen starke Kryptografie voraus. Er wies auch nach, dass eine Verschlüsselung mit perfekter Sicherheit ausschließlich in dem Falle existiert, in dem der Schlüssel dieselbe Länge wie die zu verschlüsselnde Nachricht besitzt und dazu der Schlüssel wirklich zufällig ist[4] und nur ein einziges Mal verwendet wird. Damit können die verwendeten Algorithmen frei zugänglich sein und durch eine breitere (Fach-)

[4]Beim sog. One-Time-Pad wird jedes Bit der Nachricht mit einem Bit des gleichlangen Schlüssels XOR-verknüpft; es kann leicht nachgewiesen werden, dass die Paritäten der Chiffrat-Bits unabhängig vom zu verschlüsselnden Klartext normalverteilt sind, wenn der Schlüssel zufällig (normalverteilt) ist

Öffentlichkeit diskutiert werden. Die Sicherheit der Verfahren kann jedoch in Abhängigkeit von mathematischen Erkenntnissen oder Annahmen formal ausgedrückt werden. Auf diese Weise hat die moderne Kryptografie stets einen engen Bezug zur Komplexitätstheorie, da die den Kryptoverfahren zugrundeliegenden Mechanismen auf bekanntermaßen *schwere* Probleme im Sinne der Komplexitätstherie reduziert werden, um deren Wirksamkeit bzw. Sicherheit zu beweisen. Das Maß an Sicherheit wird meist als Wahrscheinlichkeit ausgedrückt, dass z. B. ein Schlüssel durch einen Angreifer zufällig geraten werden kann.

Im Wesentlichen werden in der modernen Kryptografie zwei fundamentale Ansätze unterschieden. Die zunächst intuitiv eher einleuchtende symmetrische Verschlüsselung verwendet zur Ver- und Entschlüsselung einer Nachricht m denselben Schlüssel s. Es gilt für die Verschlüsselungsfunktion $\mathcal{E}$ und die Entschlüsselungsfunktion $\mathcal{D}$:

$$\mathcal{D}(\mathcal{E}(m, s), s) = m$$

Wichtige Vertreter der symmetrischen Algorithmen sind (der veraltete) DES[5], 3DES[6] und der aktuelle NIST[7]-Standard AES[8]. Ein gängiges Konstruktionsprinzip bei den symmetrischen Verfahren sind die sog. Feistel-Chiffren, benannt nach ihrem Erfinder Horst Feistel [24]. Hier wird die zu verschlüsselnde Nachricht in gleichgroße Blöcke zerlegt und dann mit einem rundenbasierten Ersetzungs- und Vertauschungsverfahren verschlüsselt, wie in Abbildung 1.1 dargestellt. Zuvor werden aus dem eigentlichen Schlüssel in der sog. Schlüsselerweiterung die Rundenschlüssel abgeleitet. Die Sicherheit hängt hier ganz wesentlich von der Qualität der (pseudo-) zufälligen Funktion ab, die die Rundenschlüssel generiert. Kennzeichnend für die Klasse der Feistel-Chiffren ist die Tatsache, dass für die eigentliche Verschlüsselungsfunktion F jede (pseudo-) zufällige Permutation verwendet werden kann, da die Entschlüsselung nicht auf einer Umkehrfunktion derselben beruht, sondern auf den Verschlüsselungsrunden in umgekehrter Reihenfolge. Eine Runde in einem Feistel-Verfahren zur

[5]der Data Encryption Standard gilt wegen der geringen Schlüssellänge als unsicher

[6]3DES ist eine Variante des DES bei dem eine Nachricht mit drei DES-Schlüsseln verschlüsselt wird

[7]National Institute of Standards and Technology

[8]Der Rijndael-Algorithmus wurde im Jahre 2000 als neuer NIST-Advanced-Encryption-Standard für symmetrische Verschlüsselung ausgewählt

Verschlüsselung lautet bezogen auf Abbildung 1.1:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(K_i, R_{i-1})$$

mit L_i als linker Seite des Blocks in Runde i, R_i der rechten Seite des Blocks in Runde i und K_i dem Rundenschlüssel für Runde i. F ist die Zufallsfunktion. Eine Entschlüsselungsrunde in einem Feistel-Verfahren vertauscht die Richtung der Runden, sowie die linken und rechten Seiten der Blöcke wie folgt:

$$L_{i-1} = R_i \oplus F(K_i, L_i)$$

$$R_{i-1} = L_i$$

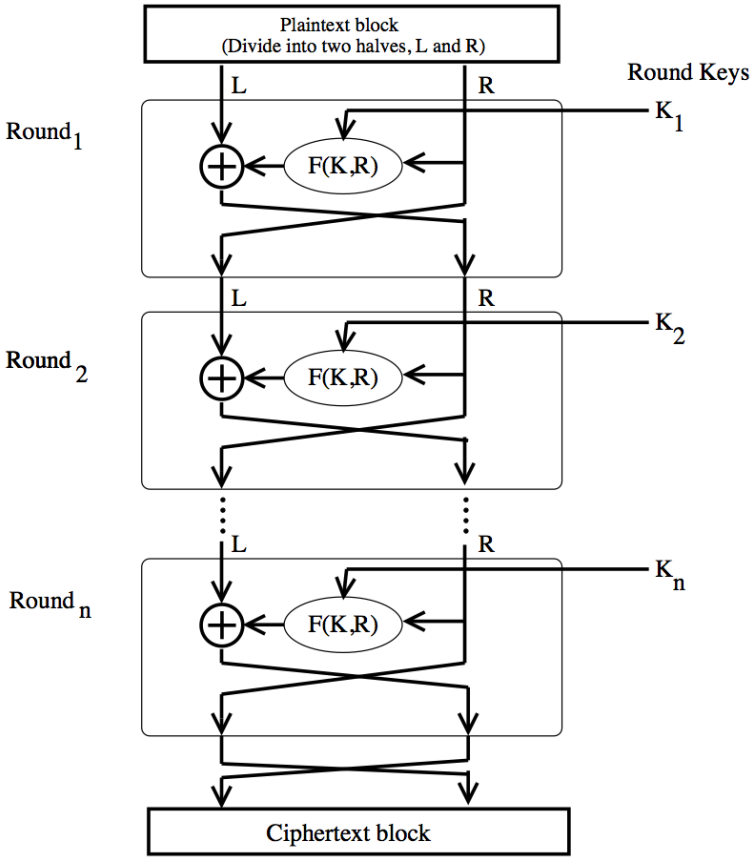


Abbildung 1.1: Struktur einer Feistel Chiffre [39]

Im Gegensatz zum Konstruktionsprinzip der Feistel-Chiffren verwendet AES ein *Substitutions-Permutations-Netzwerk* auf das im Rahmen dieser Einführung nicht näher eingegangen wird. Symmetrische Kryptosysteme bieten vor

allem den Schutz der Vertraulichkeit von Nachrichten. Diese werden also vor dem Lesen durch Dritte geschützt. Obwohl symmetrische Verfahren wie AES (und 3DES) heute für die Absicherung praktisch aller wesentlicher Datenverbindungen verwendet werden, haben symmetrische Kryptosysteme den entscheidenden Nachteil der komplexen Schlüsselverteilung. Wenn n Parteien verschlüsselt miteinander in Kontakt treten wollen, so muss jeder Teilnehmer $n-1$ Schlüssel vorhalten unter der Annahme, dass der Datenverkehr paarweise geheimgehalten werden können soll. Die Schlüssel müssen vor einer verschlüsselten Übertragung von Nachrichten aber zunächst selbst übertragen werden. Dieses fundamentale Problem wird durch die asymmetrische Verschlüsselung gelöst. Im Kosmos der symmetrischen Verfahren kann das System *Kerberos* als Infrastrukturlösung zur symmetrischen Schlüsselverwaltung verwendet werden. Dies ist jedoch nur noch vereinzelt im Einsatz.

Der vertrauliche Austausch symmetrischer Schlüssel wurde 1976 von Whitfield Diffie und Martin Hellman durch ihren Aufsatz *New Directions in Cryptography* [21] revolutioniert. Sie fanden das nach ihnen benannte Diffie-Hellman-Verfahren, bei dem sich zwei Teilnehmer nach dem Austausch einiger Nachrichten auf einen gemeinsamen geheimen Schlüssel einigen. Der Durchbruch war, dass ein Dritter alle ausgetauschten Nachrichten mitlesen kann und dennoch nicht in der Lage ist, den vereinbarten Schlüssel zu berechnen. Das DH-Verfahren basiert auf dem mathematischen *Falltürproblem* des diskreten Logarithmus', das annimmt, dass die Potenzierung im modularen Ring über den natürlichen Zahlen $\mathbb{N}$ leicht ist, die Berechnung des diskreten Logarithmus' im modularen Ring jedoch nur *schwer* im Sinne der Komplexitätstheorie lösbar ist. Der Nachteil des DH-Verfahrens ist ein mögliches Manipulieren durch einen *Angreifer in der Mitte*, der je einen Schlüssel mit den eigentlichen Kommunikationspartnern Alice und Bob aushandelt und dann die Nachrichten von Alice an Bob mit dem mit Alice vereinbarten Schlüssel entschlüsselt, die Nachricht auswertet und dann mit dem mit Bob ausgehandelten Schlüssel verschlüsselt weiterleitet. Alice und Bob bemerken von diesem Angriff nichts, da das DH-Verfahren die Kommunikationsteilnehmer nicht authentifiziert.

Die asymmetrische Kryptografie verwendet zur Verschlüsselung einer Nachricht m einen Schlüssel s und zur Entschlüsselung einen zu s komplementären

Schlüssel s'. Es gilt:

$$\mathcal{D}(\mathcal{E}(m, s), s') = m$$

Wichtig ist, dass eine Nachricht nur mit dem zum Verschlüsselungsschlüssel komplementären Schlüssel entschlüsselt werden kann. Dabei darf der eine Schlüssel nicht leicht aus dem anderen abgeleitet werden können. Dieses Vorgehen hat den entscheidenden Vorteil, dass einer der Schlüssel öffentlich verfügbar gemacht werden kann, sodass jedermann in der Lage ist, eine Nachricht damit zu verschlüsseln. Da der zweite Teil jedoch durch den Besitzer geheim gehalten wird, ist nur der Besitzer in der Lage, die entsprechenden Nachrichten wieder zu entschlüsseln. Dieses Prinzip lässt sich ebenso wie die symmetrische Verschlüsselung zur Wahrung der Vertraulichkeit ausgetauschter Nachrichten verwenden. Außerdem kann dieses Prinzip verwendet werden, um Nachrichten zu signieren, d. h. die Herkunft (genauer: die Authentizität) und die Integrität nachzuweisen. Dazu wird die Nachricht mit dem privaten Schlüssel verschlüsselt, sodass jedermann sie mit dem öffentlichen Schlüssel entschlüsseln kann, um so die Richtigkeit festzustellen. In der Praxis wird jedoch nicht die gesamte Nachricht signiert, sondern nur ein Hash-Wert, also eine kryptografische Prüfsumme. In diesem Zusammenhang stellt sich die Frage, wie ein öffentlicher Schlüssel einer Person glaubhaft zugeordnet werden kann. Dies erfolgt durch Hinzunahme einer vertrauenswürdigen dritten Partei, die die Identität dieser Person und den Schlüsselbesitz bestätigt. Diese Bestätigung wird durch ein sog. digitales Zertifikat bescheinigt, welches den öffentlichen Schlüssel der betreffenden Person enthält und mit dem privaten Schlüssel der Zertifikatsausgabestelle signiert ist. Die Signatur der Zertifikatsstelle (der Vertrauensanker) selbst kann dort eingesehen werden. Der bekannte Standard X.509 regelt die Mechanismen zur Erstellung, Verteilung und Stornierung sowie die Formate von digitalen Zertifikaten. Aufsetzend auf X.509 werden Public-Key-Infrastrukturen (PKI) betrieben. Unter der Voraussetzung, dass jeder Teilnehmer seinen geheimen Schlüssel tatsächlich geheim hält, kann auf diese Weise auch eine Nichtabstreitbarkeit etabliert werden, da sich leicht nachweisen lässt, mit welchem privaten Schlüssel eine Nachricht signiert wurde. Der wichtigste Vertreter der asymmetrischen Kryptografie ist RSA[9], dessen Sicherheit auf dem mathematischen Problem der Primzahlfaktorisierung beruht.

[9]RSA ist nach den Erfindern Rivest, Shamir und Adleman benannt

Das Faktorisierungsproblem stützt sich auf der Annahme, dass die Multiplikation von großen Primzahlen leicht ist, die Zerlegung einer großen Zahl in ihre Primfaktoren aber ein schweres Problem im Sinne der Komplexitätstheorie darstellt.

Der Nachteil der asymmetrischen Verfahren ist die im Hinblick auf symmetrische Schemata vergleichsweise hohe Rechenleistung, die zur Ver- und Entschlüsselung von Nachrichten erforderlich ist. Daher ist es gängige Praxis, hybride Kryptosysteme einzusetzen, bei denen über asymmetrische Kryptografie ein symmetrischer Sitzungsschlüssel ausgetauscht wird, mit dem dann die eigentliche Kommunikation verschlüsselt wird. Durch den einleitenden Schlüsseltausch können auch die Schutzziele der asymmetrischen Kryptografie umgesetzt werden.

Zusammengefasst sind die Schutzziele der klassischen und modernen Kryptografie:

- **Vertraulichkeit:** Informationen werden gegen Ausspähung geschützt und auf diese Weise wird die Privatsphäre der Beteiligten geschützt.
- **Authentizität:** Der Urheber oder Absender einer Nachricht ist eindeutig identifizierbar. Dies ist insbesondere bei Kommunikationsvorgängen relevant.
- **Integrität:** Daten sind fälschungssicher, d. h. vollständig und unverändert. In jedem Fall kann eine Manipulation eindeutig erkannt werden.
- **Nichtabstreitbarkeit:** Besonders im Bereich von Handel und Verträgen ist die Verbindlichkeit einer Nachricht sehr wichtig. Es kann nachgewiesen werden, mit welchem privaten Schlüssel beispielsweise eine Signatur geleistet wurde.

Diese Schutzziele lassen sich mit den bekannten Verfahren jedoch lediglich für *passive Daten* erreichen. Passiv sind solche Daten, die in verschlüsselter Form gespeichert oder übertragen werden. Zur eigentlichen Verarbeitung müssen diese Daten wieder entschlüsselt werden. Die folgenden Abschnitte skizzieren einige Szenarien, in denen die Sicherheit von Daten nicht allein passiv gewährleistet werden kann.

1.2 Motivation: vertrauliches Rechnen

In diesem Abschnitt werden sowohl bereits existierende Einsatzgebiete verschlüsselten Rechnens als auch potenzielle Verwendungen im verteilten Rechnen untersucht. Als verteiltes Rechnen sind solche Szenarien zu verstehen, bei denen in der Regel umfangreiche und rechenintensive Berechnungen in meist parallelisierbare Einzelteile aufgespalten und an unterschiedliche externe Rechenressourcen delegiert werden. Dabei sind verteilte Systeme zu unterscheiden, bei denen zwischen Benutzer und Ressourcen-Provider grundsätzlich ein Vertrauensverhältnis vorliegt (wie beim Grid-Computing) und Systemen, bei denen dies nicht notwendigerweise der Fall ist (wie beim Cloud-Computing).

Aber auch abseits der Kerngebiete der Computer-Kommunikation ist der Bedarf nach vertraulicher Programmausführung zu beobachten. So sind sog. *Hardware-Software-Packages*, also Produkte, die sowohl eine Hardware- als auch eine eingebettete Software-Komponente enthalten, nach der Auslieferung an einen Kunden potenziell der Gefahr ausgesetzt, in dieser Umgebung ohne Einfluss durch den Hersteller bzw. Rechteinhaber *reverse-engineered* zu werden. Bei einem solchen Vorgang wird das Produkt dahingehend untersucht, welche Algorithmen und Daten zum Betrieb und zur Steuerung eingesetzt werden. Das Ziel ist die Rückführung in lesbare Computerprogramme und Funktionsbeschreibungen, die die Interna des Produktes offenlegen und zur unrechtmäßigen Wiederverwendung verfügbar machen. Dieser Bereich der Urheberrechtsverletzung und Wirtschaftsspionage betrifft sowohl den Markt für industriell genutzte Anlagen als auch den Endverbrauchermarkt für Unterhaltungelektronik o. ä.

1.2.1 Verteiltes Rechnen - Grid Computing

Im Forschungsumfeld kommt für die Zwecke des verteilten Rechnens meist das *Grid-Computing* zum Einsatz. Dieses basiert auf einem Ansatz von Ian Foster aus dem Jahre 1998 [25] und stellt eine Analogie zum *Power-Grid*, d. h. dem Gebrauchsstromnetz für Haushalte dar. Die Idee dabei ist, dass die (teils auch fachspezifische) Rechenleistung wie beim Einstecken eines Stromsteckers in einen entsprechenden Anschluss abgerufen werden kann, wobei sich der Leistungskonsument nicht um die technischen Details der Leistungsbereitstellung

und -verteilung zu kümmern braucht. Die Identifikation der Kunden findet im Grid durch SSL Zertifikate statt. Im Gegensatz zur normalen Verwendung von Zertifikaten darf der Benutzer im Grid von seinem eigenen Zertifikat weitere Zertifikate ableiten und diese selbst signieren. Diese in der Regel verhältnismäßig kurzlebigen *Proxy-Zertifikate* (typischerweise 12 oder 24 Stunden im Gegensatz zu mehreren Jahren bei regulären SSL Zertifikaten) dienen dabei der Rechtedelegation an eine Rechenressource und verleihen dieser faktisch zeitlich begrenzt die Identität eines Benutzers. Dabei besteht das Grid systembedingt aus einer Vielzahl von getrennt betriebenen Ressourcen. Technisch gesehen handelt es sich hier um Cluster-Systeme (im D-Grid meist unter Kontrolle eines Batch-Systems wie PBS oder LSF), die durch eine Grid-Middleware gekapselt werden und anhand derer der Benutzer aus den zur Verfügung stehenden Grid-Ressourcen die für eine Aufgabe jeweils geeignete aussuchen kann. Dabei wird der Benutzer seltener lokal bei den Ressourcen registriert, meist kann er durch Identifikation bei einen zentralen Dienst alle Grid-Ressourcen eines Verbundes wie dem D-Grid nutzen.

Beim Grid-Computing besteht grundsätzlich ein Vertrauensverhältnis zwischen einer Grid-Ressource und dem Benutzer, d. h. dieser trifft lediglich Vorkehrungen, die Kommunikation zwischen sich und der Ressource durch Transportverschlüsselung abzusichern. Insbesondere vertraut der Benutzer darauf, dass der Ressourcenbetreiber geeignete Maßnahmen trifft, um die einzelnen Benutzer gegeneinander abzugrenzen und Zugriffe von Nutzern untereinander auf deren Daten zu unterbinden. Kennzeichnend ist das Vertrauen des Benutzers gegenüber dem Ressourcenbetreiber, dass dieser keine unerlaubten Zugriffe auf Programme und Daten des Benutzers ausführt, um diese anderen, als den dazu bestimmten Zwecken zuzuführen.

Praktisch ist dies natürlich ein Problem, und es offenbaren sich im täglichen Gebrauch des Grids im Geltungsbereich deutschen Rechts vielerlei Schwierigkeiten, insbesondere durch die Statuten des Verbundes D-Grid. So verlangte das Förderungsprogramm des Bundes, dass der wissenschaftlichen Forschung Partner aus Industrie und Wirtschaft zur Seite zu stellen seien. Dies diente dem Prinzip der Nachhaltigkeit der Forschung und damit der Weiterverwendbarkeit der Ergebnisse im industriellen Bereich. Gerade diese Partner aber, u. a. Automobilhersteller mit essenziellem Interesse daran, dass aktuelle Berechnungen

zu Karosserie- oder Motorkonstruktionen geheimgehalten werden, sind nicht in der Lage, unter solchen Voraussetzungen dem verteilten, Grid-spezifischen Ansatz zu folgen. In der Folge können solche Partner nur Ressourcenbetreiber anfragen, die spezielle vertragliche Regelungen zur Geheimhaltung und damit für Grid-Ressourcen faktisch nicht zu leistende Anforderungen eines eigenen Rechenzentrums erfüllen.

Durch dieselbe Problematik können beispielsweise Rechenzentren des Banken- und Kreditgewerbes das D-Grid in der derzeitigen Konstellation nicht nutzen, obwohl hier zweifelsohne ein großes Marktpotenzial vorhanden ist. In der Regel hat sich die Leistungsfähigkeit der IT im Kreditgewerbe am Umfang der Ultimoverarbeitung zu orientieren. Dies sind zum einen die Monatsabschlüsse, im speziellen aber auch der Quartals- oder Jahresabschluss, für den gesetzliche Fristen gemäß der Verordnung über die Rechnungslegung der Kreditinstitute und Finanzdienstleistungsinstitute (RechKredV) und des Kreditwesengesetzes (KWG) gelten. Die maximale Laufzeit einer solchen Verarbeitung ist also gesetzlich geregelt und damit indirekt auch die Leistungsfähigkeit der Rechenanlage. Die Dimensionierung der Hardware und Infrastruktur gemäß dieser Voraussetzungen führt zu der Tatsache, dass außerhalb der Abschlussverarbeitung die durchschnittlich Anlagenauslastung nur gering ist. Um eine höhere Wirtschaftlichkeit der Inhouse-IT zu erreichen, wäre die Dimensionierung gemäß der Durchschnittsauslastung wünschenswert, wobei die Lastspitzen dann durch Zukauf im Sinne des Grid-Prinzips abgefedert würden. Durch die technischen Gegebenheiten (unverschlüsselte Verarbeitung) und die rechtliche Situation des Kreditgewerbes ist eine solche Nutzung nicht möglich. Die einzige Lösung ist hier eine vertragliche Regelung, welche jedoch den Grid-Gedanken ad absurdum führt.

1.2.2 Cloud Computing

Die *Cloud* ist wohl das in der jüngeren Vergangenheit am meisten zitierte Stichwort in der IKT-Branche. Sie ist maßgeblich dafür verantwortlich, dass verteilte IT-Systeme in das öffentliche Bewusstsein gerückt sind. Neben der externen Datenspeicherung umfassen Cloud-Services auch zunehmend Dienste, die extern gespeicherte Daten aktiv verarbeiten. Dadurch ergeben sich neue Herausforderungen zur Wahrung von Vertraulichkeit und Privatsphäre

des Anwenders.

Zudem wird die Welt mobiler. Das äußert sich für den Benutzer von elektronischen Geräten vor allem dadurch, dass er plötzlich in der Lage ist, auf seine Fotos, Videoclips, Apps oder sonstige Daten von überall aus zugreifen zu können. Vorausgesetzt, er hat sie in der Cloud abgelegt, um sie dann beliebig abrufen zu können; bevorzugt über Breitband-Funknetze, über die z. B. Smartphones mit dem Internet verbunden sind. Neben der Mobilität hat die Cloud für den Anwender den Vorteil, dass die Daten leicht von allen seinen internetfähigen Geräten erreichbar sind, also beispielsweise auch vom Desktop-Computer. Außerdem werden sogenannte Online-Desktops und Online-Offices angeboten, bei denen der Anwender seine Daten direkt über den Browser auf einem virtuellen Computer im Internet bearbeiten kann. Ein verbreitetes Cloud-Szenario ist die Anmietung externer Rechenleistung und Infrastrukturkomponenten durch professionelle IT-Anwender und -Unternehmen, die diese Rechen-, Netzwerk- und Speicherkapazitäten nutzen, um eigene Ressourcen auszulagern (Stichwort Outsourcing) oder Lastspitzen in der eigenen Verarbeitung durch bedarfsgerechten Leistungszukauf abzufedern. Eine hohe Elastizität kennzeichnet dieses Vorgehen, weil Abruf und Rückgabe von Mietressourcen auch in kurzen Intervallen erfolgen können. In diesem Zusammenhang werden vorwiegend virtuelle Maschinen angefragt, die über preisbestimmende Parameter für CPU-Leistung und Hauptspeichergröße konfiguriert werden. Ergänzend werden Datenspeicher angeboten, die transparent integriert als virtuelle Festplatten fungieren. Es werden hier also nicht nur Daten gespeichert, sondern auch durch Programme auf Fremdressourcen verarbeitet. Die aktive Programmausführung bedeutet derzeit jedoch eine unverschlüsselte Ablage der Programme und der zu verarbeitenden Daten auf der externen Ressource, da die Prozessoren der gemieteten Maschinen den Programmcode nur im Klartext verarbeiten können. Anbieter von Internetdiensten verwenden oft selbst Cloud-Ressourcen, um darauf aufsetzend Mehrwertdienste für den Endanwender zu erstellen. Dadurch wird dieser indirekt auch von der Vertrauenswürdigkeit des Cloud-Providers abhängig. Der Endanwender kann in aller Regel aber nicht technisch kontrollieren, wo genau seine Daten verarbeitet werden. So könnte der tatsächliche Standort Frankfurt, Berkeley County oder Hongkong sein, samt der dort gültigen Rechtsprechung. Dabei haben z. B. US-amerikanische Internet-Unternehmen aufgrund des USA

PATRIOT Act von 2001 alle gespeicherten Daten für die US-Behörden offenzulegen. In vielen Fällen ist es nicht erwünscht oder aus vertragsrechtlichen Gründen nicht möglich, die unverschlüsselte Delegation einer Rechenleistung unter diesen Umständen durchzuführen.

1.2.3 Verbraucherelektronik

Dem Schutz geistigen Eigentums wird im Markt der Verbraucherelektronik zunehmend Beachtung geschenkt. Das betrifft zum Teil die Betriebssoftware von Geräten, vor allem aber den Schutz von Inhalten, wie bei Spielsoftware oder Unterhaltungsmedien. Dabei stößt die praktische Ausführung von Schutzmaßnahmen durch fehlerhaften Einsatz entsprechender Mechanismen immer wieder an ihre Grenzen. Ein prominentes Beispiel hierfür ist das Schutzsystem CSS (Content Scrambling System), welches zur Verschlüsselung von DVD-Inhalten verwendet wird. Durch mehrere Design-Fehler und aufgrund geringer Schlüssellängen wegen der US-Exportbestimmungen über die Ausfuhr von kryptografischen Produkten und Algorithmen wurde der Schutz in kürzester Zeit unwirksam und bereits 2005 sahen Gerichtsurteile[10] im Kopieren derart geschützter Inhalte keinen Rechtsverstoß mehr, weil es sich bei CSS nicht um einen *technisch wirksamen Schutz* im Sinne der entsprechenden Rechtsverordnung handelt.

Auf dem Gebiet der Unterhaltungselektronik für Spiele besteht seit jeher Schutzbedarf, um die Einhaltung des Urheberrechts aktiv durchzusetzen. Beispiele dafür sind im Prinzip alle derzeit erhältlichen Spielkonsolensysteme, wie die Playstation von Sony oder die XBox von Microsoft. Zwar besitzen die Medien der Systeme wirksame Kopierschutzmechanismen durch z. T. herstellerspezifische Modifikationen der Medienformate und Medienlesegeräte, jedoch geht die Raubkopierszene hier einen Schritt weiter und bedient sich Hardware-Lösungen, um die kopierten Medien auf den modifizierten Geräten lauffähig zu machen. Dazu kommen sog. *Mod-Chips* zum Einsatz, i. d. R. kostengünstige Mikrocontroller (für die Sony Playstation 1 war ein PIC 12C508 für etwa einen Euro erforderlich), die mit Schaltlitzen direkt auf das Motherboard der Spiel-

[10] Bericht auf dem finnischen Medienrecht-Blog Turre: *Finnish court rules CSS protection used in DVDs ineffective*: http://www.turre.com/2007/05/finnish-court-rules-css-protection-used-in-dvds-ineffective/

konsole aufgelötet werden. Der Mikrocontroller enthält ein Programm, dass die erforderlichen Bussignale als Ersatz für die Prüfschaltung des Kopierschutzes generiert[11], sodass auch kopierte Medien abgespielt werden.

1.2.4 Industrieelektronik

Seit der Entdeckung der Computerwürmer *Stuxnet* in 2010 und *Flame* in 2012 ist klar, dass diese Art von Bedrohung nicht mehr allein auf den Verbrauchersektor beschränkt ist, sondern dass hier durch kriminelle Technologie massiv Einfluss auf den industriellen Bereich genommen werden soll. Wegen der technischen und organisatorischen Qualität der mit diesen Werkzeugen durchgeführten Angriffe wird das Potenzial auch zur politischen Einflussnahme offenbar, weswegen dem Vorfall die entsprechende Bedeutung beigemessen wird[12]. Tatsächlich betroffen waren in dem Fall Steuerungsanlagen *Simatic S7* der Firma Siemens, auf denen technische Prozesse mit dem SCADA-System betreut werden. Die entsprechende Brisanz gewinnt der Fall dadurch, dass nukleare Industrieanlagen des Iran befallen waren und sich somit die grundsätzliche Frage nach der Sicherheit spezieller industrieller Anlagen stellt. Das Besondere an der Schadsoftware ist der als ungewöhnlich hoch eingeschätzte Aufwand, der sich in der Komplexität und *Qualität* der Software und des Infektionsweges spiegelt. So wurden zur Authorisierung der Installation auf den entsprechenden Windows-Computern gestohlene, von VeriSign ausgestellte digitale Zertifikate u. a. des Netzwerkausrüsters Realtek verwendet, um die Software zu signieren. Außerdem wurden etliche seinerzeit noch unbekannte Sicherheitslücken (*Zero-Day Exploits*) in unterschiedlichen Windows-Versionen ausgenutzt, um Code in

[11] Bei der Playstation 1 sendet der Mod-Chip kurz nach dem Reset mit genauem Timing zyklisch die Strings 'SCEE' für Sony Computer Entertainment Europe, 'SCEA' für SCE Amerika und 'SCEI' für SCE Inc. (Japan) auf den Datenbus des Geräts, bis der Bootvorgang beginnt. Je nach Herkunftsland des Mediums wären das die entsprechenden Strings, die das Gerät vom ersten Sektor des Mediums lesen würde. Da diese Sektoren jedoch nicht im Standardformat auf den Originalmedien vorliegen, können sie von handelsüblichen CD-Brennern nicht fehlerfrei reproduziert werden. Der Mod-Chip sorgt dafür, dass die bei Reset ablaufende Prüflogik dennoch mit den erwarteten Informationen versorgt wird.

[12] Die Süddeutsche Zeitung schrieb am 1. Oktober 2010 auf ihrer Webseite 'Die Büchse der Pandora ist geöffnet', die Frankfurter Allgemeine Zeitung ebenfalls auf ihrer Webseite am 22. September: 'Der digitale Erstschlag ist erfolgt'

eine Software-Bibliothek zur Prozessvisualisierung einzuschleusen (Win-CC). Letztendlich führte die Schadsoftware dazu, dass die Funktion der Zentrifugen zur Urananreicherung durch Beeinflussung von Frequenzumformern, welche die Drehgeschwindigkeit regeln, gestört wurde. Technische Details zum Thema Stuxnet hat die Firma Symantec in einem Dossier [47] veröffentlicht.

1.3 Wissenschaftlicher Beitrag

Diese Arbeit untersucht Verfahren, mit denen aktive Systeme zur Laufzeit durch homomorphe Kryptografie abgesichert werden können. Dabei liegt der Fokus auf der Ermittlung der unterschiedlichen Charakteristika der einzelnen Methoden im Bezug auf Leistungsfähigkeit und konstruktiver Komplexität der resultierenden Lösungen.

- Diese Arbeit enthält die Formalisierung eines begrenzt homomorphen (somewhat homomorphic) Kryptoschemas über der Menge $\mathbb{N}$ mit einer Reduktion auf das Faktorisierungsproblem. Dies erbringt den Nachweis, dass selbst mit einem mathematischen Unterbau von vergleichsweise geringer Komplexität eine wirksame Methode zum Schutz von Algorithmen mit begrenzten Umfang möglich ist. Außerdem werden etliche algorithmische Elementarkomponenten vorgestellt, die auf Basis des vorgestellten Kryptoschemas realisierbar und miteinander kombinierbar sind. Mit Hilfe der identifizierten Elemente wird die Suche als Schlüsselkonzept vieler wissenschaftlicher Anwendungen in unterschiedlichen Ausprägungen als homomorph verschlüsselter Vorgang konzipiert. Dabei finden wesentliche Varianten von Suchalgorithmen Beachtung, wie die exakte Suche (Dictionary Search), die Suche auf Datenströmen (Stream Search) und die tolerante Suche (Fuzzy Search) am Beispiel von Gensequenzen.

- Im Rahmen dieser Arbeit wurde die erste frei verfügbare Implementierung eines unbegrenzt homomorphen (fully homomorphic) Kryptoschemas realisiert. Dadurch ist es erstmals möglich, die Leistungsfähigkeit von homomorph verschlüsselten Algorithmen und Schaltkreisen im realen Umfeld zu untersuchen und tatsächlichen Messungen zuzuführen. Durch die Freigabe unter einer Open-Source Lizenz kann auch zum

ersten Mal eine breitere Öffentlichkeit experimentelle Arbeiten zur unbegrenzt homomorphen Kryptografie durchführen, um auf diese Weise den gesamten Themenkomplex in die Praxis zu überführen. Hierdurch wird parallel zur mathematischen Grundlagenforschung in der homomorphen Kryptografie ein informatiknaher Forschungszweig unterstüzt, der einerseits die theoretischen Konstruktionsgrundlagen verschlüsselter Algorithmen aufbauend auf diese Arbeit vertieft, wie auch die Überführung ins Software-Engineering untersucht. Das Dissertationsprojekt **hcrypt** stellt alle Ergebnisse als Software und Dokumentation auf der Webseite *http://www.hcrypt.com* zur freien Verfügung.

- Einer der Hauptbeiträge dieser Arbeit ist die erste Konstruktion eines vollständig verschlüsselten Mikroprozessorsystems mit verschlüsselter Zentraleinheit und verschlüsseltem Speicher. Ausgehend von Elementarschaltkreisen, wie einem homomorph verschlüsselten Demultiplexer werden die unterschiedlichen Einheiten eines Prozessors (arithmetisch-logische Einheit, Steuereinheit, Speichereinheit) modularisiert, sodass diese entsprechend den jeweiligen Anforderungen zusammengesetzt werden können. Zum Nachweis der Tragfähigkeit des Konzeptes wird eine beispielhafte Prozessorarchitektur vorgestellt, anhand derer Leistungskennzahlen ermittelt werden. Zur Sicherung der Ergebnisse wurden in Zusammenarbeit mit dem Erfinderzentrum Niedersachsen (EZN) und der Leibniz Universität Hannover Patente für den deutschen und den europäischen Raum angemeldet.

- Vollständig homomorph verschlüsselte Algorithmen weisen eine verhältnismäßig begrenzte Leistungsfähigkeit in Bezug auf Zeit- und Platzbedarf verglichen mit unverschlüsselten Algorithmen auf. In dieser Arbeit wird als Konzept zur Leistungssteigerung der Begriff der hybriden Algorithmen erstmals eingeführt und formalisiert. Diese Form von Algorithmen besteht zum größten Teil aus unverschlüsselten und daher höher performanten Elementen. Um dem Schutzziel Vertraulichkeit gerecht zu werden, sind jedoch die wesentlichen Algorithmenteile und Daten verschlüsselt, sodass bei dieser Form von Programmschutz einem optimalen Protokoll-Design besondere Bedeutung zukommt. Zur Verdeutlichung

der Idee wird die exakte Suche auf Basis des hybriden Konstruktionsprinzips nochmals konzipiert.

- Im digitalen Zeitalter wächst der Anteil an Waren und Dienstleistungen, die auf elektronischem Wege bezahlt werden. Im Schlussteil dieser Arbeit wird ein Bankenszenario skizziert, in dem Zahlungen in Form von Überweisungen in vertraulicher Höhe getätigt werden können. Neben den Transaktionshöhen sind auch die Kontensalden stets verschlüsselt, wobei eine Zentralstelle in die Lage versetzt wird, die Deckung der Transaktionen und Konten zu jedem Zeitpunkt zu garantieren, ohne diese jedoch entschlüsseln zu können. Alle Methoden und Protokolle basieren auf homomorpher Kryptografie und orientieren sich vom Konstruktionsprinzip her am Begriff der hybriden Systeme.

1.4 Verwandte Arbeiten

Dieser Abschnitt erstellt eine Übersicht über verwandte Arbeiten mit einem Schwerpunkt auf homomorpher Kryptografie und sicherem Rechnen bzw. Rechnen in unsicherem Umfeld. Dies ermöglicht eine Einordnung in den gegenwärtigen wissenschaftlichen Kontext. Zum jetzigen Zeitpunkt ist nach bestem Wissen kein wissenschaftliches oder industrielles Projekt bekannt, das homomorphe Kryptografie verwendet, um Prozessor- und Speicherelemente zu modellieren. Über das militärische PROCEED-Projekt des US-amerikanischen Verteidigungsministeriums sind bisher keine entsprechenden Ergebnisse veröffentlicht, daher können nur die Anforderungen skizziert werden.

Einzelpublikationen

Eine wichtige Grundlagenarbeit, auf der viele Konzepte in dieser Arbeit beruhen, ist die erste Konstruktion eines unbegrenzt algebraisch-homomorphen (fully homomorphic) Schemas durch Gentry [27]. Das Besondere an der Konstruktion ist das elegante *Bootstrapping*-Verfahren, um aus einem begrenzt homomorphen Schema ein unbegrenzt homomorphes zu erzeugen und bei dem die fortschreitende Verrauschung der Operanden durch eine *Recrypt*-Operation verringert wird. Dabei wird diese Operation selbst im Kryptoraum ausgeführt

und liefert eine weniger verrauschte Repräsentation des eingegebenen Operanden, ohne diesen entschlüsseln zu müssen. Dadurch wird eine unbegrenzt lange Verkettung von Operationen auf verschlüsselten Operanden ermöglicht. Nach demselben Prinzip arbeiten einige weitere Arbeiten von Smart et al. [55] und van Dijk et al. [22], die zwar eine Spezialisierung der mathematischen Grundlage (Smart) bzw. eine abgewandelte Grundlage (Ganzzahlen statt Gitter bei van Dijk) anwenden, jedoch auch die Idee des Bootstrappings aufgreifen. Wegen des auftretenden rechnerischen Aufwandes zur Laufzeitkorrektur der Operanden stellt sich die Frage, ob mit den begrenzt homomorphen Schemata, die die Basis für die unbegrenzten bilden, nicht ebenfalls praktische Aufgaben berechnet werden können. Dieser Ansatz wird unter anderem in Naehrig et al. [45] untersucht. Die unbegrenzt homomorphen Schemata werden gegenwärtig weiter untersucht. Hier sind folgende Richtungen zu beobachten: Einerseits werden in Stéhle et al. [56] neue Annahmen zur Lösung schwerer Probleme im Sinne der Komplexitätstheorie vorgeschlagen, die einen positiven Effekt auf die Performance homomorpher Kryptografie haben, andererseits werden die existierenden Schemata weiter spezialisiert, um die Platzkomplexität zu verringern (Coron et al. [18]). Schließlich werden weitere Schemata abseits von Gentrys ursprünglichem Konstruktionsvorschlag untersucht, die Alternativen zum Bootstrapping vorschlagen, so etwa Brakerski et al. [8] und Gentry et al. [29]. Hier wird u. a. die Technik des *Modulus-Switching* beschrieben, wobei nicht das Rauschen der Operanden verringert, sondern der Überlaufschwellwert verschoben wird. Eine exzellente Zusammenfassung des aktuellen Standes der unbegrenzt homomorphen Kryptografie findet sich in Vaikuntanathan [58].

Im Bereich der begrenzt homomorphen Kryptosysteme werden hauptsächlich algorithmische Basiskomponenten untersucht. So werden verschlüsselte Vergleiche ganzer Zahlen in Garay et al. [26] und Blake et al. [6] am Beispiel von Yaos Millionärsproblem adressiert, wobei vor allem die Kommunikationskomplexität reduziert werden soll. Damgård et al. [19] beschreiben eine Methode zum Gebotsvergleich bei Online-Auktionen basierend auf einem homomorphen Schema in Untergruppen von $\mathbb{Z}_n^*$ nach Groth et al. [35]. Eine sehr gute und äußerst detailreiche Zusammenfassung begrenzt homomorpher Kryptosysteme findet sich in Henry [37].

Neben der homomorphen Kryptografie werden andere Paradigmen zur For-

mulierung verschlüsselt ausführbarer Funktionen (*Secure Funktion Evaluation - SFE*) untersucht. Die meisten dieser Arbeiten basieren auf *Yao's Garbled Circuits* [60], die eine Funktion als booleschen Schaltkreis mit verschlüsselten Schalttabellen darstellen. Hierzu existiert eine Reihe von Erweiterungen zur praktischen Anwendung durch Malkhi et al. [44] oder zur effizienten Darstellung von Gatterfunktionen und Elementarschaltungselementen durch Kolesnikov et al. [41, 42, 43]. *Yao's Garbled Circuits* sind in Gentry et al. [30] mit homomorpher Kryptografie verknüpft worden, wobei diese als generische Schaltkreisdarstellung für Funktionen verwendet wurden. Im Bereich der algorithmischen Ansätze beschreiben Bain et al. [4] eine domain-spezifische Programmiersprache zum Rechnen auf verschlüsselten Daten.

Ein Ansatz aus der theoretischen Informatik zur Verschleierung von Speicherzugriffsmustern bei der Ausführung von Funktionen ist ein spezieller Typ von Turing Maschine, die *Oblivious Random Access Machine* (ORAM) von Goldreich et al. [31, 32]. Hier werden Algorithmen auf Turing-Maschinen reduziert, bei denen je zwei Speicherzugriffe nicht voneinander zu unterscheiden sind. So soll verschleiert werden, auf welchen Speicherbereich tatsächlich zugegriffen wird. Zu diesen Arbeiten existieren weitere von Pinkas et al. [50] zur Verringerung der Komplexität und weitere Entwicklungen zu speziellen Anwendungen von Damgård et al. [20] und Goodrich et al. [33]. Durch die besondere Struktur der homomorphen Programmausführung ist das Konzept *Oblivious Access* in einer Abwandlung auch Gegenstand dieser Arbeit (siehe auch Abschnitt 5.3). Konstruktionen mit booleschen Schaltkreisen wurden anhand von speziellen Turing-Maschinen, u. a. den *Parallel Random Access Machines (PRAM)* [59], untersucht. Auf der Grundlage dieses und ähnlicher theoretischer Modelle sind so Aussagen zum generellen Laufzeitverhalten und insbesondere zur Komplexität bestimmter Schaltkreis- und Funktionsgruppen möglich. Ein Bezug zu verschlüsselten Schaltkreisen oder zur konkreten Modellierung einsatzfähiger Prozessorkomponenten wird jedoch nicht hergestellt.

Auf der Hardware-Seite liefern Infineon mit der SLE78-Serie oder Maxim mit der DS5250-Serie die Klasse der *Secure Microcontroller*. Laut Datenblättern werden sowohl Daten- als auch Adressbus dieser Controller verschlüsselt, indem der lineare Adressraum auf eine randomisierte bijektive Permutation abgebildet wird und die Daten in Abhängigkeit von Inhalt und Adresse ver-

schlüsselt werden. Leider sind die Hersteller auf Anfrage nicht bereit, die Tragfähigkeit der Konzepte einer wissenschaftlichen Überprüfung durch die Gruppe DCSec der Leibniz Universität Hannover zur Verfügung zu stellen. Die öffentlich verfügbaren Informationen besitzen die inhaltliche Qualität von Werbebroschüren.

thep - The Homomorphic Encryption Project

Das thep-Projekt[13] bietet eine Java-Implementierung des begrenzt homomorphen Paillier-Schemas (siehe auch Kapitel 3). Hauptanliegen des Projektes ist die Schaffung einer Grundlage zum Experimentieren mit homomorpher Kryptografie. Außerdem bietet das Projekt eine Implementierung der Oblivious-Transfer Variante *GT-SCOT* (Strong Conditional Oblivious Transfer and Computing on Intervals) von Blake et al. [7]. Für die meisten der in dieser Arbeit vorgestellten Konzepte sind die durch das thep-Projekt bereitgestellten Methoden nicht ausreichend. Die insgesamt betrachteten Themen beschränken sich auf die beiden genannten praktischen Aspekte und haben bei weitem nicht den Umfang und die konzeptionelle Breite des dieser Arbeit zugrundeliegenden hcrypt-Projektes.

PROgramming Computation on EncryptEd Data (PROCEED)

Die Forschungsbehörde Defense Advanced Research Projects Agency (DARPA) des US-Verteidigungsministeriums hat unter der Ankündigung *Broad Agency Announcement DARPA BAA-10-81* vom 6. Juli 2010 Fördermittel[14] für das Projekt PROCEED ausgelobt. Dieser Unterabschnitt enthält einen Auszug aus der Projektbeschreibung im Original.

Der geplante Umfang des PROCEED-Projektes umfasst auch Themengebiete, die im hcrypt-Projekt zu dieser Dissertation schon vor dem PROCEED-Call formuliert wurden. So sind die folgenden Bereiche auch Gegenstand dieser Arbeit:

[13] http://code.google.com/p/thep/
[14] Größenordnung: 20 Mio. USD

- Programmschutz: der Schutz des Codes eines laufenden Programms vor der Einsichtnahme Dritter
- Implementierung / Optimierung: Beschleunigung verschlüsselter Programme, ggf. durch Hinzunahme von Hardwarekomponenten
- Algorithmen: Entwicklung spezieller Teilalgorithmen, die eine Verschlüsselung effizient einsetzen

Aufgrund der bisherigen Publikationslage kann zum Zeitpunkt der Fertigstellung dieser Arbeit kein Abgleich mit den Ergebnissen des PROCEED-Projektes vorgenommen werden.

⋆

PROgramming Computation on EncryptEd Data (PROCEED)
The Defense Advanced Research Projects Agency is soliciting proposals for innovative research in programming computation on encrypted data. The proposed research should investigate innovative approaches that enable revolutionary advances in science, devices, or systems. Specifically excluded is research that results primarily in evolutionary improvements to the existing state of practice.

Introduction

The goal of the PROCEED research effort is to develop practical methods for computation on encrypted data without decrypting the data and to develop modern programming languages to describe these computations. PROCEED is a comprehensive research effort with six primary research thrusts:

- Mathematical Foundations of Fully Homomorphic Encryption – Discovery and development of new mathematical underpinnings for efficient computation on encrypted data is needed in a noninteractive setting. The solution might involve fully homomorphic encryption [Gentry09, Gentry10, Smart10] that allow noninteractive computation on encrypted data. This area is captured in RA-10-80, and interested proposers are referred to that solicitation.

- Mathematical Foundations of Secure Multiparty Computation – Discovery and development of new mathematical underpinnings for efficient computation on encrypted data is needed in an interactive setting. Secure multiparty computation [Yao86, Bickson10] has a rich history of interactive computation on encrypted data, but requires further improvements to be truly practical.

- Mathematical Foundations of Supporting Security Technologies – Computation on encrypted data preserves the confidentiality of the data being computed on, but does not inherently protect the integrity of the computation, nor provide strong protection of the program, among other potentially desirable security goals. Techniques to address these and other related security issues are sought in the PROCEED research effort.

- Implementation/Measurement/Optimization – To make computation on encrypted data practical, highly optimized implementations, possibly including programmable hardware, will be needed. Experience shows there can be at least an order of magnitude difference in the performance of highly optimized cryptography implementations over less sophisticated implementations.

- Algorithms – Practical computation on encrypted data will require libraries of data structures and algorithms that are optimized for efficiency in the encrypted domain. Most current approaches to computation on encrypted data work by turning a program (with a bounded maximum input size) into a circuit. An important goal for optimization is minimizing circuit depth, which is traditionally a goal of hardware designers, not programmers.

- Programming Languages – More advanced languages are sought, with type systems that embed cryptographic knowledge, making programming computation on encrypted data no more difficult than conventional programming. Today's languages for computation on encrypted data, such as the one in the FairPlay system [Malkhi04] are simple, imperative languages that have little, if any, type system support for cryptography.

PROCEED will have a research integrator role to define a common cryptographic application programming interface (API), ensuring the compilers and cryptography implementations are interoperable and to support the Government's evaluation team.

Program Scope

The scope of the PROCEED research effort includes any and all technologies related to the following technical areas of interest:

1. Mathematical foundations of fully homomorphic encryption
2. Mathematical foundations of computation on encrypted data via secure multiparty computation
3. Mathematical foundations of supporting security technologies
4. Implementation/Measurement/Optimization of homomorphic cryptography and secure multiparty computation protocols
5. Algorithms for computation on encrypted data
6. Programming languages for computation on encrypted data
7. Integration and evaluation of the above research areas

The PROCEED research effort is focused on general-purpose computation such as higher-order mathematical algorithms, sorting, or processing relational database queries. More formally, one definition of general-purpose computation is recognizing languages in the Chomsky hierarchy: regular languages, context-free languages, context-sensitive languages, and recursively enumerable languages. Constructions that recognize these languages with greater efficiency than more general solutions, or a proof that no such construction exists, are within the scope of PROCEED. Special purpose constructions (e.g., string search and private information retrieval) are outside the scope of PROCEED. Secure multiparty computation protocols that compute on shares of data that are not, strictly speaking, encrypted are within the scope of PROCEED. Such proposals, however, must include an explicit threat model and cogent security argument for the proposed system. If the data is split among n almost identical

systems, why all the shares will not be compromised by the same underlying vulnerability in the system must be explained. Proposals for technologies to complement the confidentiality provided by computation on encrypted data (e.g., integrity) are within the scope of PROCEED. Such proposals should propose significant new research, not rehashes of industry standard techniques applied to the PROCEED problem domain. Proposals should be targeted to the most appropriate technical area. [...]

⋆

1.5 Dokumentorganisation

Kapitel 2 führt in die Grundlagen für sicheres Rechnen ein. Von der in diesem Kapitel skizzierten Motivation ausgehend werden unterschiedliche Technologien für sicheres und verschlüsseltes Rechnen herausgearbeitet und anhand der jeweils erreichbaren Ziele gegeneinander abgegrenzt. Es werden sowohl nichtkryptografische Technologien wie Code Obfuscation als auch kryptografische Technologien betrachtet. Außerdem werden bereits existierende Produktlösungen im Hard- und Softwarebereich untersucht.

Die homomorphe Kryptografie wird in Kapitel 3 eingeführt. Dazu wird zunächst anhand eines einfachen homomorphen Schemas über den ganzen Zahlen der grundsätzliche Mechanismus erläutert. Im weiteren Verlauf des Kapitels werden etablierte homomorphe Schemata vorgestellt und in Theorie und Implementierungspraxis vertieft. Eine Reihe einfacher Protokolle, die mit Hilfe homomorpher Kryptografie gesichert werden können, schließen das Kapitel ab.

Kapitel 4 zeigt in Anwendung der Grundlagen aus Kapitel 3 eine konkrete, praktische Anwendung homomorpher Kryptografie. Es wird ein homomorph verschlüsseltes Suchschema vorgestellt, bei dem mit verschlüsselten Suchbegriffen auf verschlüsselten Daten operiert werden kann. Dabei werden neben einer exakten Suche in Datenbanken und Wörterbüchern auch die exakte Suche im Datenstrom sowie die *fuzzy* Suche (eine tolerante Suche mit unscharfen Suchbegriffen) behandelt. Das Kapitel erbringt den Nachweis, dass homomorphe Kryptografie grundsätzlich anwendbar ist und sich ganz praktische Anwendungen damit absichern lassen.

Mit einer vollständigen Abstraktion homomorpher Kryptografie setzt sich Kapitel 5 auseinander. Hier werden unterschiedliche mikroelektronische Schaltungsmodule zur Abbildung einer verschlüsselten Maschine bestehend aus verschlüsseltem Prozessor und verschlüsseltem Speicher in verschlüsselbare, arithmetische Repräsentationen überführt. Diese Abstraktion hat das Potenzial, den gesamten kryptografischen Unterbau vor dem Anwendungsprogrammierer zu verbergen und kann in Software oder Hardware simuliert werden. Dieses Kapitel erläutert die Konstruktion, die Implementierungsdetails und Leistungskennzahlen der Referenzimplementierung in Software.

Die praktischen Erfahrungen aus den Kapiteln 4 und 5 werden in Kapitel 6 aufgegriffen und zu einer Strategie zur Steigerung der Leistungsfähigkeit von homomorph verschlüsselten Algorithmen und Daten ausgeweitet. Das konkrete Ziel des Kapitels ist die Formulierung und Skizzierung eines Konstruktionsprinzips für verschlüsselte Algorithmen bei dem nur ein sehr geringer Anteil tatsächlich homomorph verschlüsselt wird. Der weitaus größere Anteil dieser *hybriden Systeme* läuft unverschlüsselt ab und trägt so zur Leistungsverbesserung bei. Dabei sind die algorithmischen Bestandteile so zu wählen, dass ein festzulegendes Sicherheitsniveau erhalten bleibt.

In Kapitel 7 werden Analysen und Lösungen für ein weiteres Anwendungsfeld der hybriden homomorphen Kryptografie vorgestellt. In einem Szenario des elektronischen Zahlungsverkehrs werden Teile der bisher entwickelten verschlüsselten Algorithmen verwendet, um eine Kontoführung mit verschlüsselten Salden und Überweisungsbeträgen zu realisieren. Im Gegensatz zu Kapitel 4 werden hier jedoch nicht die vorhandenen Algorithmen lediglich homomorph verschlüsselt, sondern sehr einfache algorithmische Module durch geeignetes Arrangement in teils unverschlüsselten Protokollen gemäß des Konstruktionsprinzips aus Kapitel 6 effizient eingesetzt.

Kapitel 8 schließt diese Arbeit mit einer Zusammenfassung und einem Ausblick auf mögliche Vertiefungen in unterschiedliche Richtungen durch nachfolgende Arbeiten ab.

Kapitel 2

Grundlagen & Techniken für sicheres Rechnen

Um die Kommunikation zwischen zwei Teilnehmern zu sichern existieren bereits viele seit Jahren praxiserprobte Verfahren und Produkte wie SSL oder IPSec. Bei der Delegation von Berechnungen ist also das Problem der Wahrung vieler Schutzziele bei der *Übertragung* der notwendigen Daten schon gelöst. Auch **auf** dem verarbeitenden System gibt es viele Ansätze, vertrauliche Programme und Daten zu schützen (Zugriffskontrolle, Firewalls, verschlüsselte Datenspeicher, etc.). Anders sieht es allerdings bisher bei dem Schutz **vor** dem verarbeitenden System aus, denn in den bisherigen Ansätzen muss dem System – genauer: dem Systemadministrator oder Hersteller – vertraut werden.

Die Abschnitte dieses Kapitels führen in unterschiedliche Technologien zur Wahrung der Vertraulichkeit verarbeiteter Daten und Programme ein und erörtern die Defizite im Bezug auf einen kryptografisch nachweisbaren Schutzeffekt.

2.1 Code Obfuscation

Moderne Kryptografie basiert im Hinblick auf die Sicherheit immer darauf, dass diese von der Qualität eines Schlüssels abhängt[1]. Das bedeutet aber auch, dass die Entschlüsselung einer Chiffre, bzw. das Finden eines geheimen Schlüssels

[1]Das Kerckhoffs'sche Prinzip besagt, dass die Sicherheit einer Verschlüsselung von der Geheimhaltung des Schlüssels und nicht des Verschlüsselungsalgorithmus' abhängen soll

nicht grundsätzlich ausgeschlossen ist, sondern dass die *durchschnittliche Dauer* dafür mathematisch beweisbar quantifiziert werden kann. Der entsprechende Schlüssel muss dann so gewählt werden, dass der zu erwartende Aufwand zum Brechen der Chiffre die gegenwärtigen Möglichkeiten signifikant übersteigt.

Bei der *Code Obfuscation* (Programmverschleierung) wird dagegen kein mathematischer Maßstab angelegt, sondern ein menschlicher. Es soll nämlich erreicht werden, dass der Betrachter eines Abschnittes Quellcode, eines Disassembler-Listings oder eines Speicherauszuges nicht in der Lage ist, den Sinn des vorliegenden Programmablaufes zu verstehen. Die hierdurch erreichbare Sicherheit ist natürlich in hohem Maße subjektiv und von etlichen Faktoren abhängig, u. a.:

- Welche Hilfsmittel stehen dem Betrachter zur Verfügung? Spezialisierte Werkzeuge erleichtern das Verständnis des Codes.
- Wie groß ist der Nutzen für den Betrachter? Dies beeinflusst maßgeblich die Einsatzbereitschaft von Zeit und Mitteln.
- Welche Erfahrung besitzt der Betrachter?

Derzeit existieren mehrere Ansätze zum Verschleiern von Programmabläufen. Das Spektrum reicht von unleserlichen Sourcecode-Fragmenten über Liveverschlüsselte Programmteile bis hin zur Verwendung undefinierter Opcodes auf Maschinencode-Ebene. Die Wirkungsweise dieser Maßnahmen wird im Folgenden jeweils im Ansatz erläutert, wobei auch die Gegenmaßnahmen eines erfolgreichen Betrachters entgegengestellt werden.

2.1.1 Beispiel: Java Obfuscation

Am Beispiel der Programmiersprache Java soll eine sehr einfache Form der Code Obfuscation demonstriert werden. Dazu sind neben des gängigen *Java Development Kits* (JDK)[2] die frei zugänglichen Werkzeuge *Java Decompiler*[3] und *ProGuard*[4] erforderlich. Listing 2.1 zeigt zwei einfache abhängige Klassen, bei der die Klasse `ClassA` eine Funktion der Klasse `ClassB` aufruft.

[2]Oracle JDK: http://www.oracle.com/technetwork/java/index.html,
OpenJDK: http://openjdk.java.net

[3]http://java.decompiler.free.fr

[4]http://proguard.sourceforge.net

Listing 2.1: Java Klassen

```
1  package test.classes;
2
3  class ClassA
4  {
5          public static void main(String [] args)
6          {
7                  new ClassB().print("Hello World.");
8                  System.out.println("\n");
9          }
10 }
11
12 class ClassB
13 {
14         public void print(String s)
15         {
16                 System.out.print(s);
17         }
18 }
```

Die beiden Klassen können in ein Java Archive (JAR) als auslieferbare Applikation verpackt werden und sind durch einen einfachen Aufruf zu starten. Ebenso leicht ist allerdings die Anwendung eines Java Decompilers auf die Archiv-Komponenten, sodass die beiden Class-Dateien in ihren ursprünglichen Quelltext zurückübersetzt werden. Bis auf technisch unerhebliche Unterschiede, wie der generischen Umbenennung von Funktionsparametern, ist das Dekompilat sehr gut lesbar, und die Programmstruktur wird leicht deutlich, wie in Listing 2.2 gezeigt. Mit eventuellen Änderungen könnten die Quelldateien wieder kompiliert werden.

Listing 2.2: Java Decompiler Ausgabe

```
1  package test.classes;
2
3  import java.io.PrintStream;
4
5  class ClassA
6  {
7    public static void main(String[] paramArrayOfString)
8    {
9      new ClassB().print("Hello World.");
10     System.out.println("\n");
11   }
12 }
13
14
15 package test.classes;
16
```

```
17 import java.io.PrintStream;
18
19 class ClassB
20 {
21   public void print(String paramString)
22   {
23     System.out.print(paramString);
24   }
25 }
```

Um diese leichte Verfügbarkeit des Quelltextes zu verhindern, wird Obfuscator-Software eingesetzt. Vor der Auslieferung des Applikations-Archivs wird in diesem Beispiel die Software ProGuard darauf angewendet, um einige signifikante Änderungen vorzunehmen, die jedoch stets in einen äquivalenten Maschinencode (bzw. in diesem Falle Java Bytecode) münden müssen. Ein typischer Durchlauf einer solchen Software umfasst häufig mehrere oder alle der folgenden Schritte:

1. Shrinking: während der Schrumpfung werden alle nicht benötigten Imports, Klassen und Methoden verworfen, sodass ein kompakterer Code entsteht

2. Optimizing: die Optimierung wird u.a. durch Verwerfen von unbenutzten Methoden-Parametern, Anpassung der Modifier wie `public`, `static` oder `final` um Zugriffe zu beschleunigen und Speicherplatz zu sparen, oder durch *Inlining*[5] von Methoden durchgeführt

3. Obfuscation: der eigentliche Schutz wird durch generische Umbenennung aller Klassen und Variablen erzielt; Hauptziel dabei ist es, den bereits optimierten und kompakten Code weiter unleserlich zu machen

4. Preverification: die Bytecode-Prüfung soll sicherstellen, dass der entstandene Code den Konventionen einer Java Virtual Machine entspricht

In Listing 2.3 ist die dekompilierte Ausgabe des Obfuscators zu sehen. Deutlich zu erkennen ist, dass alle Namenskomponenten verkürzt dargestellt werden. Außerdem wird die Package-Struktur soweit wie möglich eliminiert. Offensichtlich kann dies nur bei Klassen erfolgen, die zu keinem Interface gehören,

[5] bei selten benutzen Methoden wird jeder Aufruf durch den Methodenrumpf ersetzt; dies spart Rechenzeit und Stackspeicher, jedoch zu Ungunsten der Codegröße

die also nicht durch andere Komponenten von außen aufgerufen werden. Im Beispiel wird das an der ursprünglichen `main()`-Methode deutlich, denn die modifizierte Applikation ist in der dargestellten Form nicht lauffähig, da auch die `main()`-Funktion verschleiert wurde. Um dies zu verhindern, gestatten die entsprechenden Software-Produkte die Definition der zu erhaltenden Namenskomponenten; für dieses Beispiel wurde `main()` ausdrücklich verschleiert.

Listing 2.3: Java Decompiler Ausgabe (ProGuard)

```
1  import java.io.PrintStream;
2
3  class a
4  {
5    public static void a(String[] paramArrayOfString)
6    {
7      new b().a("Hello World.");
8      System.out.println("\n");
9    }
10 }
11
12 import java.io.PrintStream;
13
14 class b
15 {
16   public void a(String paramString)
17   {
18     System.out.print(paramString);
19   }
20 }
```

2.1.2 Beispiel: Skype Schutzmaßnahmen

Die bekannte Kommunikationssoftware *Skype*[6] verwendet eine ganze Reihe unterschiedlicher Schutzmaßnahmen. Neben dem Schutz der übertragenen Daten durch Verschlüsselung und Infrastrukturmaßnahmen wie *Relaying*[7], verwendet Skype mindestens vier unterschiedliche Klassen von Schutzmaßnahmen für den Programmcode. Dieser Abschnitt gibt eine kurze Zusammenfassung der Ergebnisse von Biondi et al. [5], die einige der Methoden im Detail erläutern, wieder.

[6] http://www.skype.com

[7] ein Rechner tritt als Relay auf, wenn er Kommunikationsverkehr durchleitet

Binary Packing

Das binäre Lademodul der Skype-Software ist weitestgehend verschlüsselt und wird nach dem Laden durch den Loader des jeweiligen Betriebssystems an einer unverschlüsselten Stelle gestartet. Die verschlüsselten Programmteile werden zunächst durch XOR mit fest vorgegebenen Schlüsseln decodiert und im Codesegment abgelegt. Analog wird mit Daten des Datensegments verfahren. Außerdem werden initiale Programmteile nach der Ausführung gelöscht. Die Vorbereitung des Skype Speicher-Layouts ist in Abbildung 2.1 skizziert.

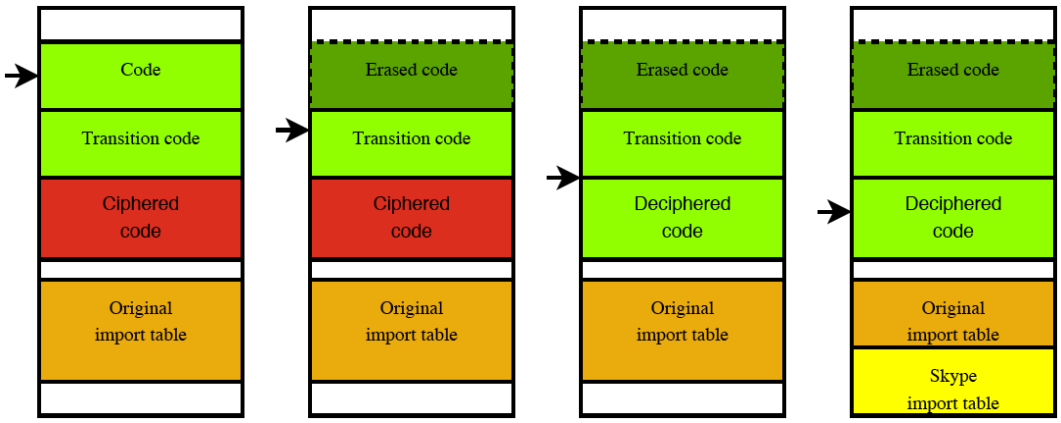


Abbildung 2.1: Skype Programmcode-Vorbereitung [5]

Neben der Entschlüsselung werden noch Teile der Import-Tabelle modifiziert. Diese Adresstabelle spielt im Zusammenhang mit dynamisch geladenen Modulen eine wichtige Rolle. Die Funktionsaufrufe in externe Bibliotheken können in den Anwendungsprogrammen nicht statisch gesetzt werden, sondern müssen durch den Loader des Betriebssystems beim Laden der dynamischen Module aufgelöst und in die Import-Tabelle eingetragen werden. Die resultierende Funktionsliste wird dann als Sprungtabelle für Anwendungsprogramme verwendet. Listing 2.4[8] zeigt die Rolle der Import-Tabelle.

Listing 2.4: Verwendung der Import-Tabelle

```
1 ...inline app code...
2 00401002 |. E8 7B0D0000     CALL 00401D82              ; \GetModuleHandleA
3 ...thunk table...
4 00401D82  $-FF25 4C204000, JMP DWORD PTR DS:[40204C];KERNEL32.GetModuleHandleA
```

[8] http://sandsprite.com/CodeStuff/Understanding_imports.html

```
5  ...memory address value of pointer...
6  40204C > FC 3D 57 7C   ;little endian pointer value
```

Das Anwendungsprogramm verwendet einen indirekten Sprung an die Stelle des Zeigers in `DS:[40204C]`. An dieser Adresse im Datensegment legt der Loader des Betriebssystems die Ladeadresse der Funktion `GetModuleHandleA` ab, nachdem er die Bibliothek `KERNEL32.DLL` (in diesem Falle unter Windows) geladen und gemappt, d. h. den Funktionsnamen in der Bibliothek tatsächliche Ladeadressen zugeordnet hat. Das Beispiel verwendet noch eine weitere Indirektion, die *Thunk Table*, eine Anwendung könnte aber auch direkt die Import-Tabelle adressieren: `CALL DWORD PTR DS:[40204C]`.

Durch die Modifikation der Import-Tabelle und das Ersetzen einzelner Funktionsadressen durch Indirektionen in den Skype-Programmcode, der die Funktionen dann mittelbar aufruft, soll die Verwendung bekannter Funktionen in betriebssystemnahen Bibliotheken verschleiert werden.

Code Integritätsprüfung

Um Laufzeitmodifikationen am Programmcode zu erkennen, verwendet Skype *Checksummer*. Das sind kurze Programm-Routinen, die eine Prüfsumme über einen Programmabschnitt bilden und so Veränderungen erkennbar machen sollen. In der Praxis werden aus den Prüfsummen Sprungadressen abgeleitet, sodass sich die Richtigkeit der Prüfsumme durch den weiteren Programmablauf ergibt und kein Vergleich mit einem statischen Literal erforderlich ist. Außerdem ändern durch Debugger gesetzte Breakpoints die Checksumme und führen so zu Fehlern in der Ablaufverfolgung.

Anti-Debugger Techniken

Skype verwendet unterschiedliche Anti-Debugger Techniken, um eine Ablaufverfolgung des Programmcodes zu verhindern. Zum einen versucht Skype, die Treiber bekannter Debugger wie *Softice*[9] oder *Rasta Ring 0 Debugger*[10] zu laden um deren Anwesenheit zu erkennen. Zusätzlich wirkt sich die Anwendung der bereits erwähnten Checksummer ebenfalls positiv für Skype gegen Debugger aus. Außerdem zählt Skype an etlichen Programmstellen die für das

[9] Weiterentwicklung mittlerweile eingestellt

[10] http://rr0d.droids-corp.org

betreffende Programmsegment erforderlichen Prozessorzyklen, um einen laufenden Beobachtungsprozess zu detektieren. Sollte eine Debugger-Erkennung positiv verlaufen, so werden die Prozessorregister mit Zufallswerten überschrieben und an eine zufällige Stelle einer zufälligen Speicherseite gesprungen. Dieses *Debugger-Trapping* erschwert die rückwirkende Codeanalyse, da die erforderlichen Informationen, wie die Werte des Stack Frame oder des Extended Instruction Pointer nicht mehr zur Verfügung stehen.

Address Obfuscation

Ziel der Code Obfuscation in Skype ist hauptsächlich, den Ablauf des Programmcodes einer statischen Analyse zu entziehen. Dies wird dadurch erreicht, dass zur Laufzeit dynamische Sprungadressen berechnet werden, statt diese statisch im Code zu verwenden. Listing 2.5 zeigt ein Code-Fragment, in dem das Vorgehen zu sehen ist.

Listing 2.5: Dynamische Sprungadressen

```
1           mov  eax , 9FFB40h
2           sub  eax , 7F80h
3           mov  edx , 7799C1Fh
4           mov  ecx , [ ebp-14h ]
5           call eax ; sub_9F7BC0
6           neg  eax
7           add  eax , 19C87A36h
8           mov  edx , 0CCDACEF0h
9           mov  ecx , [ ebp-14h ]
10          call eax ; eax = 009F8F70
11          ...
12  sub_9F8F70 :
13          mov  eax , [ ecx+34h ]
14          push esi
15          mov  esi , [ ecx+44h ]
16          sub  eax , 292C1156h
17          add  esi , eax
18          mov  eax , 371509EBh
19          sub  eax , edx
20          mov  [ ecx+44h ] , esi
21          xor  eax , 40F0FC15h
22          pop  esi
23          retn
```

In den Zeilen 1-9 wird durch einige arithmetische Operationen und den Aufruf einer weiteren Funktion die Sprungadresse für die Funktion `sub_9F8F70` ermittelt. Der Aufruf erfolgt dann mit einem Sprung an den gerade berechneten

Inhalt des EAX-Registers.

Execution Flow Rerouting

Eine weitere Maßnahme zum Verschleiern des Programmablaufes ist das absichtliche Hervorrufen von Fehlerzuständen (Exceptions), in denen dann eine Fehlerroutine (Exception-Handler) asynchron aufgerufen wird. Listing 2.6 zeigt ein Beispiel, in dem vor dem Auslösen einer Exception in einer Subroutine Vorbereitungen getroffen werden, um nach dem Abarbeiten der Fehlerbehandlungsroutine mit dem normalen Programmablauf fortfahren zu können.

Listing 2.6: Falsche Exceptions

```
1         lea   edx , [ esp+4+var4 ]
2         add   eax , 3D4D101h
3         push  offsetarea
4         push  edx
5         mov   [ esp+0Ch+var4 ] , eax
6         call  RaiseException
7         rol   eax , 17h
8         xor   eax , 350CA27h
9         pop   ecx
```

Ein ähnliches Prinzip verwendet ein Verfahren, bei dem im Programmablauf keine Sprungbefehle verwendet werden und somit die sprungauslösenden Opcodes im Programmcode mit höherem Aufwand identifiziert werden müssen. Listing 2.7 zeigt ein kurzes Beispiel.

Listing 2.7: Sprung durch Rücksprung

```
1         mov   eax, return1
2         push  eax
3         mov   eax, return2
4         push  eax
5         ret ; jump to return2
6 return1:
7         ...
8 return2:
9         ...some code...
10        ret ; back to return1
```

Zunächst werden die Adressen der Sprungziele in umgekehrter Reihenfolge auf dem Stack abgelegt, um dann mit einer Return-Anweisung an die jeweils zuletzt auf dem Stack abgelegte Adresse *zurückzukehren*. Dieses Verfahren wird auch im Rahmen des sog. *Return Oriented Programming* [17] benutzt, einem

im Bezug auf den Programmcode nicht-invasiven Angriff auf laufende Prozesse, bei dem durch Overflows Rücksprungadressen auf den Stack gebracht werden, die den Programmablauf beeinflussen.

Fazit: Code Obfuscation

Obwohl die aufgezeigten Schutzmaßnahmen (insbesondere diejenigen beim Fallbeispiel *Skype* aufgezeigten) aus praktischer Sicht recht erfolgreich eine Ablaufverfolgung durch jedermann verhindern können, so ist es jedoch theoretisch möglich, ausgehend von dem unverschlüsselten Initialisierungsprogrammcode den gesamten Ablauf und Code der geschützten Software zu rekonstruieren. Dies erfordert basierend auf den Erfahrungen in [5] sicherlich einen erheblichen (von Fall zu Fall nur schwer abschätzbaren) Aufwand, dessen Höhe jedoch im Sinne der Komplexitätstheorie nicht vergleichbar mit den einschlägigen *schweren* mathematischen Problemen ist, da es sich zur Laufzeit ausschließlich um konstruktive und nicht kryptografische Maßnahmen handelt. Im Gegensatz zu den hier umrissenen Methoden werden im Hauptteil dieser Arbeit Verfahren erörtert, deren Sicherheit auf die gängigen, in der theoretischen Informatik untersuchten Probleme reduziert werden kann.

2.2 Garbled Circuits

Bereits 1986 stellte Andrew C. Yao ein Verfahren für die verschlüsselte Ausführung von booleschen Schaltkreisen vor. Den *Garbled Circuits* liegt ein Prinzip zugrunde, bei dem sowohl verschlüsselt als auch durch Vertauschung verschleiert wird. Zentrales Element ist dabei die Verbindung zwischen zwei booleschen Gattern, der gedachte Draht. Im physikalischen Modell kann dieser einen von zwei Zuständen einnehmen, 0 oder 1. In Yaos Modell wird den beiden Zuständen z_0 und z_1 je eine Zufallszahl als Ersatz für den physikalischen Schaltwert zugeordnet, sodass $z_0 \neq z_1$. Der Wertebereich der Zahl ist dabei grundsätzlich unerheblich. Für ein Gatter mit zwei Eingangsleitungen a und b werden die vier Zustände $\{a_{z_0}, a_{z_1}, b_{z_0}, b_{z_1}\}$ notiert. In der Schalttabelle des Gatters werden die Zufallswerte der Ausgangsleitung gemäß der zu bildenden Schaltfunktion hinterlegt. Seien nun $a \leftarrow \{a_{z_0}, a_{z_1}\}$ und $b \leftarrow \{b_{z_0}, b_{z_1}\}$ die Zufallswerte der Eingangsleitungen und $c \leftarrow \{c_{z_0}, c_{z_1}\}$ die Zufallswerte der Ausgangsleitung

eines UND-Gatters. Die Schalttabelle des Gatters ist in Tabelle 2.1 dargestellt.

$\wedge$	a_{z_0}	a_{z_1}
b_{z_0}	c_{z_0}	c_{z_0}
b_{z_1}	c_{z_0}	c_{z_1}

Tabelle 2.1: Yao Schalttabelle UND (ersetzen)

Die alleinige Ersetzung der ursprünglichen Schaltwerte bringt noch keinen Sicherheitseffekt, da die Topologie der Tabelle die Schaltfunktion verrät und damit auch einen Rückschluss auf die Schaltwerte zulässt. Der zweite Schritt erfordert ein symmetrisches Verschlüsselungsschema mit den Funktionen $c \leftarrow E(p,k)$ zur Verschlüsselung und $p \leftarrow D(c,k)$ zur Entschlüsselung unter dem Schlüssel k. Die Verschlüsselung der Schalttabelle wird nach dem in Tabelle 2.2 gezeigten Schema durchgeführt, wobei die Ersatzwerte der Eingangssignale als Zeilen- und Spaltenschlüssel für die Ausgangsschaltwerte verwendet werden.

$\wedge$	a_{z_0}	a_{z_1}
b_{z_0}	$c_0 \leftarrow E(E(\text{'YAO'}\|\|c_{z_0}, a_{z_0}), b_{z_0})$	$c_1 \leftarrow E(E(\text{'YAO'}\|\|c_{z_0}, a_{z_1}), b_{z_0})$
b_{z_1}	$c_2 \leftarrow E(E(\text{'YAO'}\|\|c_{z_0}, a_{z_0}), b_{z_1})$	$c_3 \leftarrow E(E(\text{'YAO'}\|\|c_{z_1}, a_{z_1}), b_{z_1})$

Tabelle 2.2: Yao Schalttabelle UND (verschlüsseln)

In diesem Beispiel wird jeder Ausgangsschaltwert zunächst mit seinem Spaltenschlüssel und danach mit dem Zeilenschlüssel verschlüsselt. Die Reihenfolge der Verschlüsselung ist im Prinzip unerheblich, es muss jedoch in umgekehrter Reihenfolge der Verschlüsselung entschlüsselt werden. Außerdem wird jedem Ausgangsschaltwert eine Markierung vorangestellt, damit während der Auswertung korrekte Werte erkannt werden können, wie gleich gezeigt wird. Als letzter Schritt nach der Verschlüsselung und vor der Auswertung wird die verschlüsselte Tabelle noch permutiert, wobei die Positionen der Chiffretexte vertauscht werden. Dies lässt sich ohne Verlust der Eindeutigkeit der Schalttabelle durchführen, weil der ursprüngliche Schaltwert durch die Verschlüsselung mit den entsprechenden Spalten- und Zeilenschlüsseln im Chiffrat codiert wird. Eine mögliche Permutation ist in Tabelle 2.3 abgebildet.

Soll eine nicht vertrauenswürdige Partei nun das Gatter auswerten, werden ihr die beiden Eingabewerte, beispielsweise a_{z_0} und b_{z_1} sowie die verschlüssel-

$\mathfrak{c}_0 \leftarrow c_1$	$\mathfrak{c}_1 \leftarrow c_3$
$\mathfrak{c}_2 \leftarrow c_2$	$\mathfrak{c}_3 \leftarrow c_0$

Tabelle 2.3: Yao Schalttabelle UND (permutieren)

te Schalttabelle übergeben. Diese wird nun mit den zur Verfügung gestellten Schlüsseln (einem Spalten- und einem Zeilenschlüssel) entschlüsselt. Das Ergebnis ist in Tabelle 2.4 dargestellt.

$d_0 \leftarrow D(D(\mathfrak{c}_0, a_{z_0}), b_{z_1})$	$d_1 \leftarrow D(D(\mathfrak{c}_1, a_{z_0}), b_{z_1})$
$d_2 \leftarrow D(D(\mathfrak{c}_2, a_{z_0}), b_{z_1})$	$d_3 \leftarrow D(D(\mathfrak{c}_3, a_{z_0}), b_{z_1})$

Tabelle 2.4: Yao Schalttabelle UND (entschlüsseln)

Nach der Entschlüsselung der vier Schaltwerte wird nur einer die Markierung ('YAO') tragen, sodass dies der Ausgangswert des Gatters für die beiden Eingabewerte sein muss. Die drei übrigen Werte lassen sich nicht entschlüsseln, zumindest ist das Ergebnis der Entschlüsselung für den Auswertenden unbrauchbar. Die Topologie der Tabelle lässt sich also verbergen, obwohl einer der Werte entschlüsselt werden konnte. Um nach diesem Prinzip einen Schaltkreis mit vielen Gattern zu konstruieren, werden die Ausgabewerte eines Gatters als Eingabewerte eines Eingangs des Folgegatters verwendet.

Oblivious Transfer

Das Grundprinzip von Yaos Circuits wurde bis hierher als reines Delegationsszenario dargestellt, d. h. die Partei A generiert den verschlüsselten Schaltkreis für eine Funktion $\mathcal{F}$ und überlässt diesen zusammen mit den initialen Eingabewerten $\{s_0, ..., s_n\}$ der zweiten Partei B, welche den Schaltkreis als $c \leftarrow \mathcal{F}(s_0, ..., s_n)$ auswertet. Das Konzept von Yao sieht auch den Fall vor, in dem die Partei B (bzw. sogar noch weitere Parteien) ebenfalls eine Eingabe $\{t_0, ..., t_m\}$ in den Schaltkreis eingeben darf. Dieses Prinzip ist in der Literatur als *Multiparty-Computation* bekannt, wobei die Eingaben der beiden (oder mehr) Parteien untereinander und gegenseitig geheim gehalten werden sollen. Da es sich bei diesem Schema aber im Prinzip um eines mit geheimem Schlüssel handelt, stellt sich die Frage, wie die Eingabe von B der zu berechnenden Funktion $c \leftarrow \mathcal{F}(s_0, ..., s_n, t_0, .., t_n)$ mit dem geheimen Schlüssel von A verschlüsselt

werden kann, ohne A die Klartexte zu übertragen.

Dieses Problem wird mit Hilfe des *Oblivious Transfer* (etwa: *unbewusste Übertragung*) gelöst. Hierzu hat Michael Rabin 1981 eine Lösung vorgelegt [51], die Instanzen mit unterschiedlichen mathematischen Ansätzen zulässt. Die Übertragung eines von zwei Argumenten wird dabei oft als $OT_2^1(\mathcal{S}, \mathcal{E}, M_0, M_1)$ notiert und von Even, Goldreich und Lempel mit Hilfe des eines beliebigen asymmetrischen Kryptosystems $\mathcal{K}$ mit zwei Nachrichten M_0, M_1 im Nachrichtenraum $\mathcal{M}$ zwischen einem Sender $\mathcal{S}$ und einem Empfänger $\mathcal{E}$ wie folgt implementiert [23]:

1. $\mathcal{S}$ generiert eine Instanz von $\mathcal{K}$ (ein Schlüsselpaar mit öffentlichem P und geheimem S) und wählt zwei zufällige Nachrichten $m_0, m_1 \in \mathcal{M}$

2. $\mathcal{S}$ überträgt $\{P, m_0, m_1\}$ zu $\mathcal{E}$

3. $\mathcal{E}$ besitzt sein Auswahlargument $r \in \{0, 1\}$ und wählt ein zufälliges $k \in \mathcal{M}$

4. $\mathcal{E}$ überträgt $q \leftarrow Enc(k, P) \boxplus m_r$ an $\mathcal{S}$

5. $\mathcal{S}$ berechnet $k'_i \leftarrow Dec(q \boxminus m_i, S)$ für $0 \leq i \leq 1$ und wählt ein zufälliges $s \in \{0, 1\}$

6. $\mathcal{S}$ überträgt $\{M_0 \boxplus k'_s, M_1 \boxplus k'_{s \oplus 1}, s\}$ an $\mathcal{E}$

7. $\mathcal{E}$ berechnet schließlich $M \leftarrow M_r \boxminus k$

Die Operatoren $\boxplus$ und $\boxminus$ bezeichnen dabei Permutationen über $\mathcal{M}$ mit $\forall x, y \in \mathcal{M} : (x \boxplus y) \boxminus y = x$. Der Operator $\oplus$ bezeichnet die Addition modulo 2. Am Ende des Protokolls kann der Empfänger aufgrund seiner Kenntnis von r und k eine der beiden übertragenen Nachrichten korrekt entschlüsseln. Die vorliegende Implementierung erfordert mehrere Interaktionsrunden zwischen Sender und Empfänger, um die zur Unkenntlichmachung der nicht ausgewählten Nachricht erforderlichen Argumente auszutauschen. In Abschnitt 3.6 wird ein Protokoll vorgestellt, das das Problem des Oblivious Transfers auf Basis homomorpher Kryptografie mit einer einzigen Interaktionsrunde löst.

Fazit: Garbled Circuits

Der erste Ansatz mit mathematischem Nachweis, der einer tatsächlichen Verschlüsselung von Funktionen dient, wurde 1986 von Yao vorgestellt. Nach seinem Konzept der *Garbled Circuits* [61] werden Algorithmen in ihre Abbildungen als boolesche Single-Pass-Schaltkreise überführt. Bei den entstehenden Schaltnetzen werden dann die Schalttabellen der Gatterelemente verschlüsselt und die Positionen in den Schalttabellen vertauscht, um die jeweilige boolesche Funktion zu verschleiern. Dies gewährleistet, dass der Algorithmus, der aus den booleschen Funktionen zusammengesetzt wird, selbst von der ausführenden Instanz nicht eingesehen werden kann. Auch sind die in den Funktionen enthaltenen Daten nicht extrahierbar. Obwohl diese Methode mit der Implementierung von Malkhi [44] durchaus prototypisch einsetzbar ist, besteht im Kernkonzept ein großer Nachteil: Durch die Veränderung der Schalttabellen wird eine Abhängigkeit zwischen den Schaltelementen erzeugt, sodass diese nur noch in der vorgesehenen Reihenfolge schaltbar sind. Die Folge ist, dass sich nur eintaktige Schaltkreise erzeugen lassen, also solche, die sich in einem einzigen, linearen Durchlauf abarbeiten lassen. Um dies zu erreichen, müssen alle Kontrollstrukturen vor der Ausführung linear aufgelöst werden. Die Anzahl von Schleifendurchläufen muss beispielsweise bereits vorher bekannt sein, damit der Schleifenkörper *entfaltet*, also entsprechend der Wiederholungszahl vervielfacht werden kann. Weiterhin muss der Schaltkreis unter Einbezug der Eingabedaten generiert werden und kann somit nur für genau diesen Parametersatz verwendet werden. Der Grund hierfür ist, dass wenn ein zweiter Parametersatz auf dem selben Schaltkreis berechnet werden würde, die ausführende Instanz die Schalttabellen rekonstruieren und somit sowohl Algorithmus als auch Daten rekonstruieren könnte. Aus demselben Grund besteht auch keine Möglichkeit, veränderbare Speicherinhalte oder Register zur temporären Ablage zu modellieren. Dies führt dazu, dass sich mit diesem Konzept nur begrenzte Algorithmen abbilden lassen und diese zudem nur für einen speziellen Eingabedatensatz genutzt werden können. Die Malkhi-Implementierung widmet sich vorwiegend der effizienten Konstruktionsphase der Schaltkreise und dem effizienten Austausch von verschlüsselten Eingabedaten. Das Problem der Linearisierung des Programmablaufs im Chiffrat wird nicht gelöst.

2.3 Sichere Hardware

Neben softwaregestützten Mechanismen zum Systemschutz werden auch Hardwaremittel eingesetzt, wobei meist der bauliche Schutz, d. h. die Verhinderung der Ausspähung durch Einschließen oder Eingießen in bestenfalls unzerstörbare Komponenten im Vordergrund steht. Analog zum Ansatz bei der Kryptografie, bei der die Annahme gemacht wird, dass das Berechnen eines Schlüssels ab einem gewissen zeitlichen Aufwand nicht mehr lohnt, wird bei sicherer Hardware angenommen, dass ebenfalls der Kostenanstieg zum Eindringen in die Hardwarestrukturen den Nutzen übersteigt. Diese Annahme gilt zwar grundsätzlich für die breite Masse an Angriffen, hindert Hacker in speziellen Fällen jedoch nicht an der erfolgreichen Durchführung, wie an einem Beispiel gezeigt wird.

2.3.1 Trusted Computing

Das Konzept *Trusted Platform Computing* ist grundsätzlich kein Mittel zum verschlüsselten Rechnen an sich, es soll jedoch einen logischen Schritt vor der Ausführung von Software sicherstellen, dass nur vertrauenswürdige Software geladen wird und damit eine sichere Basis hergestellt wird. TPC sieht somit die Plattform, auf der gerechnet wird, als prinzipiell nicht vertrauenswürdig an. Es müssen also neben der eigentlichen Ausführung von Betriebssystem und Anwenderprogrammen spezielle Abläufe die *Integrität* der Plattform sicherstellen. Integrität bedeutet in diesem Zusammenhang, dass sich der entsprechende Computer seit seinem Reset in einem definierten Zustand befindet und dieser Zustand nachvollziehbar ist und einem Anwender glaubhaft gemacht werden kann. Die Integrität kann in diesem Fall auf eine Folge von SHA-Hashes reduziert werden, wobei der initiale Status der Hashwert des Speicherabbildes eines vertrauenswürdigen ROMs sein muss. Vertrauenswürdig heißt hier, dass das Speicherimage des BIOS eines Rechners, das von einer vertrauenswürdigen Partei im Sinne einer Public Key Infrastruktur signiert wurde, für echt und unverändert befunden wird. Dieser Vorgang wird technisch von einem signierten Stück Code, dem *Core Root of Trust for Measurement* vollzogen. Zu dem entstandenen Hashwert $\mathcal{H}_0$ (Ausgangszustand) werden dann während des *trusted Boot*-Vorgangs die Hashes aller geladenen Programme und Module hinzugefügt, sodass der kumulative Hash $\mathcal{H}_i = SHA(\mathcal{H}_{i-1}||SHA(module))$

zu jedem Zeitpunkt einen definierten Zustand hat. Werden andere als die für den definierten Zielzustand erforderlichen Module während des Boot-Vorgangs geladen, unterscheidet sich der endgültige Hashwert vom erwarteten, und so kann diese Abweichung leicht festgestellt werden. Entscheidend ist, dass der Zustand in einem geschützten Hardware-Baustein gespeichert wird und dass eine unrechtmäßige Modifikation des Startvorgangs durch das vorherige Laden des eingeschleusten Moduls, welches diese Modifikation vornimmt, sichtbar wird. Die Hash-Sequenz einer Linux-Implementierung des Trusted Boot ist in Abbildung 2.2 dargestellt.

	Measurement Value (fingerprint == SHA1)	Measurement Hook	File Name	
#000	9797EDF8D0EED36B1CF92547816051C8AF4E45EE	ima-init	boot-aggregate	Aggregate
#001	F7A0BF5A67CE98BC06316F77CA1F404A2D447534	mmap-file	init	Executable
#002	38C5D31E5DAD3F1B012FDD35B4E011E783CE6FD8	mmap-file	ld-2.3.2.so	Library
#003	42F7960321992201671388B8AAFC9E37F6936B226	mmap-file	libc-2.3.2.so	Library
#004	A4DC5EDF06698646CD76916F16E95C37E55DC12B	mmap-file	bash	Executable
#005	F4F6CB0ACC2F1BEE13D60330011DF926D24E5688	mmap-file	libtermcap.so.2.0.8	Library
#006	AE1BC1746AFD2AC1ECD1D9EEEAEBD125A6A9E88D	mmap-file	libdl-2.3.2.so	Library
#007	CFBC7EC3302145AB78A307C0D41DBB9A4251377B	mmap-file	libnss_files-2.3.2.so	Library
#008	805572455CF5BF50A7EE42E3CC6B0EDA65AF17A4	mmap-file	initlog	Executable
#009	C95CBC5625719649103E0D1C3595967474842F7B	mmap-file	hostname	Executable
#010	0CAA342424F420FF29B7FB2FCF278F973600681B	mmap-file	mount	Executable
#011	5E45D898530F31BADEF5E247EBCF4AB57A795366	bash-source	functions	Bash Source
#012	A253AF3AB981711A13AE45D6B46462386E628076	mmap-file	consoletype	Executable
#013	2E37B839BC4EC1B6BE1BDF5BACD1E7B56567D8D9	bash-source	i18n	Bash Source
#014	C9D1B3E2CD0995E16AE6DD98B388FD873324740D	bash-source	init	Bash Source
#015	590F75EE97E0FC560F07FCB07A8646FADEC88C2A	mmap-file	uname	Executable
#016	5E851EFA4601B3AFCA9EAE75ED53688606630BFA	mmap-file	grep	Executable
#017	32798F58C4F1B4CD017B09BCAAF2A22D345E7E4F	mmap-file	sed	Executable
#018	CE516DE1DF0CD230F4A1D34EFC89491CAF3D50E4	mmap-file	libpcre.so.0.0.1	Library
#019	22EAF1B6009B23150367F465694AC63314866558	bash-script	setsysfont	Bash Comma
#020	8B15F3556E892176B03D775E590F8ADF9DA727C5	bash-script	unicode_start	Bash Comma
#021	A4C5F9D457DA16E47768423A68F135259F7180D7	mmap-file	kbd_mode	Executable
#022	497ED7F80C33AF25307DFC80970571C51006CE6A	mmap-file	dumpkeys	Executable
#023	04A0599405EBD306CEF2447679C8F4B5159A55C7	mmap-file	loadkeys	Executable
#024	AE327AD27D02BF2DE96557A1B4053D02129B1394	mmap-file	setfont	Executable
#025	7334B75FDF47213FF94708D2862978D0FF36D682	mmap-file	gzip	Executable
#026	93D65AB85CF5EE1ACD9E6BE5057D622D80AB5E10	mmap-file	dmesg	Executable
#027	B6E90C3A25B69C3B1D3B643DB7D9504FBC36C1D1	mmap-file	minilogd	Executable

Abbildung 2.2: Trusted Boot Hash-Sequenz einer Linux-Implementierung

Dazu bedient sich TPC des erwähnten Hardware-Bausteins, dem Trusted Platform Module (TPM). Es handelt sich dabei um einen über den LPC[11]-Bus an den Chipsatz des Rechners angeschlossenen weiteren Chip, auf dem kryptografische Schlüssel generiert, signiert und gespeichert werden können. Jedes TPM besitzt dabei eine eigene *Identität*, weil es einen eindeutigen Wurzelschlüssel besitzt. Das Schlüsselkonzept des TPM ist derart ausgelegt, dass der Identitätsschlüssel das TPM niemals verlässt und die abgeleiteten Schlüssel immer mit einem TPM-generierten Schlüssel geschützt sind. Der Besitz eines korrekt signierten TPM-Schlüssels kommt demnach einer Identität in diesem

[11] low pin count, ein serieller Bus

Konzept gleich.

Der Umstand der mutmaßlichen Identifizierbarkeit eines TPM - und damit eines Computersystems - wurde dem Konzept übrigens in der Öffentlichkeit zum Verhängnis, da dadurch Verfahren für digitales Rechtemanagement (DRM) prinzipiell die konsumierten Medieninhalte an diese Identität knüpfen könnten. Dies ist zum größten Teil der Informationspolitik der damaligen Trusted Computing Platform Alliance (TCPA) zuzuschreiben, die sich bald darauf in Trusted Computing Group (TCG) umbenannte und der es nicht gelang, den passiven Charakter des TPM einer breiteren Öffentlichkeit darzustellen.

Zentrales Steuerungselement der TCG-Spezifikation[12] ist der Software Stack TSS (TCG Software Stack). Hier werden in der gängigen Form separater Funktionsebenen (Layer) die unterschiedlichen Steuerungs- und Anwendungsbereiche des TPM und der sicheren Umgebung implementiert und strukturiert. Der Stack besteht aus den folgenden Ebenen (beginnend bei der Hardwareebene):

- TPM Device Driver (Gerätetreiber)
- TSS Device Driver Library (Standard Hardware Interface)
- TSS Core Services (Systemservice)
- TSS Service Provider (Standard API für Anwendungen)

Das eingesetzte TPM ist technisch gesehen ein mit speziellen Funktionen ausgestatteter Mikroprozessor. Die Kernfunktionen entsprechen so oder so ähnlich dem Beispiel Infineon SLB 9635:

- 16-Bit Mikroprozessor in 0.22 μm CMOS Technologie
- 24 Platform Configuration Registers (PCRs)
- 10 Schlüsselspeicher (Slots)
- 1.5 kB nichtflüchtiger Speicher
- EEPROM für Firmware und Schlüssel / Daten
- Krypto-Engine (2048-Bit RSA)

[12] hier ist stets die Spezifikation Rev. 1.4 von 2007 [1] gemeint

- Hashing-Engine (SHA-1)
- Taktzähler und physische Einbruchserkennung
- Low Pin Count (LPC) Bus Interface

Die Plattformintegration eines TPM ist durch die Anbindungsart per seriellem Bus minimal-invasiv und beschränkt die Möglichkeit des aktiven Eingriffs des Moduls auf Abläufe und Prozesse des beherbergenden Computersystems von vornherein. Abbildung 2.3 zeigt ein beispielhaftes Blockschaltbild der Integration eines TPM in die Intel Atom-Plattform.

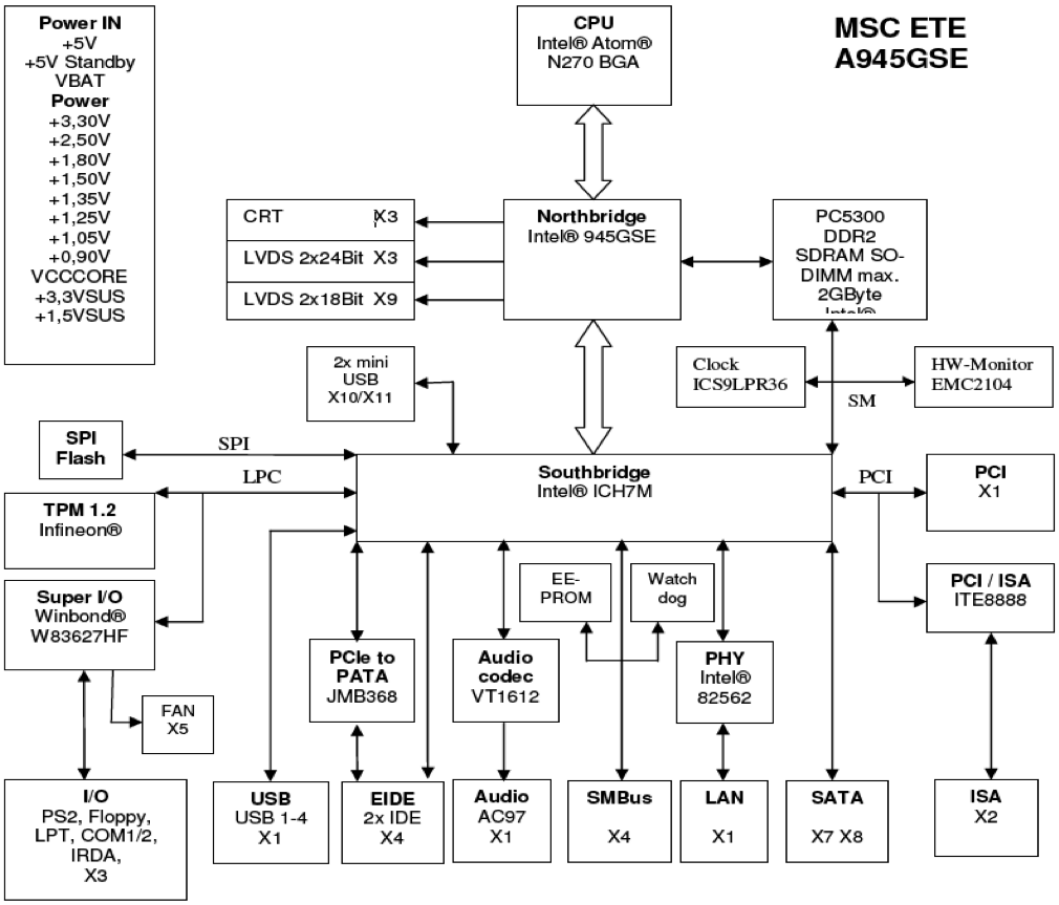


Abbildung 2.3: TPM-Integration in die Intel Atom-Plattform (MSC GmbH, ETE-A945GSE User's Manual)

Ein wichtiger Bestandteil des TCG-Konzeptes ist Vertrauen, welches auf unterschiedlichen Ebenen erzeugt werden soll. Grundlegend für dieses Vertrauen

ist jeweils ein Schlüssel oder ein signiertes Maschinenprogramm, dessen Sicherheit und Unveränderlichkeit durch physikalischen Schutz plausibel gemacht werden soll. Die unterschiedlichen Vertrauensdimensionen im TCG-Konzept sind die folgenden:

- RTR Reporting (Signieren von Statusinformationen)
- RTS Storage (Sicherung von außerhalb gespeicherten Daten)
- RTM Measurement (Feststellen des Zustandes der Plattform)

Das Schlüsselkonzept sieht mehrere Schlüssel für unterschiedliche Aufgaben vor. Zentrales und identifizierendes Element ist der *Endorsement Key*, ein 2048-Bit RSA Schlüssel, der je TPM als Unikat generiert wird. Mit diesem übernimmt der Eigentümer der Plattform die Kontrolle, und er dient im weiteren Betrieb zur Generierung der Schlüssel zur Attestierung und zum Signieren. Der im Rahmen der Plattformübernahme generierte *Storage Root Key* ist der physikalische Wurzelschlüssel einer auf den TPM-Chip bezogenen Schlüsselhierarchie und verschlüsselt direkt oder indirekt alle auf dem TPM generierten weiteren Schlüssel, wenn diese außerhalb des TPM gespeichert werden sollen (*Wrapping*, etwa: Umhüllung). Das TPM bietet nur eine begrenzte Anzahl von Speicherplätzen für Schlüssel zum unmittelbaren Gebrauch während der Ausführung einer Anwendung, und so ist eine Speicherung außerhalb des TPM je nach Anwendungsfall erforderlich, jedoch nur bei sog. migrierbaren Schlüsseln möglich. Weitere Schlüssel sind beispielsweise die *Attestation Identity Keys*, mit deren Hilfe der Plattformzustand anonym (d. h. ohne direkten Einsatz des identitätsstiftenden Endorsement Keys) signiert werden kann.

Der Systemzustand (genauer gesagt: der Hashwert eines Systemzustandes) kann zur Verschlüsselung von Daten verwendet werden. Die TCG-Spezifikation spricht dabei von *Sealing* (Versiegelung). Der besondere Nutzen an diesem Verfahren ist die Tatsache, dass die so versiegelten Daten nur noch in demselben Systemzustand wieder entschlüsselt werden können. Die Verwendbarkeit der Daten wird also an einen speziellen Zustand gebunden und ist in jedem anderen Zustand ausgeschlossen. Werden Daten mit dem öffentlichen Schlüssel eines bestimmten TPM verschlüsselt, so werden diese Daten an die Plattform mit der entsprechenden Identität gebunden (*Binding*). Dieser Modus ist insbesondere für das Digitale Rechtemanagement (DRM) interessant. Hier werden

die lizensierten (Medien-) Daten an die Plattform des Lizenznehmers geknüpft, sodass sie nur auf der speziellen Plattform lauffähig sind. Dies erfordert jedoch noch weitere kollaborative Anstrengungen der umgebenden Softwarekomponenten (Betriebssystem, Media-Player, etc.).

Auf der Hackerkonferenz *Blackhat DC 2010* wurde von Christopher Tarnovsky (Flylogic Engineering) ein Verfahren vorgestellt, das die Schlüssel eines TPM ermitteln kann. Obwohl dies ein physikalisch-chemotechnischer Vorgang im Wert von etwa 200000 Dollar ist, zeigt er doch, dass selbst sog. *tamper-proof* (gegen Eindringen geschützte) Hardware einem entsprechend hohen Aufwand nicht standhalten kann.

Die folgenden drei Abschnitte behandeln die Interpretationen und Implementierungen der TCG-Spezifikation aus einer hardwaregetriebenen (Intel) und zwei aus einer softwaregetriebenen (Microsoft und OpenSource) Perspektive.

Intel Trusted Execution Technology (LaGrande)

Die ehemals unter der Marketingbezeichnung*LaGrande* eingeführte Trusted Execution Technology (TXT) setzt auf dem Konzept und der technischen Realisierung des Trusted Computing auf. Dabei fokussiert die Technologie softwarebasierte Angriffe auf ein Computersystem [34]. Das TPC-Kernkonzept verknüpft Intel auf seinen vPro Plattformen mit weiteren Technologien wie VT-d ('Virtualization Technology for directed I/O'), um folgende Ziele zu erreichen:

- Der *verifizierte Systemstart* soll als Implementierung des Trusted Boot einen definierten Systemzustand herstellen, auf dem alle weiteren Sicherheitsmechanismen aufsetzen

- Die *Attestierung* des Systemzustandes soll lokale Benutzer und andere Systeme von der Vertrauenswürdigkeit des Computersystems überzeugen

- Die *geschützte Ausführung* von Programmen soll die Verarbeitung sensibler Daten abgeschottet von anderer Software ermöglichen

- *Versiegelter Speicher* dient zur geschützten Ablage von kryptografischen Schlüsseln oder anderer Daten

Microsoft NGSCB (Palladium)

Der Ansatz von Microsoft zum Trusted Computing ist die Next-Generation Secure Computing Base (ehemals *Palladium*). Auch hierbei handelt es sich um eine TPC-Implementierungsvariante, die sich jedoch in Ergänzung zum hardwarelastigen Vorgehen von Intel auf die Betriebssystemebene konzentriert. Dennoch verwendet auch Microsoft eine Sicherheitshardware als Grundlage, welche abstrakt als SSC (Security Support Component) bezeichnet wird. Gemeint ist damit das TPM. Ziel von NGSCB ist die parallele Abarbeitung von *herkömmlichen* und *sicheren* Operationen auf derselben Maschine. Dazu soll neben den Systemebenen der klassischen Windows-Architektur ('Left-Hand-Side') ein zweiter abgesicherter Kern ('Right-Hand-Side') namens *Nexus* etabliert werden, der die Anforderungen an ein sicheres System nach Ansicht von Microsoft erfüllt. Diese gleichen natürlich im Wesentlichen den Zielen der Intel-Implementierung:

- *Strenge Prozessisolation* zur geschützten Ausführung von gegeneinander abgeschotteten Programmen
- *Geschützter Speicher* ist nur für die damit verknüpften Prozesse zugreifbar
- *Sichere Datenwege (I/O)* sollen Ein- und Ausgabe zu vertrauenswürdiger Peripherie schützen
- *Attestierung* stellt den definitionsgemäßen Zustand eines Systems sicher

Die Notwendigkeit zur gleichzeitigen Unterstützung von zwei Kernen wird mit den befürchteten Kompatibilitätsproblemen begründet, weil sämtliche Anwendungssoftware auf eine neue API umgestellt werden müsste. In [2] werden die Prinzipien und die Umsetzung in eine Programmierschnittstelle für NGSCB mit Hilfe einer formalen Logik untersucht. Die Technologie sollte in Windows Longhorn (Vista) zum ersten Mal zum Einsatz kommen, wobei zunächst nur die Implementierung des Trusted Boot geplant war. Der aktuelle Stand des Projektes ist unklar, zumindest ist die Informationsversorgung seitens Microsoft unbefriedigend[13]. In Windows Vista wurde als einzige Funktionalität des

[13] http://www.microsoft.com/resources/ngscb/default.mspx (abgerufen am 14. Mai 2012)

gesamten Konzeptes die Festplattenverschlüsselung *BitLocker* eingeführt, die vom TPM jedoch nicht notwendigerweise Gebrauch macht.

2.3.2 EMSCB (Turaya)

Eine Open-Source Variante eines TCG-konformen Systems ist das im Rahmen des BMWi-Projektes EMSCB (European Multilaterally Secure Computing Base) entstandene Turaya. Den Nutzen des Projektes beschreibt das Konsortium auf seiner Webpräsenz wie folgt:

"Durch die auf neuesten Betriebssystem-, Trusted Computing- und Softwaretechnik-Technologien aufbauende Entwicklung bieten sich vielfältige Nutzungsmöglichkeiten von vertrauenswürdigen Plattformen wie EMSCB. Weiterhin hat EMSCB viele positive ökonomische und politische Vorteile, die im Folgenden näher erläutert werden.

Multilaterale Sicherheit: Die Plattform erlaubt die Durchsetzung lokaler (z. B. Endanwender) und externer (z. B. Content-Provider) Sicherheits- und Zugriffs-Policies. Einerseits soll die Plattform Schutz gegen böswilligen Code (z. B. Trojanische Pferde oder Viren), aber auch gegen Verletzungen der persönlichen Sicherheits-Policies zur Verfügung stellen. Andererseits schützt sie Content-Provider vor der Umgehung ihrer Lizenzvereinbarungen nachdem der Verbraucher diese bereits akzeptiert hat. Im Gegensatz zu der sehr großen Zahl vorhandener Lösungen kann die EMSCB Trusted Computing Plattform nicht durch Software-Angriffe umgangen werden. Da Trusted Computing Hardware einen angemessenen Grad an Manipulationssicherheit anbietet, ist eine Umgehung der Sicherheitsmechanismen nur durch teure Hardwareanalyse oder aufwändige Hardwaremanipulation möglich, welche von normalen Benutzern kaum durchgeführt werden kann.

Offene Architektur: Aufgrund der offenen Architektur und der geringen Komplexität der sicherheitsrelevanten Komponenten ist diese Plattform sehr zuverlässig. Die geringe Komplexität reduziert auch die Wahrscheinlichkeit von Ausfällen während der Entwicklung und des Wartungsprozesses, was umgekehrt die Vertrauenswürdigkeit der Implementierung erhöht. Zusätzlich scheint eine Auswertung entsprechend vorherrschender Sicherheitsstandards wie z. B. Common Criteria möglich. Die offene Architektur erlaubt es, notwendige Updates, Verbesserungen und Anpassungen der Plattform an einzelnen Anforde-

rungen unabhängig von spezifischen Herstellern durchzuführen.

Vermeidung eines möglichen Missbrauchs von Trusted Computing Technologie: Kritiker von Trusted Computing und vor allem Mitglieder der Open Source Gemeinde sind über die möglichen Gefahren von Trusted Computing Technologie besorgt.

Diese kann

- die Kontrolle des Endbenutzers über seinen Computer einschränken
- für Zensur missbraucht werden
- die Rechte der Endanwender verringern
- Datenschutzrechte verletzen
- für verschiedene Arten der Spionage genutzt werden.

Der so entstehende Konflikt zwischen den Interessen und den Sicherheitsanforderungen der Endanwender (Individualdatenschutz und Selbstbestimmung) und den Anbietern von Inhalten und Anwendungen, kann durch eine allseits vertrauenswürdige Computerplattform, die das Gleichgewicht der Interessen aller beteiligten Parteien garantiert, gelöst werden. Im Sinne multilateraler Sicherheit vergleicht EMSCB die Sicherheitsanforderungen der Endanwender mit den Lizenzvereinbarungen der zu installierenden Software und kann eine Installation im Falle eines Konfliktes verhindern. Außerdem erlaubt die offene Architektur dem Endanwender das Design und den Quellcode selbst zu evaluieren und so Klarheit über die Funktionen (z. B. ausschließen systemweiter Zensur) zu erlangen.

Effiziente Übertragbarkeit: Da die sicherheitskritischen Komponenten von EMSCB nur von den vom Mikrokern bereit gestellten Schnittstellen abhängig sind, kann EMSCB auch sehr einfach auf anderen Systeme, wie z. B. PDAs, Smart Phones oder embedded Systemen migriert werden. Es gibt eine große Bandbreite von Einsatzmöglichkeiten in neuen Multimedia- oder Informationssystemen, wie z. B. in der Automobilindustrie.

Versprechen für die Zukunft: Die EMSCB Architektur ist zu herkömmlichen Betriebssystemen kompatibel. Der zukünftige Einfluss und die Wichtigkeit von Trusted Computing Betriebssystemen wird durch die große Aufmerksamkeit,

mit welcher bekannte Betriebssystemmonopolisten sich dieser Technologie zuwenden, deutlich. Durch eine alternative, offene Plattform, können sicherheitskritische Anwendungen, bis zu einem bestimmten Grad, betriebssystemunabhängig und für zukünftige Anforderungen gestaltet werden."[14]

Als eines der Leitmotive wird die Offenheit des Systems angeführt. Dies deutet auf die Befürchtung hin, dass eine ausschließlich in Herstellerhände übertragene Verantwortung für die Umsetzung des TCG-Konzeptes für den Endanwender ungünstige Folgen haben könnte. Dass die *Big Player* in der Hard- und Softwareindustrie Produkte zum Nachteil ihrer Kunden entwickeln würden, ist eine Argumentation, die in der gesamten öffentlichen Diskussion über Trusted Platform Computing und digitales Rechtemanagement vorherrscht.

Fazit: Trusted Platform Computing zielt auf die Sicherung einer Plattform gegen softwarebasierte Angriffe auf deren Integrität. Die Grundannahme dabei ist, dass die Bedrohung grundsätzlich auf die Plattform gerichtet ist, d. h. dass diese vor schadhafter Software zu schützen ist. Soll auf einer entfernten Ressource ein Programm mit vertraulichen Daten verarbeitet werden, so kann zunächst der Plattformzustand über den Mechanismus der *Remote Attestation* abgefragt werden. Im Falle des positiven Befundes auf Client-Seite kann dann mit der Übertragung und Ausführung fortgefahren werden. Trotz der TCG-konformen Durchführung aller zur Verfügung stehenden Mechanismen kann der Client nicht verhindern, dass auf Ressourcen-Seite die Verarbeitung der Maschine hardwaremäßig angehalten und der Speicherinhalt analysiert wird. Zur nachweislich vertraulichen Datenverarbeitung in einem Delegationsszenario ist das Trusted Platform Computing somit nicht geeignet.

2.3.3 Secure Microcontroller

Ein hardwarebasierter Ansatz zur Sicherung der Ausführung von vertraulichen Programmen bilden die *sicheren Microcontroller*. Hier ist der Schutzmechanismus weniger die kryptografische Verschlüsselung als vielmehr die Verschleierung des Programmablaufs. Das Prinzip soll hier am Beispiel eines DS5250 von Maxim/Dallas Semiconductor vorgestellt werden. Es handelt sich bei die-

[14] http://www.emscb.de/content/pages/Nutzen.htm abgerufen am 16.05.2012

sem Typ im Kern um einen 8051-kompatiblen Microcontroller mit funktionalen Erweiterungen zur Sicherung der enthaltenen Software und Daten in zwei Richtungen. Zum einen enthält der Chip Schaltungen zur Einbruchserkennung und zur Selbstlöschung, wodurch physische Einwirkung auf das Bauteil keinen Erfolg bringt. Zum anderen bietet der Controller eine Busverschlüsselung mit einem Mechanismus zur Verschleierung des Speicherzugriffs.

Die Busverschlüsselung basiert auf einem Controller-internen geheimen Schlüssel, welcher bei der Produktion einmal auf dem Chip generiert wird und diesen nicht verlässt. Die Verschlüsselung der Datenbussignale hat zur Folge, dass auch die Speicherinhalte verschlüsselt sind. Die Verschlüsselung basiert dabei auf dem geheimen Schlüssel k und der Verschlüsselungsfunktion $c \leftarrow Enc(m, a, k)$ mit dem Klartextdatum m und der Speicheradresse a. Der Einbezug der Speicheradresse führt dazu, dass eine Klartextnachricht - an unterschiedlichen Stellen im Speicher abgelegt - verschiedene Darstellungen besitzen kann.

Zur Verschleierung des Speicherzugriffs generiert der Controller eine schlüsselabhängige bijektive Abbildung des physischen Adressraums auf ein randomisiertes Speicherlayout. Das bedeutet, dass jede logische Speicheradresse eindeutig auf eine andere physische abgebildet wird, wodurch vor allem lineare Sequenzen im Speicherzugriff aufgebrochen werden sollen. Dies soll verhindern, dass ein Angreifer durch Horchen am Bus (*Bus-Sniffer*) übertragene Daten oder Speicherzugriffsschemata (lineare Programmabläufe, Schleifen, etc.) abhören kann. Zusätzlich fügen viele Controller dieser Art *dummy reads* und *dummy writes* in das Speicherzugriffsschema ein, die dann ausgeführt werden, wenn es der Programmablauf eigentlich nicht erfordert und der Bus jeweils nicht für tatsächliche Zugriffe benötigt wird. Dies soll zusätzliche Verschleierung erzeugen.

Abbildung 2.4 zeigt das Blockschaltbild des DS5250, wobei besonders die Einheiten *Program Memory Cryptograph* und *Data Memory Cryptograph* am Bus-Ausgang des Chips für die eben beschriebenen Eigenschaften verantwortlich sind. Außerdem sorgt die Watchdog-Schaltung mit Temperatur- und Takt-Monitor dafür, dass der Chip nur innerhalb der Spezifikationen betrieben werden kann und nicht beispielsweise in einer niedrig getakteten Testumgebung.

Selbst bei sog. sicherer Hardware sind etliche Angriffsvektoren bekannt. Hier

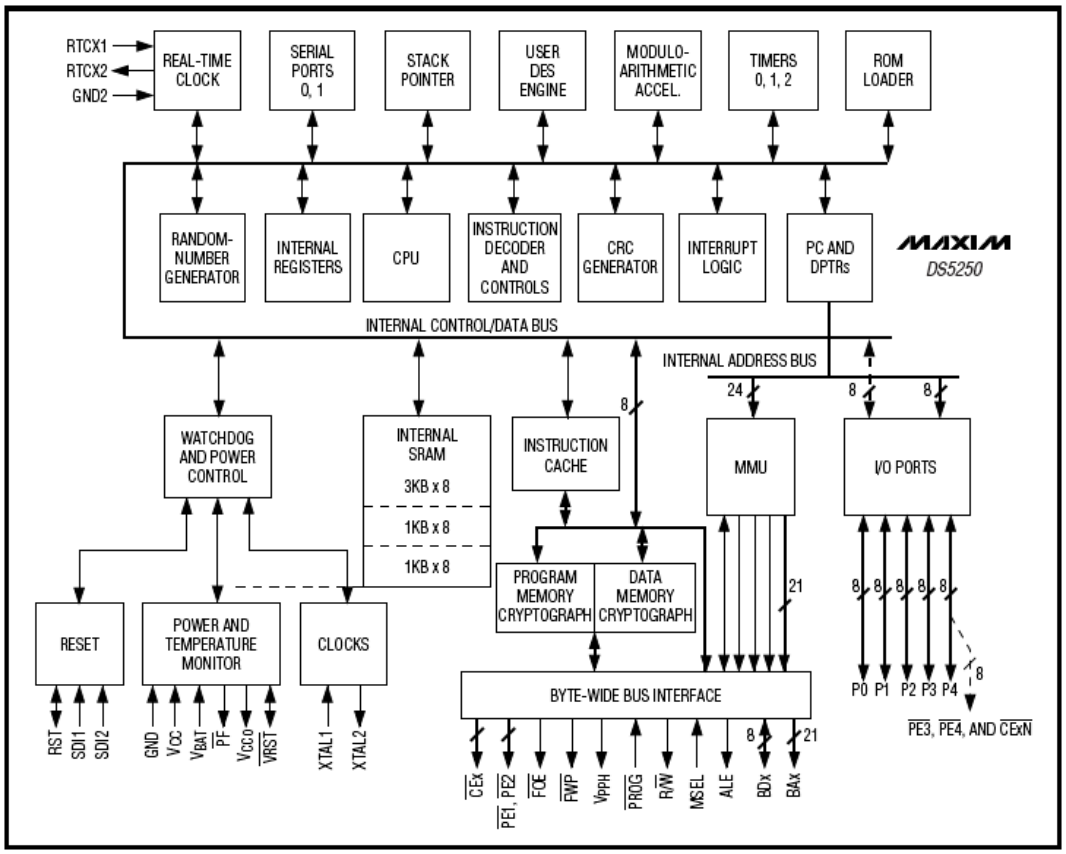


Abbildung 2.4: Blockschaltbild des DS5240 Microcontrollers (Maxim DS5250 Abridged Data Sheet Rev. 4; 11/08)

soll nur beispielhaft eine Variante erörtert werden. Es handelt sich dabei um sog. Power-Attacken. Die Grundannahme hierfür ist, dass die unterschiedlichen funktionalen Einheiten eines Mikroprozessors unterschiedlich viel Energie benötigen und somit durch eine Analyse der Stromaufnahme über die Zeit gesehen ein Energieprofil erstellt werden kann, aus dem die Aktivität der Funktionseinheiten hervorgeht. Abbildung 2.5 zeigt ein solches Profil der Stromaufnahme für eine Smartcard, die eine DES-Operation ausführt.

Klar zu erkennen sind die 16 DES-Runden, wobei bei einer höheren Sampling-Rate und einer höheren Auflösung die Details der verschlüsselten Daten sichtbar gemacht werden können. Dies liegt im Wesentlichen an einem Programmierstil, der diese Form des Angriffs nicht berücksichtigt und so z. B. Programmsprünge und arithmetische Operationen in Abhängigkeit von Zwischen-

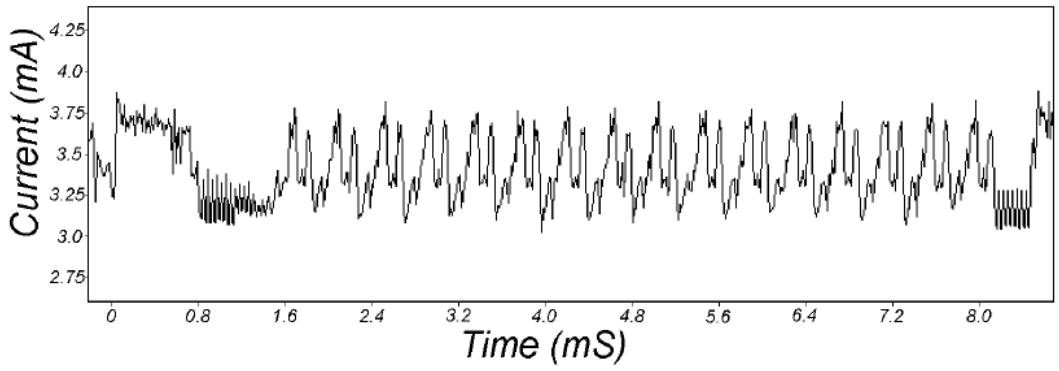


Abbildung 2.5: Energieprofil einer DES-Operation auf einer Smartcard [40]

ergebnissen oder Flags ausführt. Derzeit existieren viele Varianten der Power-Attacke, neben der SPA (Simple Power Analysis) vor allem die DPA (Differential Power Analysis), bei der die Abweichungen zu einer Referenz-Energieaufnahme ermittelt werden. Um solchen Angriffen zu begegnen, wurden Gegenmaßnahmen entwickelt, um sichtbare Unterschiede in der Stromaufnahme zu verringern. Dazu wird einer Funktionseinheit eine zweite zur Seite gestellt, die eine komplementäre Operation ausführt, sodass die Energieaufnahme beider Einheiten zusammen stets gleich ist [3]. Die technische Herausforderung ist dabei, die *energetisch komplementäre* Operation zu finden, die nicht notwendigerweise mit dem *binären Komplement* des Programmcodes und der Daten übereinstimmen muss.

Fazit: TPC & Sichere Hardware

Trusted Platform Computing kann den Schutz von Daten und Algorithmen nur indirekt wahren. Es kann lediglich sichergestellt werden, dass die Algorithmen auf einer Plattform ausgeführt werden, welcher der Nutzer vertraut. Dieses Vertrauen basiert auf dem kryptographisch signierten Nachweis des Plattformzustandes gegenüber dem Benutzer. Allerdings können die Programme und Daten nicht vor dem Betreiber der Plattform geschützt werden, denn dieser ist stets im Besitz des Wurzelschlüssels der Schlüsselhierachie seiner Plattform.

Sichere Hardware in Form von *Secure Microcontrollern* eignet sich zur Herstellung von Hardware-/Software Packages, also solchen Produkten, bei denen die Steuerung samt Software fest mit dem Produkt verbunden ist. Eine

Änderung der Software bedingt die Rücksendung des Microcontrollers zum Softwarehersteller, weil es durch das Schlüsselkonzept nicht möglich ist, die Software außerhalb des Controllers zu verschlüsseln. Die Software muss also unverschlüsselt durch den Controller gelesen und on-chip mit dem geheimen Schlüssel verschlüsselt und in dieser Form in dem Speicher des Controllers bzw. des Produktes abgelegt werden. Wenn die Software bzw. die Daten vertraulich sind, so kann dieser Vorgang nur beim Hersteller des Produktes erfolgen.

2.4 Homomorphe Kryptografie

Ein Homomorphismus ist in der Mathematik eine strukturerhaltende Abbildung zwischen zwei Mengen. Seien $\mathcal{P}$ und $\mathcal{C}$ die beiden betrachteten Mengen und f eine Funktion mit der Umkehrfunktion f', welche Transformationen zwischen Elementen der beiden Mengen vornehmen. Existiert nun eine Operation $\oplus$, sodass gilt

$$\forall x, y \in \mathcal{P} : f'(f(x) \oplus f(y)) = x \oplus y$$

so spricht man von einem Homomorphismus. Dieser kann beispielsweise additiv sein, sofern $\oplus$ einen Additionscharakter besitzt, oder multiplikativ, wenn die Operation Multiplikationscharakter besitzt. Der Charakter der Operationen kann in beiden Mengen auch unterschiedlich sein, d. h. auf Elementen der Menge $\mathcal{C}$ bewirkt eine Multiplikation eine Addition auf den entsprechend zurückgeführten Elementen der Menge $\mathcal{P}$. Existieren zwei Operationen $\oplus$ und $\otimes$, sodass gleichzeitig gilt

$$\forall x, y \in \mathcal{P} : f'(f(x) \oplus f(y)) = x \oplus y, f'(f(x) \otimes f(y)) = x \otimes y$$

handelt es sich um einen algebraischen Homomorphismus, weil er in der Lage ist, algebraische Strukturen zwischen zwei Mengen abzubilden. Der praktische Nutzen eines solchen Homomorphismus' liegt darin, dass er dazu verwendet werden kann, um Informationen aus der einen Menge zu verschlüsseln, indem sie in die andere Menge transferiert werden. Im verschlüsselten Zustand werden dann Operationen auf den Daten ausgeführt, welche sich homomorph auf

die später entschlüsselten Daten auswirken. In der Folge können so Rechenaufträge an Dritte delegiert werden, auch wenn diese nicht vertrauenswürdig sind (in Kapitel 3 wird definiert, was *vertrauenswürdig* in diesem Zusammenhang bedeutet).

Neben der Durchführung elementarer arithmetischer Operationen auf den verschlüsselten Daten kommt dem algebraischen Homomorphismus eine besondere Bedeutung zu, wenn man ihn zur Darstellung von booleschen Schaltkreisen nutzt. Dazu werden beliebige Schaltkreise auf Repräsentationen aus `XOR`- und `AND`-Gates reduziert, welche einer Addition bzw. Multiplikation modulo 2 entsprechen. Auf diese Weise lassen sich beliebige Funktionen auf verschlüsselten Daten realisieren. Die Kapitel 4 und 5 arbeiten diesen Ansatz der Funktionsabbildung mit booleschen Schaltkreisen im Detail aus. Zunächst führt Kapitel 3 weiter in die homomorphe Kryptografie ein und erläutert mehrere Schemata mit homomorphen Eigenschaften.

Die homomorphe Kryptografie bildet die Grundlage für die in dieser Arbeit vorgestellten Techniken und Verfahren. Im nächsten Kapitel wird sie daher detailliert betrachtet.

Kapitel 3

Homomorphe Kryptografie

Die homomorphe Kryptografie ist eine Richtung der Verschlüsselung, die von den etablierten Kryptosystemen in einem wesentlichen Punkt abweicht. Die klassischen Verfahren verschlüsseln hauptsächlich passive Daten, sei es mit einem geheimen Schlüssel (Secret Key Cryptography) oder mit Paaren aus öffentlichem und geheimem Schlüssel (Public Key Cryptography), wie in Abschnitt 1.1 skizziert. *Passiv* bedeutet in diesem Zusammenhang, dass die Daten durch den Vorgang der Verschlüsselung gewissermaßen eingefroren werden und für die Dauer der Verschlüsselung zwar sicher im Sinne einer Nichtlesbarkeit sind, jedoch in diesem Zustand nicht direkt weiterverarbeitet werden können. Der Verschlüsselungszustand beschränkt sich daher systembedingt auf die Datenablage oder denTransport. Die homomorphe Kryptografie dagegen gestattet es, Daten auch im verschlüsselten Zustand zu verarbeiten, wodurch sich viele neue Anwendungsgebiete erschließen, sowie ältere Gebiete unter neuen Voraussetzungen reevaluieren lassen. Dieses Kapitel führt in das Thema ein und erläutert drei homomorphe Kryptosysteme, ein Secret Key Verfahren und zwei Public Key Verfahren.

In der Literatur wird zwischen mathematisch unterschiedlichen homomorphen Kryptosystemen unterschieden. *Somewhat homomorphic* (etwa: *halbwegs homomorph*) sind solche Systeme, die als quantitatives Merkmal nur eine beschränkte Anzahl von mathematischen Operationen (hauptsächlich begrenzt ist die Anzahl der Multiplikationen) in einer Sequenz für einen Chiffretext zulassen und *fully homomorphic* Systemen, die dieser Beschränkung nicht unterliegen. In dieser Arbeit werden *somewhat homomorphic* Systeme als *beschränkt*

homomorph betitelt. Ein qualitatives Unterscheidungsmerkmal ist, ob das betreffende System im Bezug auf nur einen Operationstyp homomorph ist oder ob zwei Operationen möglich sind. Der letztere Fall ist dann *algebraisch* homomorph und kann beispielsweise Additionen und Multiplikationen auf Chiffretexten ausführen. Ein algebraisch homomorphes System hat den Vorteil, dass sich damit beliebige Operationen konstruieren lassen, da sich aus der Addition und Multiplikation alle denkbaren Operationen zusammensetzen lassen.

3.1 Sicherheit homomorpher Kryptosysteme

Dieser Abschnitt erläutert einige Begriffe und Zusammenhänge, die die theoretische Sicherheit von Kryptosystemen betreffen, mit besonderer Berücksichtigung homomorpher Kryptografie.

- *Nicht-Unterscheidbarkeit* (Indistinguishability, IND) ist eine wichtige Eigenschaft eines probabilistischen Kryptosystems. Ein Angreifer kann hier zwei Klartextnachrichten m_0 und m_1 frei wählen. Er erhält eine der beiden Nachrichten verschlüsselt zurück, wobei die Wahrscheinlichkeit, ob m_0 oder m_1 zurückgeliefert wird genau $\frac{1}{2}$ beträgt. Der Angreifer muss nun unterscheiden, welche der beiden Klartextnachrichten in dem Chiffretext verschlüsselt wurde. Das Kryptosystem ist IND-sicher, wenn dies dem Angreifer mit einer Wahrscheinlichkeit von $\frac{1}{2} + \epsilon$ gelingt, wobei ϵ vernachlässigbar klein ist. Probabilistisch ist ein Kryptosystem, wenn die Verschlüsselung nicht deterministisch ist, wenn also zwei Verschlüsselungsvorgänge für denselben Klartext unterschiedliche Chiffretexte ergeben können.

- *Angriff mit wählbarem Klartext* (Chosen plaintext attack, CPA) ist ein Angriffsmodell, bei welchem dem Angreifer gestattet wird, einen Klartext beliebig zu wählen und er im Anschluss an eine korrekte Verschlüsselung dieses Klartextes gelangt.

- *Angriff mit wählbarem Chiffretext* (Chosen ciphertext attack, CCA1) ist ein Angriffsmodell, bei dem einem Angreifer gestattet wird, beliebig viele bestehende Chiffretexte zu wählen, für die ihm dann die Klartexte

zur Verfügung gestellt werden. Im Anschluss muss er einen Chiffretext eigenständig entschlüsseln.

- *Angriff mit adaptiv wählbarem Chiffretext* (Adaptive chosen ciphertext attack, CCA2[1]) ist ein Angriffsmodell, bei welchem der Angreifer zu beliebigen Zeitpunkten beliebig viele Chiffretexte entschlüsseln lassen darf, mit Ausnahme des Chiffretextes, den er eigenständig entschlüsseln soll.

Offensichtlich kann kein homomorphes Kryptosystem die IND-CCA2 Eigenschaft und damit den höchsten theoretisch möglichen Sicherheits-Level besitzen. So könnte ein Angreifer beliebige bereits entschlüsselte Chiffretexte mit einer verschlüsselten Konstante multiplizieren und damit an für ihn unterscheidbare Verschlüsselungen der mit derselben entschlüsselten Konstante multiplizierten Klartexte gelangen. Derzeit ist noch unbekannt, ob die IND-CCA1 Eigenschaft für homomorphe Kryptosysteme erreichbar ist. Durch eine probabilistische Eigenschaft erreichen die meisten homomorphen Schemata zumindest die IND-CPA Eigenschaft (siehe auch den folgenden Abschnitt).

Im Gegensatz dazu besitzt beispielsweise das (multiplikativ-homomorphe) RSA-Kryptosystem nicht die IND-CPA Eigenschaft, da es eine deterministische Verschlüsselungsfunktion verwendet und so ein Angreifer leicht die Zuordnung eines Chiffretextes zu einem Klartext ermitteln kann, indem er das IND-CPA Experiment mit $m_0 = m_1$ durchführt.

Die Sicherheitsanalyse eines Kryptosystems erfolgt stets unter der Annahme eines Angreifermodells. Dieses legt das Verhalten des Angreifers fest und definiert die Fähigkeiten, die dieser im Bezug auf ein bestimmtes Szenario besitzt.

- *Honest-but-curious* Adversary (etwa: ehrlich aber neugierig) hält sich in einem Ablauf zwischen zwei oder mehr Parteien an das vorgesehene Protokoll. Er versucht jedoch, an mehr Informationen zu gelangen als für ihn vorgesehen sind. Allerdings kann er dazu lediglich die laut Protokoll anfallenden Daten lesend auswerten und für seine Zwecke verwenden.

- *Malicious* Adversary (wörtlich: bösartig), handelt im Gegensatz zum lesenden Angreifer in nicht vorhersehbarer Weise. Insbesondere ist nicht

[1] auch *non-malleability*, etwa: Manipulationsresistenz

gesichert, dass der manipulierende Angreifer dem Protokoll folgt und definitionsgemäße Werte bei der Kommunikation verwendet. Bei diesem Angreifermodell muss zudem damit gerechnet werden, dass das Protokoll verzögert oder vorzeitig abgebrochen wird.

Die Sicherheitsbetrachtungen in dieser Arbeit erfolgen, sofern nicht anders gekennzeichnet, unter der Annahme eines Honest-but-curious Angreifermodells. Im Rahmen des verteilten Rechnens können solche Angreifer auch noch während oder nach Durchführung eines Angriffs Informationen vom Angegriffenen erhalten. Dies ist bei einem destruktiven Angriff, bei dem sich der Angreifer offensichtlich regelwidrig verhält, nicht der Fall.

3.2 Überblick über homomorphe Kryptosysteme

Bisher wurden etliche homomorphe Schemata entdeckt, die meisten sind jedoch auf die eine oder andere Weise begrenzt. Die folgenden Unterabschnitte fassen die homomorphen Eigenschaften einiger begrenzt homomorpher Kryptosysteme anhand der Überblicksarbeit [37] zusammen. Dabei erheben die Angaben keinen Anspruch auf Vollständigkeit und stellen jeweils auch nur die homomorphe Eigenschaft des jeweiligen Kryptosystems verkürzt dar. Dadurch soll ein erster Eindruck für die Vergleichbarkeit der Systeme vermittelt werden.

Das RSA-Kryptosystem

Das RSA-Kryptosystem beherrscht die homomorphe Basisoperation einer Multiplikation modulo n. Für zwei Chiffretexte $c_1 = {m_1}^e \; mod \, n$ und $c_2 = {m_2}^e \; mod \, n$ und einen öffentlichen Schlüssel e gilt

$$c_1 c_2 \; mod \; n = {m_1}^e {m_2}^e \; mod \; n$$

$$= (m_1 m_2)^e \; mod \; n$$

und ergibt so das verschlüsselte Ergebnis von $m_1 m_2$. Da das RSA-Kryptosystem nicht IND-CPA-sicher ist, wird die homomorphe Eigenschaft selten genutzt.

Das Goldwasser-Micali Kryptosystem

Das probabilistische Kryptosystem von Goldwasser und Micali bietet eine homomorphe Modulo-2-Addition (XOR). Für zwei Chiffretexte $c_1 = -1^{m_1} {r_1}^2$ und $c_2 = -1^{m_2} r_2$ mit den Zufallszahlen r_1, r_2 gilt

$$\begin{aligned} c_1 c_2 &= (-1^{m_1} {r_1}^2)(-1^{m_2} {r_2}^2) \\ &= -1^{(m_1+m_2)}(r_1 r_2)^2 \\ &= -1^{(m_1+m_2 \ mod\ 2)}(r_1 r_2)^2 \end{aligned}$$

und $c_1 c_2$ ist eine Verschlüsselung des Ergebnisses von $m_1 + m_2 \ mod\ 2$.

Das El-Gamal Kryptosystem

Das El-Gamal Public-Key Kryptosystem beherrscht die homomorphe Multiplikation zweier Chiffretexte sowie die gemischt-homomorphe Multiplikation zwischen einem Chiffretext und einem Klartextfaktor, sowie die gemischt-homomorphe Potenzierung zwischen Chiffretext und Klartextexponent. Für zwei Chiffretexte (c_1, c_2) und (d_1, d_2) als Verschlüsselungen der Klartexte m_1, m_2 unter Zuhilfenahme zweier Zufallswerte y_1, y_2 und dem öffentlichen Schlüssel (g, h) gilt

$$\begin{aligned} (c_1 d_1, c_2 d_2) &= (g^{y_1} g^{y_2}, (m_1 h^{y_1})(m_2 h^{y_2})) \\ &= (g^{y_1+y_2}, m_1 m_2 h^{y_1+y_2}) \end{aligned}$$

und $(c1d1, c2d2)$ ist eine Verschlüsselung des Ergebnisses von $m_1 m_2$.

Das Benaloh Kryptosystem

Das Benaloh Kryptosystem kann Chiffretexte homomorph addieren und subtrahieren. Außerdem sind die Multiplikation und die Potenzierung mit Klartextkonstanten möglich. Bei zwei Chiffretexten $c_1 = y^{m_1} {u_1}^r$ und $c_2 = y^{m_2} {u_2}^r$, einer Zufallszahl r und dem öffentlichen Schlüssel (y, n) gilt

$$\begin{aligned} c_1 c_2 \ mod\ n &= (y^{m_1} {u_1}^r)(y^{m_2} {u_2}^r) \ mod\ n \\ &= y^{m_1+m_2}(u_1 u_2)^r \ mod\ n \end{aligned}$$

und $c_1 c_2$ ist eine Verschlüsselung des Ergebnisses von $m_1 + m_2$. Ferner ist

$$c_1 c_2^{-1} \ mod \ n = (y^{m_1} {u_1}^r)(y^{m_2} {u_2}^r)^{-1} \ mod \ n$$
$$= (y^{m_1} {u_1}^r)(y^{-m_2} (u_2^{-1})^{\ r}) \ mod \ n$$
$$= y^{m_1 - m_2} (u1u2^{-1})^{\ r} \ mod \ n$$

die Verschlüsselung der Ergebnisses von $m_1 - m_2$.

Das Okamoto-Uchiyama Kryptosystem

Das probabilistische Okamoto-Uchiyama Kryptosystem beherrscht die homomorphe Addition und Subtraktion zweier Chiffretexte sowie die gemischt-homomorphe Multiplikation zwischen Chiffretext und Klartextfaktor und die gemischt-homomorphe Addition zwischen Chiffretext und Klartextsummand. Bei zwei Chiffretexten $c_1 = g^{m_1} h^{r_1}$ und $c_2 = g^{m_2} h^{r_2}$, dem öffentlichen Schlüssel (g, h, n) und Zufallszahlen r_1, r_2 gilt

$$c_1 c_2 = g^{m_1 + m_2} h^{r_1 + r_2} \ mod \ n$$

wobei $c_1 c_2$ das verschlüsselte Ergebnis von $m_1 + m_2$ ist. Es gilt außerdem für eine Konstante k

$${c_1}^k = g^{k m_1} h^{k r_1} \ mod \ n$$

wodurch die Verschlüsselung des Ergebnisses von $k m_1$ dargestellt wird.

Das Paillier Kryptosystem

Das probabilistische Kryptosystem von Paillier beherrscht homomorphe Addition und Subtraktion von Chiffretexten, die gemischt-homomorphe Addition und Subtraktion zwischen Chiffretext und Klartextsummand sowie die gemischt homomorphe Multiplikation zwischen Chiffretext und Klartextfaktor. Seien bei einem öffentlichen Schlüssel (g, n) und Zufallszahlen r_1, r_2 die betrachteten Chiffretexte $c_1 = g^{m_1} r_1^n \ mod \ n^2$ und $c_2 = g^{m_2} r_2^n \ mod \ n^2$. Dann ist

$$c_1 c_2 \ mod \ n^2 = g^{m_1} {r_1}^n g^{m_2} {r_2}^n = g^{m_1 + m_2} r_1 {r_2}^n \ mod \ n^2$$

eine Verschlüsselung des Ergebnisses von $m_1 + m_2$. Ferner ist

$$c_1 g^k \ mod \ n^2 = g^{m_1} {r_1}^n g^k = g^{m_1 + k} {r_1}^n \ mod \ n^2$$

eine Verschlüsselung von $m_1 + k$ und außerdem

$$c_1{}^k \ mod \ n^2 = (g^{m_1} r_1{}^n)^k = g^{km_1} (r_1{}^k)^n \ mod \ n^2$$

die Verschlüsselung von km_1.

Das **Boneh-Goh-Nissim Kryptosystem** ähnelt dem Paillier Kryptosystem. Allerdings gestattet das BGN-System eine abschließende homomorphe Multiplikation. In der Praxis ist so die Auswertung verschlüsselter 2-DNF[2] Ausdrücke möglich.

3.3 Ein algebraisches, begrenzt homomorphes Kryptosystem

Die in diesem Abschnitt eingeführten Grundlagen zu einem homomorphen Secret-Key Kryptosystem sowie die Analysen und Beispielanwendungen im Folgeabschnitt wurden in [11] wissenschaftlich publiziert.

Dieser Abschnitt beschreibt ein sehr einfaches, algebraisch homomorphes Secret-Key Schema, welches auf der Faktorisierung der Näherung einer Semiprimzahl, d. h. einer aus zwei primen Faktoren und einem Additiv zusammengesetzten Zahl beruht. Das zugrundeliegende mathematische Problem ist in der englischsprachigen Literatur auch als *Approximate GCD-Problem* bekannt. Die Konstruktion in dieser Arbeit gestattet jedoch die direkte Reduktion auf das Faktorisierungsproblem. Im Gegensatz zu einem unbegrenzt homomorphen Schema handelt es sich hierbei um ein homomorphes Schema, bei dem die Anzahl der Operationen, vor allem der Multiplikationen, im verschlüsselten Raum von der Schlüssellänge abhängig und begrenzt ist. Hier werden die Eigenschaften des Schemas untersucht und einige grundlegende Protokolle vorgestellt, die sich mit dem Schema elegant lösen lassen. Zum einen handelt es sich um die bekannten Szenarien des *Oblivious Transfer* und das *Millionärsproblem*, zum anderen werden fortgeschrittene Techniken bearbeitet, wie die geheime Auswahl aus vielen Elementen (analog einem Speicherzugriff) oder die geheime Suche

[2]Die binäre disjunktive Normalform beschreibt in der booleschen Algebra Terme der Form $(a \wedge b) \vee (a \wedge c) \vee (c \wedge d)$

in vielen Elementen mit verschlüsselten Suchbegriffen. Diese Grundlagen werden in späteren Abschnitten aufgegriffen, um komplexere Problemstellungen zu lösen.

3.3.1 Basisschema

Das begrenzt homomorphe Schema $\mathfrak{H}$ ist von den folgenden Parametern abhängig:

- der Sicherheitsparameter λ
- die Anzahl η der Bits des Wertebereiches eines Chiffrezeichens
- der prime Modul p, der geheime Schlüssel der Größenordnung 2^{λ}
- die Anzahl ρ der Bits des in einer Chiffre verfügbaren Nachrichtenraums, $\lambda - \eta$

Das Kryptosystem beherrscht verschlüsselte, positive Ganzzahlarithmetik im Zahlenbereich $1..p$. Für binäre (mod 2) Arithmetik wird der Klartext so gewählt, dass die Parität dem zu verschlüsselnden Binärwert entspricht. Das Schema $\mathfrak{H}$ ist durch das Tupel $\{P, C, K, E, D, \oplus, \otimes\}$ gekennzeichnet, dessen Elemente wie folgt definiert sind:

P ist der Klartextraum $\mathbb{N}^P$, bestehend aus Elementen der Menge $\mathbb{N}^+$, begrenzt durch den primen Modul p der Größenordnung 2^{λ}, sodass für zwei Operanden im Klartext $a, b \in \mathbb{N}^P$ gilt: $a \cdot b < p$ und $\mathbb{N}^P := \{x | 0 < x < 2^{\eta}\}$.

C ist der Chiffretextraum $\mathbb{N}^+$.

K ist die Funktion zum Generieren des geheimen Schlüssels $p \in \mathbb{N}^+$. Zusätzlich wird ein Kompressionsargument d mit $d \leftarrow 2s + rp$ und $r \in \mathbb{N}^+$ sowie $s \in \mathbb{N}^C$ erzeugt, durch das die Kompaktheit hergestellt wird (siehe 3.3.4), wobei die Elemente in $\mathbb{N}^C$ in Abhängigkeit von $p \in \mathbb{N}^P$ so gewählt werden, dass $\forall x \in \mathbb{N}^C, \forall y \in \mathbb{N}^P : 0 < 2x < y$.

E ist die Verschlüsselungsfunktion. Ein Operand im Klartext a wird verschlüsselt, indem ein zufälliges Vielfaches des Schlüssels p aufaddiert wird: $a' = a + (rp)$. Wenn r nicht prim ist, so muss es mindestens einen großen Primfaktor besitzen. Außerdem muss das verschlüsselte a' in $\mathbb{N}^+$ liegen.

D ist die Entschlüsselungsfunktion. Das Klartextergebnis ist der Teilrest einer Chiffre modulo des Schlüssels p: $a = a' \; mod \; p \; mod \; 2$.

$\oplus$ ist die Addition im Chiffretextraum. Aufgrund der Struktur des Schemas handelt es sich hierbei um die gewöhnliche Addition für Ganzzahlen. Das Schema ist gemischt additiv, d. h. $D(E(a)+E(b)) = a+b$ und $D(E(a)+b) = a+b$.

$\otimes$ ist die Multiplikation im Chiffretextraum. Aufgrund der Struktur des Schemas handelt es sich hierbei um die gewöhnliche Multiplikation für Ganzzahlen. Das Schema ist gemischt multiplikativ, d. h. $D(E(a) \cdot E(b)) = a \cdot b$ und $D(E(a) \cdot b) = a \cdot b$.

In diesem Schema ist der positive Klartext gleichzeitig das *Rauschen* für den Chiffretext, denn er interferiert mit dessen restfreier Primzahlenzerlegung. Das Rauschen zwingt den Angreifer dazu, Annahmen über die numerische Nähe eines Chiffrats zu einer Semiprimzahl zu treffen und erhöht den Aufwand der Zerlegung. Zur Herstellung eines probabilistischen Verfahrens wird im Kryptoraum Arithmetik modulo 2 angewendet, d. h. ein Klartext-Bit wird in einer zufälligen Ganzzahl-Repräsentation a mit gleicher Parität dargestellt und daraufhin verschlüsselt. Dies führt zu einer beliebigen Anzahl von unterschiedlichen Repräsentationen eines Klartextes. Die verschlüsselte Darstellung besitzt im Gegensatz zur Ganzzahl-Repräsentation a nicht mehr die gleiche Parität wie das Klartext-Bit, da zur Verschlüsselung zufällige, gerade oder ungerade Vielfache r des Schlüssels p zu a addiert werden. Dieses Vorgehen ist geeignet, die Originalparität zu verbergen. Im Falle eines ungeraden Vielfachen von p sollte r ebenfalls eine große Primzahl sein, im Falle eines geraden Vielfachen das Doppelte einer großen Primzahl, um trotzdem einen großen primen Faktor zu garantieren. Durch die randomisierte Verschlüsselung ergibt sich idealerweise, dass die Paritäten der Chiffretexte gleichverteilt sind, d. h. es gilt für alle

n Chiffretexte a'_i von Ganzzahl-Repräsentationen a_i

$$\frac{\sum_{i=0}^{n} (a'_i \ mod \ 2)}{n} = 0.5$$

Lemma 3.3.1. *Die Paritäten der Chiffretexte sind normalverteilt. Die diskrete Wahrscheinlichkeitsverteilung der unabhängigen Ereignisse Klartext und Schlüssel ergibt, dass es dabei nicht von Bedeutung ist, welche Paritätsverteilung bei den Klartexten besteht.*

Beweis. Es sei die Wahrscheinlichkeit, dass die Repräsentation

$$a' = a + rp$$

eines Klartextes gerade ist Pr_g und das Gegenteil

$$Pr_u = 1 - Pr_g.$$

Dieses Verhältnis lässt sich an einer Menge vorliegender Klartexte einfach ermitteln, wobei in jedem Falle gilt:

$$Pr_g + Pr_u = 1.$$

Weiterhin ist die Wahrscheinlichkeit Pr_0, dass die Parität eines Schlüssels rp gerade ist, wegen der Zufallseigenschaft von r und der Primzahleigenschaft von p genau $\frac{1}{2}$, ebenso das Gegenteil Pr_1. Somit gilt

$$Pr[a' \ ist \ gerade] = (Pr_g \cdot Pr_0) + (Pr_u \cdot Pr_1) = \frac{1}{2} \underbrace{(Pr_g + Pr_u)}_{=1} = \frac{1}{2}$$

und zwar unabhängig von der Parität des verschlüsselten Klartextes. □

3.3.2 Richtigkeit

Dieser Abschnitt untersucht die Richtigkeit sowohl der Verschlüsselung und Entschlüsselung als auch der homomorphen Operationen. Die folgenden Lemmata stehen unter der Annahme, dass der geheime Schlüssel $p \in \mathbb{N}^+$ eine große Primzahl ist und dass die beiden Operanden a und b zwei beliebige positive Ganzzahlen mit $a, b \ll p \in \mathbb{N}^+$ sind.

Lemma 3.3.2. *Das Schema $\mathfrak{H}$ ist gemischt additiv und die Addition ist korrekt, wenn $a + b < p$.*

Beweis. Die verschlüsselte Addition $(a' \oplus b')$ ist definiert als

$$(a' \oplus b') = (a + r_1 p) + (b + r_2 p)$$

$$= a + b + (r_1 + r_2)p$$

und ergibt entschlüsselt $mod\ p$ das Resultat $(a + b)$. Die gemischt additive Operation ist definiert als

$$(a' + b) = (a + rp) + b = a + rp + b\ mod\ p = (a + b).$$

□

Lemma 3.3.3. *Das Schema $\mathfrak{H}$ ist gemischt multiplikativ und die Multiplikation ist korrekt, wenn $a \cdot b < p$.*

Beweis. Die verschlüsselte Multiplikation $(a' \otimes b')$ ist definiert als

$$(a' \otimes b') = (a + r_1 p)(b + r_2 p)$$

$$= ab + a(r_2 p) + b(r_1 p) + (r_1 r_2)p^2$$

$$mod\ p = (ab)$$

Die gemischt multiplikative Operation ist definiert als

$$(a'b) = (a + rp) \cdot b = ab + brp\ mod\ p = (ab).$$

□

Lemma 3.3.4. *Das Schema $\mathfrak{H}$ ist in der Lage, eine Sequenz von mindestens $log_2\rho - log_2\eta$ arithmetischen Operationen korrekt auszuführen.*

Beweis. Die Wahl der Faktoren r und p definiert den für Nutzdaten zur Verfügung stehenden Bereich ρ. Die Anzahl der Bits errechnet sich aus der Differenz der Größe des Rauschens und der Größe des primen Schlüssels p: $\rho = log_2 p - \eta$. Das Rauschen eines Chiffretextes wächst mit jeder Addition um höchstens ein Bit (den Additionsübertrag), die mindestens korrekte Anzahl von Additionen ist somit ρ. Bei einer Multiplikation sind die Bitlängen des jeweiligen Rauschens der Operanden zu addieren, sodass mindestens $log_2\rho - log_2\eta$ Multiplikationen korrekt ausgeführt werden können. Dies steht unter der Annahme, dass immer die Operanden mit dem größten Rauschen miteinander verknüpft werden, wobei n Multiplikationen höchstens $\eta 2^n$ Bits an Rauschen erzeugen können. □

3.3.3 Sicherheit

Wie oben gezeigt wurde, ist das Schema $\mathfrak{H}$ algebraisch korrekt und kann die erforderlichen Operationen korrekt darstellen. Um die Sicherheit, genauer: die Vertraulichkeit der verschlüsselten Daten zu untersuchen, werden unterschiedliche Angriffsmodelle betrachtet und bewertet. Die Ergebnisse dieses Abschnittes führen dann zur Auswahl geeigneter Parameter für das Schema.

Lemma 3.3.5. *Seien die Parameter $(p, q) \in \mathbb{N}^+$ von der Größenordnung 2^λ. Jeder Angriff $\mathcal{A}$ in λ-Polynomialzeit auf das Verschlüsselungsschema kann in einen Algorithmus $\mathcal{B}$ umgewandelt werden, mit dem die Faktorisierung beliebiger ganzer Zahlen in λ-Polynomialzeit gelöst werden kann.*

Beweis. Sei $\mathcal{A}_{ex}$ eine λ-Polynomialzeitfunktion

$$p \leftarrow \mathcal{A}_{ex}(a')$$

mit welcher ein Primfaktor p aus einem beliebigen Chiffretext a' effizient berechnet werden kann. Das bedeutet, dass $\mathcal{A}_{ex}$ in der Lage ist, p aus beliebigen zusammengesetzten Zahlen der Form $a + pq$ zu extrahieren und kann somit in der Funktion

$$\mathcal{B}_{fac}(i) : p \leftarrow \mathcal{A}_{ex}(0 + i)$$

verwendet werden, um beliebige Ganzzahlen i zu faktorisieren, die sich trivialerweise als $0 + i$ darstellen lassen. □

Im Folgenden wird die Sicherheit des Verschlüsselungsschemas gegen einen Angreifer $\mathcal{A}$ gezeigt, der den Klartext aus einem ihm vorliegenden Chiffretext berechnen will. Dazu wird aufbauend auf Lemma 3.3.5 eine IND-CPA[3] äquivalente Eigenschaft des Verschlüsselungsschemas gezeigt. Das bedeutet, dass zwei Chiffretexte von einem Angreifer praktisch nicht zu unterscheiden sind.

Lemma 3.3.6. *Die Sicherheit des Verschlüsselungsschemas ist IND-CPA äquivalent und die Erfolgswahrscheinlichkeit* $Pr[Exp_{ind-cpa} = 1] - \frac{1}{2}$ *ist für alle probabilistischen Polynomialzeitalgorithmen vernachlässigbar.*

Beweis. Da $\mathcal{A}$ kein öffentlicher Schlüssel zur Verfügung steht, wird ihm für das folgende Experiment im Nachrichtenraum $\mathcal{M}$ Zugriff auf ein Verschlüsselungsorakel $\mathcal{O}_{Enc}$ gestattet. Dieses gibt für einen eingegebenen Klartext einen Chiffretext unter dem geheimen Schlüssel aus.

$$
\begin{array}{ll}
\boldsymbol{Exp_{ind-cpa}} = \{ & \\
\{m_0, m_1\} & \leftarrow \mathcal{M} \\
i & \xleftarrow{R} \{0,1\} \\
c & \leftarrow \mathcal{O}_{Enc}(m_i) \\
i_{\mathcal{A}} & \leftarrow \mathcal{A}_{ind-cpa}(c, \{m_0, m_1\}) \\
\text{return} & \begin{cases} 1 & \text{wenn } i = i_{\mathcal{A}} \\ 0 & \text{sonst} \end{cases} \\
\} &
\end{array}
$$

Das Verschlüsselungsorakel $\mathcal{O}_{Enc}$ ist wie folgt definiert:

$$
\begin{array}{ll}
\boldsymbol{\mathcal{O}^{\lambda,p}_{Enc}(m)} = \{ & \\
a & \xleftarrow{R} 2[2^{\eta}] \\
r & \xleftarrow{R} [2^{\lambda}]
\end{array}
$$

[3]IND-CPA bedeutet 'indistinguishable under chosen plaintext attack', dem Angreifer wird also gestattet, zwei Klartexte zu wählen, bevor diese verschlüsselt werden; bei einem IND-CPA sicheren Kryptosystem kann der Angreifer die entstehenden Chiffrate nicht mehr den Klartexten zuordnen, siehe auch Abschnitt 3.1

$c \quad \leftarrow (m \ mod\ 2) + a + rp$

return c

}

Die Parität der Ausgabe des Orakels $\mathcal{O}_{Enc}(m)$ ist gleichverteilt, sodass $Pr[\mathcal{O}_{Enc}(m) \equiv_2 0] = Pr[\mathcal{O}_{Enc}(m) \equiv_2 1] = \frac{1}{2}$ für beliebige ganze Zahlen $m \in \mathbb{N}^+$, d. h. die Verschlüsselungsfunktion gestattet keinen Rückschluss auf die Parität der Klartexte. Dazu genügt es, mit Hilfe von Lemma 3.3.1 zu zeigen, dass $Pr[\mathcal{O}_{Enc}(m) \equiv_2 1] = \frac{1}{2}$.

$$Pr[\mathcal{O}_{Enc}(m) =_2 1] = Pr[(m \ mod\ 2) + a + r \cdot p \equiv_2 1]$$

$$Pr[(m \ mod\ 2) + a \equiv_2 1] \cdot \underbrace{Pr[r \cdot p \equiv_2 0]}_{=0.5} + \underbrace{Pr[(m \ mod\ 2) + a \equiv_2 0]}_{=0} \cdot \underbrace{Pr[r \cdot p \equiv_2 1]}_{=0.5}$$

$$\frac{1}{2} \underbrace{(Pr[(m \ mod\ 2) + a \equiv_2 1])}_{=1} = \frac{1}{2}$$

□

Lemma 3.3.7. *Ein lesender Angreifer (honest-but-curious), welcher während des Transports oder während der Verarbeitung eine durch den geheimen Schlüssel p verschlüsselte Nachricht mitliest, kann keine inhaltlichen Informationen über die Nachricht oder den geheimen Schlüssel p ableiten.*

Der lesende Angreifer hat keine außer den öffentlichen bzw. den gemäß des Protokolls verfügbaren Informationen. Diese Informationen sind entweder unverschlüsselt oder durch den geheimen Schlüssel p verschlüsselt. Der Angreifer unterliegt somit der in Lemma 3.3.5 getroffenen Annahme.

Lemma 3.3.8. *Der Protokollteilnehmer ist ein manipulierender (malicious) Angreifer. Auch durch absichtliches Abweichen von Protokollen und Rechenvorschriften kann der manipulierende Angreifer keine Schlüsse auf Klartextdaten oder den geheimen Schlüssel ziehen.*

Auch der manipulierende Angreifer hat keine Informationen außer den öffentlich verfügbaren (möglicherweise unverschlüsselten) Daten oder solchen Daten, die mit dem geheimen Schlüssel p verschlüsselt wurden. Dies schließt

gleichzeitig aus, dass der Angreifer durch manipulierte oder gefälschte Daten einen Chiffretext entschlüsseln oder den geheimen Schlüssel berechnen kann. Destruktive Abweichungen vom Protokoll führen zum Abbruch der Verarbeitung und somit ergibt sich dadurch für den Angreifer kein Vorteil beim Brechen der Verschlüsselung.

Folgerung

Das Schema $\mathfrak{H}$ ist sicher, wenn nicht mehr als ein korrektes Paar von Klartext und Chiffretext für einen Angreifer verfügbar ist. Für die Protokolle und Algorithmen in diesem Kapitel ist diese Einschränkung hinnehmbar, da es sich ausschließlich um Delegationsszenarien handelt, bei denen eine Partei die Berechnung von Daten an eine andere Partei auslagert. Positiv wirken sich die gemischt-additiven und gemischt-multiplikativen Eigenschaften auf eine solche Delegation aus, da auf diese Weise Klartextdaten ohne vorherige, asymmetrische Verschlüsselung verarbeitet werden können. Dazu muss der verarbeitenden Partei lediglich ein initialer Chiffretext zugänglich gemacht werden.

3.3.4 Kompaktheit

Die Länge der Chiffretexte steigt exponentiell mit der Anzahl der auf sie angewendeten Multiplikationen. Ziel dieses Abschnitts ist es, die *Kompaktheit* des Verschlüsselungsschemas nachzuweisen, d. h. dass die Länge eines Chiffrats unabhängig von der Anzahl der darauf angewendeten Operationen ist. Zu diesem Zweck wird ein Kompressionsargument d eingeführt, das im Prinzip eine verschlüsselte 0 darstellt.

Lemma 3.3.9. *Die Operation $a'' \leftarrow a' - \lfloor \frac{a'}{d} \rfloor \cdot d$ mit $d \leftarrow 2s + rp$ reduziert die Bitlänge eines Chiffrats a' auf 2λ. Diese Reduktion ist korrekt für alle $2s \in \mathbb{N}^+ < a$, d. h. die Parität des Klartextes bleibt erhalten.*

Beweis. Die Reduktion von $a' \leftarrow a + r_1 p$ by $d \leftarrow 2s + r_2 p$ wird durchgeführt mit $a'' \leftarrow a' - \delta$ mit

$$\delta \leftarrow \lfloor \frac{a'}{d} \rfloor \cdot d \Rightarrow \delta \leftarrow \lfloor \frac{a + r_1 p}{2s + r_2 p} \rfloor \cdot (2s + r_2 p)$$

$$I : a + r_1 p < 2s + r_2 p \Rightarrow \delta = 0$$

$$II : a + r_1 p \geq 2s + r_2 p \Rightarrow n = \lfloor \frac{a + r_1 p}{2s + r_2 p} \rfloor, n \geq 1$$

$$\delta \leftarrow n \cdot (2s + r_2 p) = 2ns + nr_2 p$$

$$a'' \leftarrow a + \underbrace{r_1 p}_{mod\ p=0} - \underbrace{2ns}_{gerade} - \underbrace{nr_2 p}_{mod\ p=0} \ mod\ p \equiv_2 a$$

□

3.3.5 Parameterwahl

Wie oben gezeigt wurde, basiert die Sicherheit des Verschlüsselungsschemas auf der bekannten Annahme, dass die Faktorisierung einer großen Zahl schwierig ist. Aus diesem Grunde muss das Produkt aus dem primen Schlüssel p und dem Zufallsfaktor r hinreichend groß sein, um eine Instanz dieses Problems zu erzeugen. Angesichts der früheren *RSA-Challenge*, bei der wettbewerbsmäßig große Zahlen zu faktorisieren waren, sollten die Faktoren größer als 512 Bits sein. Bei einer Faktorgröße von 1024 Bits ($\lambda = 1024$) ergibt sich bei einer gleichzeitigen initialen Größenordnung von $\eta = 8$ Bits für das Rauschen eine fehlerfrei berechenbare Sequenz von $log_2\lambda - log_2\eta = 7$ Multiplikationen.

Kleine Faktoren. Die Verschlüsselungsfunktion darf keine Chiffretexte mit $a = 0$ erzeugen, um die Entstehung kleiner Faktoren zu unterbinden: werden zwei unterschiedliche Verschlüsselungen von 0 mit $a'_1 = 0 + r_1 p$ und $a'_2 = 0 + r_2 p$ generiert, könnte ein Angreifer durch Subtraktion leicht eine zusammengesetzte Zahl $b' = |(a'_1 - a'_2)| = |(r_1 - r_2)|p$ erzeugen, die mit hoher Wahrscheinlichkeit einen verhältnismäßig kleinen Faktor $|(r_1 - r_2)|$ enthält, da für r_1 und r_2 die gleiche Größenordnung vorausgesetzt werden kann.

3.4 Das algebraische, unbegrenzt homomorphe Gentry Kryptosystem

Dieser Abschnitt behandelt das Kryptosystem von Craig Gentry [27]. Im Gegensatz zu dem im vorherigen Abschnitt skizzierten Schema basiert es auf einer Anwendung der Gittertheorie, einem Teilgebiet der Geometrie, wobei die

Werte der verschlüsselten Operanden durch Positionen im Raum im Verhältnis zu bestimmten Bezugspunkten dargestellt werden[4]. Vor der Konstruktion Gentrys wird kurz in den Zusammenhang zwischen der Gittertheorie und der Kryptografie eingeführt. Eine vollständige Einführung findet sich in [54].

3.4.1 Gittertheorie und Kryptografie

Zunächst werden die im weiteren Verlauf verwendeten Begriffe definiert.

Definition 3.4.1 (Gitter)**.** Seien $b_1, b_2, ..., b_n$ linear unabhängige Vektoren im m-dimensionalen Raum $\mathbb{R}^m$. Die diskrete additive Untergruppe

$$L(b_1, b_2, ..., b_n) := \sum_{i=1}^{n} b_i\mathbb{Z} = \left\{ \sum_{i=n}^{n} t_i b_i \middle| t_1, t_2, ..., t_m \in \mathbb{Z} \right\}$$

bildet ein Gitter in $\mathbb{R}^m$ zur Basis $b_1, b_2, ..., b_n$. Die Dimension (auch Rang) des Gitters ist $rg(L) := n$. Gilt $rg(L) = m$, so ist das Gitter *vollständig.*

Definition 3.4.2 (Untergitter)**.** Seien ferner $L, U \subset \mathbb{R}^m$ Gitter. U ist genau dann ein Untergitter von L, falls $U \subseteq L$. Untergitter von $\mathbb{Z}^m$ heißen *ganzzahlig.*

Definition 3.4.3 (Basismatrix)**.** Die Matrix $B := [b_1, b_2, ..., b_n] \in \mathbb{R}^{m \times n}$ wird Basismatrix zum Gitter L genannt. $L(B) := L(b_1, b_2, ..., b_n)$ wiederum ist das von B erzeugte Gitter.

Definition 3.4.4 (Grundmasche)**.** Die Grundmasche zu der Basismatrix $B = [b_1, b_2, ..., b_n]$ ist das Parallelepiped

$$\mathcal{P}(B) := \left\{ \sum_{i=1}^{n} x_i b_i \middle| 0 \leq x_i \leq 1 \right\}$$

Um die Gittertheorie für die Kryptografie nutzbar zu machen, bedarf es schwer zu lösender Probleme im Sinne der Komplexitätstheorie. Die in diesem Zusammenhang relevanten Probleme sind

[4] in der englischsprachigen Literatur auch als *lattice-based cryptography* bezeichnet

- Shortest-Vector-Problem (SVP): hier wird ein Gittervektor $b \in L, b \neq 0$ gesucht, für den die Euklidische Norm $||b||$ minimal wird
- Closest-Vector-Problem (CVP): es soll für einen Punkt $x \in \mathbb{R}^m$ eine Gittervektor $b \in L \subset \mathbb{R}^m$ gefunden werden, sodass der Abstand $||b - x||$ minimal wird
- Shortest-Base-Problem (SBP): Bestimmung der Gitterbasis $b_1, b_2, ..., b_n$ von L mit dem minimalen Produkt $\prod_{i=1}^{n} ||b_i||$

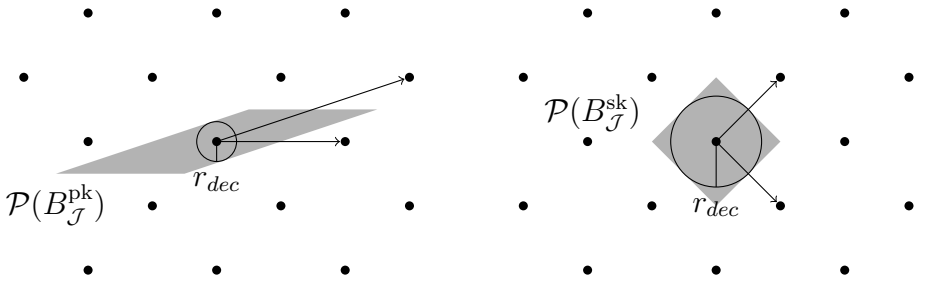
Die Grundlage für Lattice-(Gitter-) Kryptografie sind Ideale über Gittern, wobei ein solches Gitterideal (*ideal lattice*) in diesem Zusammenhang ein Gitter $\subseteq \mathbb{Z}^n$ ist, welches durch ein Ideal im isomorphen Quotientenring $R = \mathbb{Z}[x]/\langle f \rangle$ erzeugt wird. Der Isomorphismus wird durch ein Polynom p im Quotientenring R konstruiert, wobei f ein monisches, irreduzibles Polynom n-ten Grades und Vektor von Koeffizienten ist. Ferner ist p ein Polynom vom maximalen Grad $n - 1$, und es liegt der Koeffizientenvektor von p in $\mathbb{Z}^n$. Somit definiert ein im Bezug auf die Operationen $+$ und $\cdot$ über R geschlossenes Ideal $\mathcal{I} \subseteq \mathbb{Z}[x]/\langle f \rangle$ durch den Isomorphismus ein Untergitter in $\mathbb{Z}^n$.

Die Vorteile des Gitter-basierten Ansatzes sind

- **Einfache Entschlüsselung:** In der Gitter-basierten Kryptografie ist die Entschlüsselung oft eine sehr einfache Operation. Beispielsweise ist dies im Schema von Smart und Vercauteren[55], einer Spezialisierung von Gentry, lediglich eine Matrix-Vektor-Multiplikation modulo 2.
- **Homomorphismus:** Die Anwendung von Gitteridealen in einem Kryptosystem über einem Ringideal als Chiffrenraum resultiert in einem homomorphen Schema, weil Ideale im Bezug auf die Addition $+$ und die Multiplikation $\cdot$ geschlossen sind.

3.4.2 Gentrys Konstruktion

Das Grundkonzept aller Gentry-basierten unbegrenzt homomorphen Schemata ist das Folgende: zunächst basiert das System auf Gitteridealen um die homomorphe Eigenschaft herzustellen. Der Algorithmus zur asymmetrischen Schlüsselgenerierung erzeugt dann zwei Basen $(B_{\mathcal{J}}^{\text{pk}}, B_{\mathcal{J}}^{\text{sk}})$ des Ideals $\mathcal{J}$, welches den



(a) Kleiner Radius r_{dec} für die öffentliche Basis $B_{\mathcal{J}}^{\mathrm{pk}}$

(b) Großer Radius r_{dec} für die geheime Basis $B_{\mathcal{J}}^{\mathrm{sk}}$

Abbildung 3.1: Vergleich der öffentlichen und geheimen Basis.

Chiffrenraum repräseniert. Die erste Basis ist öffentlich, und Gentry schlägt zur Darstellung der entsprechenden Matrix die *Hermite Normalform* (HNF) vor. Die Basis kann leicht durch das Gauss'sche Reduktionsverfahren berechnet werden und besitzt eine eindeutige Darstellung. Die Basisvektoren der zweiten (geheimen) Basis stehen *senkrechter* aufeinander und sind daher geeignet, einen Repräsentanten eines Vektors in dem Gitter zu finden. Abbildung 3.1 zeigt eine grafische Darstellung der beiden Basen. Für jede Basis B des Gitters $\mathcal{J}$ ist ein halboffenes Parallelepiped (auch Spat) $\mathcal{P}(B_{\mathcal{J}}) = \{\sum_i x_i b_i | x_i \in [-\frac{1}{2}, \frac{1}{2}]\}$ (in der Abbildung grau unterlegt) um den Ursprung eingezeichnet, welches das Maschenvolumen darstellt.

Die Operation $y = x \mod B_{\mathcal{J}}$ ist so definiert, dass $x - y$ auf dem Gitter liegt. Zur Veranschaulichung ist dies der nächstgelegene Gitterpunkt, für den das Parallelepiped den gegebenen Vektor beinhaltet. Offensichtlich wirkt sich der Grad der Orthogonalität der Basisvektoren auf die Approximation der Modulo-Operation aus: je näher die Vektoren dem rechten Winkel sind, umso exakter ist auch die Modulo-Operation. Das zugrundeliegende Problem der Approximation ist das NP-schwere *Closest-Vector-Problem* (CVP). Dabei liefert die Basis in HNF-Darstellung keinen Lösungshinweis für das CVP.

Die Verschlüsselung eines Wertes resultiert in einem Vektor, der auf den Gitterpunkt zeigt, der den verschlüsselten Wert repräsentiert. Dieser Vektor liegt im Parallelepiped der Basis $B_{\mathcal{J}}^{\mathrm{sk}}$ um den Gitterpunkt, wodurch die Entschlüsselung einfach als $c \mapsto c \mod B_{\mathcal{J}}^{\mathrm{sk}}$ durchgeführt werden kann. Weiterhin ist das Ergebnis einer Addition zweier Elemente (des Rings $\mathbb{Z}[x]/\langle f \rangle$) ein Element, welches nahe dem Gitterpunkt liegt, der die Summe beider Elemente

repräsentiert und die Multiplikation liefert analog dazu ein Element, das nahe an dem Gitterpunkt liegt, der das Multiplikationsergebnis darstellt. Mit einer gewissen Toleranz können so auch Operationssequenzen durchgeführt werden, allerdings weichen die Zielelemente dann zunehmend von den Gitterpunkten ab, sodass die Entschlüsselung nach einer bestimmten Schwelle nicht mehr korrekt durchzuführen ist. Dies ist im Sinne von Gentrys Arbeiten ein begrenzt homomorphes Schema, weil nur eine begrenzte Anzahl von Operationen fehlerfrei durchgeführt werden kann. In der Theorie boolescher Schaltkreise resultiert daraus ein Schaltkreis von begrenzter Tiefe.

Wenn die Parameter des Schemas so gewählt werden können, dass ein Schaltkreis streng begrenzter Tiefe realisierbar ist, der den Entschlüsselungsalgorithmus des Schemas abbilden kann, ohne selbst dabei *überzulaufen* (Bootstrapping), so lässt sich der geheime Schlüssel in verschlüsselter Form dem Entschlüsselungsschaltkreis bereitstellen, wodurch dann ein verschlüsselter *Recrypt*-Algorithmus konstruiert werden kann. Die Ausgabe dieser Funktion ist eine verschlüsselte Repräsentation des eingegebenen verschlüsselten Wertes mit verringertem geometrischen Fehler im Bezug auf die geheime Basis des Gitterideals. Sofern der Recrypt-Schaltkreis so flach ist, dass auf den resultierenden Repräsentationen noch weitere homomorphe Operationen fehlerfrei ausführbar sind, kann ein unbegrenzt homomorphes Schema konstruiert werden, in dem die Recrypt-Operation zyklisch bei Bedarf auf die verschlüsselten Operanden angewendet wird.

Die ursprüngliche Konstruktion Gentrys besitzt durch die Codierung der Chiffren in Matrixform den Nachteil einer exorbitanten Größe des öffentlichen Schlüssels. Diese liegt je nach Wahl der Parameter des Schemas im Bereich von Gigabytes. Ebenso liegt die Dauer der Recrypt-Operation im Minuten- bis Stundenbereich [28]. Um diese Nachteile zu mildern, existieren einige Varianten und Spezialisierungen des Gentry-Schemas. Im folgenden Abschnitt wird eine solche Spezialisierung beschrieben, deren vorrangiges Ziel *relativ kleine* Schlüssel- und Chiffratlängen sind.

3.5 Das algebraische, unbegrenzt homomorphe Smart-Vercauteren Kryptosystem

Der folgende Abschnitt beschreibt die erste öffentlich verfügbare Implementierung der Variante von Smart und Vercauteren [55] des Gentry-Kryptosystems. *Die hier vorgestellte praktische Umsetzung wurde als studentische Arbeit unter meiner Betreuung erstellt und in [49] wissenschaftlich publiziert.*

Die Implementierung entstand aufgrund der Tatsache, dass die bis dahin einzige und nicht öffentlich verfügbare, prototypische Umsetzung des ursprünglichen Gentry-Systems offensichtlich sogar für experimentelle Arbeiten ungeeignet war[5]. Außerdem konnten Smart und Vercauteren selbst die Implementierung ihres Systems nicht vollständig durchführen, da es ihnen nicht gelang, den *Recrypt*-Mechanismus in der höchstens zulässigen Schaltkreistiefe zu formulieren. Details hierzu werden in diesem Abschnitt diskutiert. Der Quellcode wurde unter einer Open Source Lizenz auf der Website `http://www.hcrypt.com` einer breiteren Öffentlichkeit als experimenteller Prototyp zugänglich gemacht. Dies geschah aufgrund der Überzeugung, dass es erforderlich sei, eine funktionierende Implementierung für weitere Studien und Experimente frei verfügbar zu machen.

Zum tieferen Verständnis dieses Abschnittes ist die Lektüre der Veröffentlichung von Smart et. al [55] hilfreich.

3.5.1 Das begrenzt homomorphe Schema

Das begrenzt homomorphe Schema[6] funktioniert nur so lange korrekt, wie die Abweichung der Koeffizienten $C(x)$ innerhalb der Schranke r bleibt. Dabei hängt diese Schranke im Wesentlichen von einem weiteren Parameter N ab. In der vorliegenden Implementierung ergibt die experimentelle Wahl der Parameter $r = 2^{\nu}/(2\sqrt{N}) = \frac{2^{384}}{16}$. Die generische Berechnung der Eckwerte ist in [55] zu finden.

[5] siehe hierzu auch [28]

[6] siehe auch Kapitel 2.4

Die grundlegende Idee ist, dass $G(x)$ ein Ideal $\mathfrak{p} = (G(x))$ in $\mathbb{Z}[x]/F[x]$ mit der zweielementigen Form $\mathfrak{p} = \langle p, x - \alpha \rangle$ generiert. Der Homomorphismus lautet

$$\begin{aligned} \varphi_{\mathfrak{p}_i} : \mathbb{Z}[x] &\to (\mathbb{Z}[x]/F(x))/\mathfrak{p} \\ C(x) &\mapsto C(\alpha) \mod p \end{aligned} \tag{3.1}$$

Dies wird für die Verschlüsselung verwendet.

Schlüsselerzeugung

Notation Polynome werden als große lateinische Buchstaben dargestellt. Für ein gegebenes Polynom $G(x)$ n-ten Grades lauten die Koeffizienten $(g_0, \ldots, g_n)$, sodass $G(x) = \sum_{i=0}^{n} g_i x^i$.

```
1: keygen(pk, sk)
2: {
3:     F(x) = x^n + 1                          // monisch, irreduzibel, Grad n
4:     do {
5:         G(x) =zufälliges Polynom in B^even_{∞,n}(μ)
6:         ++ g_0                               // konst. Koeff. ungerade
7:         p = fmpz_poly_resultant(G(x), F(x))
8:     } while p ist nicht prim
9:     D(x) = F_mpz_mod_poly_gcd_euclidean(G(x), F(x))
10:    α = −d_0                                 // α = Wurzel von D(x)
11:    (r, Z(x), t) = fmpz_poly_xgcd(G(x), F(x))
12:    pk.p = p; pk.α = α                       // pk, sk sind einfache structs
13:    sk.p = p; sk.B = z_0 mod 2p
14: }
```

Erläuterung

Zeile 3 $x^n + 1$ wird als Polynom gewählt, da es keine Wurzel in $\mathbb{Z}[x]$ besitzt

Zeile 5 [55] definiert

$$\mathcal{B}_{\infty,n}(r) := \left\{ \sum_{i=0}^{n-1} a_i x^i : a_i \in [-r, r] \right\} \quad .$$

Analog dazu wird definiert

$$\mathcal{B}_{\infty,n}^{\text{even}}(r) := \left\{ \sum_{i=0}^{n-1} 2a_i x^i : a_i \in \left[-\frac{r}{2}, \frac{r}{2} \right] \right\} \quad .$$

Im Algorithmus werden zufällige Koeffizienten in $[-\frac{\mu}{2}, \frac{\mu}{2}]$ gewählt und verdoppelt.

Zeile 8 der Miller-Rabin Primzahltest wird verwendet. $p \neq 0$ prim impliziert dass $F(x)$ und $G(x)$ teilerfremd sind und dass $G(x)$ irreduzibel ist (weil bereits $F(x)$ irreduzibel ist). Somit generiert $G(x)$ ein Hauptideal $\mathfrak{p}$ in $\mathbb{Z}[x]/F(x)$.

Zeile 9 der hier verwendete GCD-Algorithmus ist eine Variation von Euklids Algorithmus über Polynomen modulo p. Hier wird die zweielementige Repräsentation $\langle p, x - \alpha \rangle$ von $\mathfrak{p}$ mit der Norm p von $\mathfrak{p}$ und α als Wurzel von $F(x) \mod p$. Die Wurzel von $D(x)$ ist somit auch Wurzel von $F(x)$ und $G(x)$.

Zeile 11 der erweiterte GCD-Algorithmus erzeugt $Z(x) \cdot G(x) = p \mod F(x)$. Hier wird ein geheimer Subschlüssel generiert. Die Entschlüsselung verlangt die Berechnung von $\frac{1}{G(x)} = \frac{Z(x)}{p}$. In der konkreten Implementierung wird ganzzahlig gerundet, daher ist nur z_0 für den Subschlüssel relevant.

Encrypt, Decrypt, Add, Mult

Der Code für encrypt, decrypt, add und mult gibt im Wesentlichen den Pseudocode in [55] wieder, da sich die Operationen recht gut auf GMP-Funktionen abbilden lassen.

Die Verschlüsselungsfunktion generiert ein Polynom $C(x)$, bei dem die Parität des konstanten Gliedes der Parität der zu verschlüsselnden Nachricht entspricht. Zur Transformation in den Chiffreraum wird Gl. 3.1 verwendet. Die Berechnung lautet $C(\alpha) \mod p$ mit fmpz_poly_evaluate().

Bei einer Beschränkung der Eingaben durch b_1 und b_2 ist die Ausgabe durch $b_1 + b_2$ beschränkt, da die Koeffizienten addiert werden. Die Schranke nach einer Multiplikation lautet $b_1 + b_2 + b_1 \cdot b_2$. Im Folgenden wird untersucht, wieviele Multiplikationen möglich sind, d. h. der Chiffretext kann noch korrekt ent-

schlüsselt werden. Nach d Multiplikationen ist das Ergebnis beschränkt durch $\mu^{2^d} = 4^{2^d}$.

$$4^{2^d} = \frac{2^{384}}{16} \Leftrightarrow 2^d = 190 \Leftrightarrow d = \lfloor log_2 190 \rfloor = 7$$

Das ist für die Operation recrypt() ausreichend und lässt Raum für einige darüber hinausgehende homomorphe Anwendungsoperationen.

3.5.2 Das unbegrenzt homomorphe Schema

Dieser Abschnitt beschreibt eine Version des Entschlüsselungsalgorithmus' nur aus XOR und AND Gattern bestehend. Auf diese Weise kann die Entschlüsselung in verschlüsselter Form auf einen Chiffretext angewendet werden, sodass letztendlich ein neuer Chiffretext mit geringerem Rauschen generiert wird. Damit diese *Recrypt*-Operation effektiv ausgeführt werden kann, muss die Schaltkreistiefe dieser Funktion flach genug sein, um nicht selbst bezogen auf das Rauschen *überzulaufen*. Dieser Abschnitt enthält eine neue Rundungsfunktion, die diesem Anspruch Rechnung trägt.

Erweiterte Schlüsselerzeugung

Zusätzlich zu den Parametern B, α und p, die in der begrenzt homomorphen Version von keygen erzeugt wurden, wird nun ein verschlüsselter Entschlüsselungshinweis generiert, der helfen soll, den Entschlüsselungsschaltkreis klein zu halten. Der Hinweis wird konstruiert als $\{\mathfrak{c}_i, B_i\}_{i=1}^{s_1}$ sodass $\sum_{i=1}^{s_1} \mathsf{decrypt}(\mathfrak{c}_i) B_i = B$. Da die Sicherheit dieses Vorgehens auf dem Verstecken des Parameters B in den additiven Komponenten von B_i basiert, ist die Größe des Feldes B_i natürlich von zentraler Bedeutung. Tatsächlich kann die Sicherheit des Hinweises auf das *Subset-Sum Problem* reduziert werden.

```
1: keygen()                                          // Fortsetzung ...
2: B* = ⌊B / S2⌋                                     // Schritt 1: verteilen
3: for (i = 0; i < S2; i++) {
4:     pk.B_i = B*
5:     pk.c_i = 1
6: }
7: for (i = S2; i < S1; i++) {
```

```
 8:     pk.B_i = random in [−p, p]
 9:     pk.c_i = 0
10: }
11: for (i = 0; i < S_2; i++) {                    // Schritt 2: zufällig add./sub.
12:     r_+ = random in [0, p − 1]
13:     r_− = −r_+
14:     add and subtract r_+ and r_− from random B_j
15: }
16: for (i = 0; i < S_1; i++ {                     // Schritt 3: mischen
17:     j = random in [0, S_1 − 1]
18:     swap pk.B_i and pk.B_j
19:     swap pk.c_i and pk.c_j
20: }
```

$pk.B$	$\lfloor \frac{B}{4} \rfloor$	$\ldots$	$\lfloor \frac{B}{4} \rfloor$	$B - 4\lfloor \frac{B}{4} \rfloor$	rand	$\ldots$	rand
$pk.\mathfrak{c}$	0		0	0	1		1

Tabelle 3.1: Initiale Verteilung des Hinweises

Hinweise:

Schritt 1 Die Verteilung des Hinweises startet mit einem Feld wie in Tabelle 3.1 beschrieben.

Schritt 2 Zufallswerte werden auf bzw. von den ersten S_2 Werten des Feldes addiert und subtrahiert. In jedem Schritt gilt die Invariante $\sum_{i=1}^{s_1} \mathsf{decrypt}(\mathfrak{c}_i) B_i = B$.

Schritt 3 Um die Hinweisverteilung zu randomisieren, wird das Feld gemischt.

Recrypt

Der Zweck der Recrypt-Operation ist es, ausgehend von einem *verrauschten* Chiffretext einen äquivalenten Chiffretext mit geringerem Rauschen zu generieren. Das Rauschen wird mit jeder homomorphen Additions- oder Multiplikationsoperation erhöht, bis letztendlich eine korrekte Entschlüsselung nicht mehr

möglich ist. Ohne die Recrypt-Operation können nur arithmetische Sequenzen von beschränkter Tiefe berechnet werden (beschränkt homomorph).

Die grundlegende Idee bei der Recrypt-Operation ist die Entschlüsselung des Chiffretextes im Chiffreraum selbst unter Anwendung homomorpher Operationen. Zur Erinnerung: die Entschlüsselung berechnet folgendes:

$$\left(c - \left\lfloor B\frac{c}{p}\right\rceil\right) \equiv_2 \left(c - \left\lfloor \sum_i \mathfrak{c}_i B_i \frac{c}{p}\right\rceil\right) \equiv_2 \left(c - \left\lfloor \sum_i \mathfrak{c}_i (B_i \cdot c \mod 2p)/p\right\rceil\right)$$

Die rechte Seite der Gleichung kann unter Zuhilfenahme des öffentlichen Schlüssels und des Hinweises berechnet werden, der in Abschnitt 3.5.2 konstruiert wurde. Aufgrund der Tatsache, dass die $\mathfrak{c}_i$ verschlüsselt sind, ist auch das Ergebnis der Berechnung verschlüsselt. Das Ergebnis ist aber eine *entrauschte* Repräsentation von c.

```
1: for (i = 0; i < S_1; i++) {
2:     d = (B_i · c mod 2p)                          // d ∈ [0, 2)
3:     for (j = 0; j < T; j++) {
4:         C_ij = encrypt(⌊d⌋) · c_i mod p
5:         d = (d − ⌊d⌋) · 2                         // Konvertierung zur Basis 2
6:     }
7: }
```

Nach der Anwendung des o. g. Algorithmus' enthält jede Zeile der Matrix (C_{ij}) eine binäre Repräsentation von $(B_i \cdot c \mod 2p)$ mit $T-1$ Bits Präzision im Chiffretext. Als Optimierung in Zeile 4 kann geprüft werden, ob der Term $\lfloor d \rfloor$ den Wert 0 ergibt und dann entweder $\mathfrak{c}_i$ oder $\mathsf{encrypt}(0)$ anwenden. Dadurch kann eine Multiplikation eingespart werden, und gleichzeitig entsteht ein weniger verrauschter Chiffretext. Als nächstes werden die einzelnen Zeilen mit einem möglichst flachen Schaltkreis addiert. Zu diesem Zweck werden die Additionen in einzelne Schritte aufgespalten:

1. Berechnung der Hamming-Gewichte der einzelnen Zeilen

   Die Berechnung der Hamming-Gewichte nutzt (elementare) symmetrische Polynome, wie in [55] vorgeschlagen. Mit diesem Ansatz können wesentlich kleinere Addierwerke erzeugt werden, als mit ausschließlicher An-

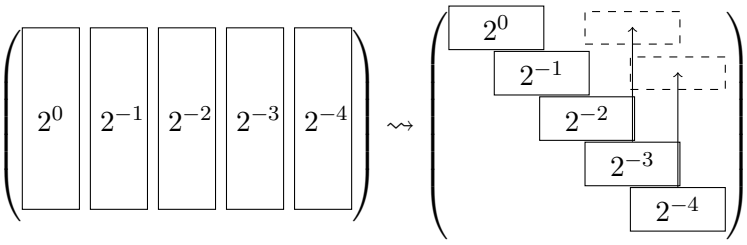


Abbildung 3.2: Schieben und Vereinigen der Zeilen

wendung von Halb- und Volladdierern. Das Polynom $e_k(X_1, \ldots, X_n) = \sum_{1 \leq j_1 < j_2 < \ldots < j_k \leq n} X_{j_1} \ldots X_{j_k}$ gibt direkt das k-te Bit des Hamming-Gewichtes der Eingabe an. Der Pseudo-Code für den Algorithmus ist in [55, p. 15] angegeben.

2. Schieben und Vereinigen der Hamming-Gewichte gemäß der Wertigkeit der Spalte

   Nach der Berechnung der Hamming-Gewichte kann mit den Werten wie in Abbildung 3.5.2 weiter verfahren werden. Zu beachten ist, dass hier keine Schaltgatter erforderlich sind, weil hier nur *umverdrahtet* wird.

3. Anwendung eines Carry-Save-Addierwerkes bis nur noch zwei Zeilen übrig sind

   Der Carry-Save-Adder ist ein Volladdierer, der auf drei Eingaben gleichzeitig angewendet wird. Dadurch kann eine konstante Schaltkreistiefe erzielt werden.

4. Anwendung eines Ripple-Carry-Addierwerkes für die letzte Addition

   Die abschließende Addition mit einem Ripple-Carry-Adder besitzt eine zur Eingabegröße lineare Schaltkreistiefe.

Nach diesem Schritt ist die Matrix auf eine einzige Zeile reduziert worden. Diese enthält eine Repräsentation von $\sum_i \mathfrak{c}_i(B_i \cdot c \mod 2p)/p$ in Festkommadarstellung.

Eine flache Rundungsfunktion

Um im Rahmen der Recrypt-Operation eine mit weniger Anfangsrauschen versehene Verschlüsselung der Eingabe zu erzeugen, schlagen Smart et al. [55] eine

Rundung anhand der beiden niedrigstwertigen Nachkommabitstellen vor. Dies führte jedoch bei einer ersten Implementierung zu einem Schaltkreis, der zur korrekten Ausführung eine zu große Tiefe besaß. Ein neuer Vorschlag, dessen Wirksamkeit in dieser Arbeit nachgewiesen wird, ist die Rundung anhand des geringstwertigen Bits. Durch diese Maßnahme wird die Schaltkreistiefe verringert und führt zu einer tatsächlichen Einsparung von zwei Multiplikationen. Ein Vergleich der beiden Verfahren wird in Tabelle 3.2 zusammengefasst.

$e_0.e_1e_2$	dezimal	eff. Rundung	Rundung in [55]
0.00	0.00	0	0
0.01	0.25	1	0
0.10	0.50	1	1
0.11	0.75	1	1
1.00	1.00	1	1
1.01	1.25	0	1
1.10	1.50	0	0
1.11	1.75	0	0

Tabelle 3.2: Vergleich der Rundungsverfahren

Die Ausgabe des gesamten Algorithmus wird angegeben durch:

$$\left(c - \left\lfloor \sum_i \mathfrak{c}_i (B_i \cdot c \mod 2p)/p \right\rceil \right) = (c + e_0 + e_1) \mod 2 \quad .$$

Die Modifikation der Rundungsfunktion kann nur Auswirkungen auf die Sicherheit des Gesamtsystems haben, die jedermann im Besitz des öffentlichen Schlüssels herbeiführen kann. Da die Rundungsfunktion ausschließlich im Rahmen der Recrypt-Operation angewendet und diese nur in Verbindung mit dem öffentlichen Schlüssel ausgeführt wird, würde die Konstruktion eines wirksamen Angriffs über die modifizierte Rundungsfunktion die Sicherheit des Kryptosystems verletzen und könnte auf eine effiziente Lösung des zugrundeliegenden *Closest-Vector-Problems* über Gitteridealen reduziert werden.

Fazit Die in dieser Arbeit vorgestellte Implementierung ist ein Schritt dahingehend, die homomorphe Kryptografie einer breiteren Masse von Anwendern

zugänglich zu machen, sodass weitere Fortschritte im praktischen Bereich ermöglicht werden. Außerdem dient sie als Grundlage für weitere Umsetzungen verwandter Kryptosysteme, da zu erwarten ist, dass einzelne Ansätze übernommen werden können.

3.6 Homomorph verschlüsselte Funktionen

Der folgende Abschnitt behandelt die Verwendung des vorgestellten Krypto-Schemas $\mathfrak{H}$ aus Abschnitt 3.3 für die Lösung realer Probleme. Um dies zu verdeutlichen, werden neue Lösungen für bekannte Verfahren und Protokolle vorgestellt, die sich durch den Einsatz der homomorphen Kryptografie elegant lösen lassen. Dabei werden zunächst Lösungen für elementare Protokolle und Algorithmen vorgestellt, die dann im weiteren Verlauf zur Lösung weitaus komplexerer Probleme zusammengeführt werden.

3.6.1 Homomorph verschlüsselte Beispielalgorithmen

Zur Formalisierung eines Kryptosystems, das Operationen auf den Chiffretexten gestattet, wird zunächst eine Notation dieser Operationen benötigt. In den meisten Fällen wird auf Chiffretexten operiert, die verschlüsselte Binärwerte im Sinne der booleschen Schaltalgebra darstellen. Daher werden auch die verschlüsselt ausgeführten Funktionen als boolesche Schaltkreise aufgefasst, die in Vollmer [59] als endliche und gerichtete azyklische Graphen definiert werden. Dabei ist jeder Knoten des Graphen entweder Eingabe, Ausgabe oder boolesche Funktion der Basis B. In dieser Arbeit ist die Basis $B = \{\wedge, \oplus\}$ mit Fan-In 2 bei beiden Funktionen. Für jeden Schaltkreis C sei f_C die boolesche Funktion, die durch C berechnet wird.

Definition 3.6.1. Ein *homomorphes Kryptoschema* $\mathcal{S}$ ist ein Tupel von Funktionen

$$
\begin{aligned}
(\mathsf{KeyGen}_{\mathcal{S}}() &\mapsto (pk, sk),\\
\mathsf{Encrypt}_{\mathcal{S}}(m, pk) &\mapsto \mathfrak{c},\\
\mathsf{Decrypt}_{\mathcal{S}}(\mathfrak{c}, sk) &\mapsto m,\\
\mathsf{Evaluate}_{\mathcal{S}}(\mathfrak{c}_i, C, pk) &\mapsto \mathfrak{c}')
\end{aligned}
$$

mit den folgenden Eigenschaften:

1. *Korrektheit von Ver- und Entschlüsselung:* $\mathsf{Decrypt}_{\mathcal{S}}(\mathsf{Encrypt}_{\mathcal{S}}(m, pk), sk) = m$ für alle Ausgaben von $\mathsf{KeyGen}()_{\mathcal{S}}$.

2. *Korrektheit der homomorphen Eigenschaft:*

$$\mathsf{Decrypt}_{\mathcal{S}}\big(\mathsf{Evaluate}_{\mathcal{S}}((\mathsf{Encrypt}_{\mathcal{S}}(m_0, pk), \ldots, \mathsf{Encrypt}_{\mathcal{S}}(m_n, pk)), C, pk), sk\big) = f_C(m_0, \ldots, m_n)$$

   für alle Eingaben in $\mathsf{KeyGen}_{\mathcal{S}}()$ und alle booleschen Schaltkreise C mit $|C| \leq n$. Gibt es keine Begrenzung n, so handelt es sich um ein *unbegrenzt* homomorphes (fully homomorphic) Schema.

Unter der Annahme, dass es sich bei den Chiffretexten um verschlüsselte Bitwerte handelt, können die booleschen Operationen $\wedge$ (AND) und $\oplus$ (XOR) durch die arithmetischen Operationen $\cdot$ und $+$ ersetzt werden, wie Tabelle 3.3 zeigt.

(a) "$+ \mod 2$", "$\oplus$"

	0	**1**
0	0	1
1	1	0

(b) "$\cdot \mod 2$", "$\wedge$"

	0	**1**
0	0	0
1	0	1

Tabelle 3.3: Überführung boolescher Operationen in $+$ und $\cdot$

3.6.2 Einfache Protokolle

Als *einfache Protokolle* sind Methoden zu verstehen, die bereits ohne den Einsatz homomorpher Kryptografie implementiert worden sind und die in der Fachliteratur aufgrund ihrer verständlichen Struktur gern als Stellvertreterprobleme herangezogen werden. Dabei sind die einfachen Protokolle stets die Lösung für ein sehr konkretes Szenario und gewinnen in der Abstraktion an Relevanz für ein weites Spektrum an Problemstellungen.

k-Bit Oblivious Transfer

Das Oblivious Transfer Protokoll dient zum Austausch von Daten, wobei geheim bleiben soll, welches Element ausgewählt wurde. Ebenso sollen die nicht

ausgewählten Elemente geheim bleiben. Mit dem homomorphen Schema $\mathfrak{H}$ kann dieses Problem leicht für Elemente beliebiger Länge in einer einzigen Interaktionsrunde gelöst werden. Das Protokoll gliedert sich wie folgt:

1. Alice hat ein $a \in \{0,1\}$ und den geheimen Schlüssel SK
2. Bob hat $m_{0,1} \in \{0,1\}^k$, zwei k-Bit Elemente
3. Alice wählt zufällig eine η-Bit Zahl $t \in \mathbb{N}^+$ sodass $t \equiv a \ mod\ 2$ und verschlüsselt $a' \leftarrow E(t)_{SK}$
4. Bob wählt zufällig η-Bit Zahlen $m'_{i_j} \in \mathbb{N}^+$ mit $m'_{i_j} \equiv m_{i_j} \ mod\ 2$ für $i \in \{0,1\}$ und $0 \leq j \leq (k-1)$. Er berechnet $s_j \leftarrow ((m_{0_j} * (a'+1)) + (m_{1_j} * a'))$ für $0 \leq j \leq (k-1)$
5. Alice entschlüsselt das gewählte Element durch $s_j \leftarrow D(s'_j)_{SK} \ mod\ 2$ für $0 \leq j \leq (k-1)$

Die multiplikative Tiefe dieses Protokolls ist 1. Bob ermittelt das für ihn unbekannte Ergebnis der Auswahl durch Anwendung eines binären Selektors über den beiden Elementen. Dazu wird das Argument von Alice als 1-Bit Adresse verwendet. Bob berechnet $s \leftarrow (m_0 \wedge \neg a) + (m_1 \wedge a)$ durch Auswertung von logischen AND-Operationen. Um solche booleschen Terme mit einem homomorphen Schema auswerten zu können, müssen sie zunächst in eine arithmetisierte Repräsentation überführt werden. Dabei wird die Äquivalenz zwischen booleschen Operationen und arithmetischen Operationen auf einzelnen Bits ausgenutzt: die logische XOR-Operation entspricht der übertragslosen Addition, die logische AND-Operation kann durch eine Multiplikation abgebildet werden. Ein logisches OR lässt sich als zusammengesetzte Operation ausdrücken: $a \vee b = (a \oplus b) \oplus (a \wedge b) \Rightarrow a \vee b \rightsquigarrow ((a+b)+(a*b)) \ mod\ 2$. Siehe hierzu auch Kapitel 5.

Geheime Auswahl

Die Technik des Oblivious Transfer Protokolls kann verallgemeinert werden und zu einer geheimen, index-basierten Auswahl aus n Elementen erweitert werden. In einem solchen Szenario gibt es also mehr als zwei Elemente zur

Auswahl, sodass auch die *Adresse* bzw. der Index des auszuwählenden Elements größer ist, nämlich $log_2 n$ Bits für eine Auswahl aus n Elementen. Das Protokoll gleicht dem des Oblivious Transfers, der Algorithmus ähnelt einem binär adressierten Speicherzugriff, wie in Abbildung 3.3 dargestellt.

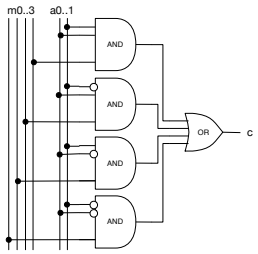


Abbildung 3.3: Speicherschaltkreis

Die Schaltfunktion, welche eine Speicherzelle $m \in \{m_0, m_1, m_2, m_3\}$ mit einer Adresse $a \in \{(00), (01), (10), (11)\}$ auswählt, lautet $m = (\neg a_0 \wedge \neg a_1 \wedge m_0) \vee (a_0 \wedge \neg a_1 \wedge m_1) \vee (\neg a_0 \wedge a_1 \wedge m_2) \vee (a_0 \wedge a_1 \wedge m_3)$. Das Protokoll zur verallgemeinerten, geheimen Auswahl lautet:

1. Bob besitzt 4 Elemente $m_0..m_3 \in \{0, 1\}$
2. Alice hat ein Argument $a \in \{0, 1\}^2$ und den geheimen Schlüssel SK
3. Alice wählt zufällige η-Bit Zahlen $t_i \in \mathbb{N}^+$ sodass $t_i \equiv a_i \; mod \; 2$ für $0 \leq i \leq 2$ und verschlüsselt jedes $a'_i \leftarrow E(t_i)_{SK}$
4. Bob wählt zufällige η-Bit Zahlen $m'_i \in \mathbb{N}^+$ sodass $m'_i \equiv m_i \; mod \; 2$ für $0 \leq i \leq 3$. Er berechnet $s' \leftarrow ((a'_0 + 1) * (a'_1 + 1) * m'_0) + (a'_0 * (a'_1 + 1) * m'_1) + ((a'_0 + 1) * a'_1 * m'_2) + (a'_0 * a'_1 * m'_3)$
5. Alice entschlüsselt das ausgewählte Element mit $s \leftarrow D(s')_{SK}$.

Als Optimierung kombiniert Bob die Zwischenergebnisse der einzelnen Speicherzeilen (z. B. $\neg a_0 * a_1 * m_2$) nicht multiplikativ, sondern durch Additionen. Folgende Annahme liegt dem zugrunde: bei Verwendung einer $log_2 n$ Adresse für n Elemente wird immer genau ein Element ausgewählt, d. h. das Zeilenergebnis für dieses Element ist 1, sofern die gewählte Speicherzelle eine 1 enthält, alle unselektierten Zeilen ergeben 0. Anstatt $(\neg a_0 * \neg a_1 * m_0) \vee (a_0 * \neg a_1 * m_1) * ...$

rechnet er also $(\neg a_0 * \neg a_1 * m_0) + (a_0 * \neg a_1 * m_1) * ...$, wodurch je Element eine Multiplikation eingespart werden kann. Das Protokoll hat dadurch eine multiplikative Tiefe von $log_2 n$ für n Elemente und kann auf k-Bit Elemente erweitert werden, indem k Instanzen parallel eingesetzt werden.

Geheimer Vergleich

Eine spezielle Form des geheimen Vergleichs ist das *Millionärsproblem*: zwei Millionäre, Alice und Bob, möchten ermitteln, welcher der Reichere von beiden ist. Sie möchten aber nicht die Höhe ihres jeweiligen Vermögens offenlegen, sondern wollen dies geheimhalten. Mit Hilfe des homomorphen Schemas $\mathfrak{H}$ kann dieses Problem durch Anwendung binärer Addierer gelöst werden:

1. Alice hat $a \in \{0,1\}^k$, eine binäre k-Bit Little-Endian Repräsentation ihrer Vermögenssumme in $a_0..a_{k-2}$, $a_{k-1} \leftarrow 0$ und den geheimen Schlüssel SK.

2. Bob hat $b \in \{0,1\}^k$, analog zu a, $b_{k-1} \leftarrow 0$

3. Alice wählt zufällige η-Bit Zahlen $t_i \in \mathbb{N}^+$ sodass $t_i \equiv a_i \ mod\ 2$ und verschlüsselt $a'_i \leftarrow E(t_i)_{SK}$ für $0 \leq i \leq (k-1)$

4. Bob wählt zufällige η-Bit Zahlen $b'_i \in \mathbb{N}^+$ sodass $b'_i \equiv b_i \ mod\ 2$ für $0 \leq i \leq (k-1)$. Er berechnet $b'_i \leftarrow b'_i + 1$ für $0 \leq i \leq (k-1)$, wählt eine zufällige η-Bit Zahl $t' \in \mathbb{N}^+$ mit $t' \equiv 1 \ mod\ 2$, berechnet $u'_0 \leftarrow b'_0 + t'$ und $c'_0 \leftarrow b'_0 * t'$, ferner $u'_i \leftarrow b'_i + c'_{i-1}, c'_i \leftarrow b'_i * c_{i-1}$ für $1 \leq i \leq (k-1)$. Er hat nun das binäre Zweierkomplement von b' in u'. Er berechnet $s'_0 \leftarrow a'_0 + u'_0$ und $c'_0 \leftarrow a'_0 * u'_0$. Schließlich berechnet er $s'_i \leftarrow a'_i + u'_i + c'_{i-1}$ und $c_i \leftarrow (a'_i * u'_i) + (c'_{i-1} * (a'_i + u'_i))$ für $1 \leq i \leq (k-1)$.

5. Alice entschlüsselt $s \leftarrow D(s'_{k-1})_{SK}$ und schließt für beide Parteien, dass $a < b$ wenn $s \equiv 1 \ mod\ 2$ und andernfalls $a \geq b$.

Alice generiert zunächst eine binäre Repräsentation ihres Arguments a und verschlüsselt die einzelnen Bits. Bob generiert ebenso eine binäre Repräsentation seines Arguments b und wandelt sie in eine Reihe von Ganzzahlen b'_i

mit gleicher Parität, wie die einzelnen Bits von b. Er berechnet das Zweierkomplement (die numerische Negation) seines Arguments und addiert das verschlüsselte Argument von Alice mit einem binären Addierer. Er berechnet somit $a' - b'$. Das höchstwertige Bit des Ergebnisses gibt das Vorzeichen an, und Alice schließt bei einem positiven Vorzeichen darauf, dass ihr Vermögen größer oder gleich dem von Bob ist. Bei einem negativen Vorzeichen ist das Vermögen von Bob größer.

Es ist zu beachten, dass Bob sein Argument zwar in Ganzzahlen gleicher Parität codiert, jedoch unverschlüsselt in die Berechnung einbringt. Es entfällt also die asymmetrische Verschlüsselung der Argumente mit einem öffentlichen Schlüssel. Das zugrundeliegende mathematische Prinzip wird in Abschnitt 3.3.2 erläutert. Die multiplikative Tiefe des Protokolls ist 2.

3.7 Ergebnisse

Das zurückliegende Kapitel führt in die homomorphe Kryptografie ein und beschreibt mehrere begrenzt und unbegrenzt homomorphe Kryptosysteme. Es wird ein Konzept vorgestellt, mit dem sich digitale Schaltungen und damit beliebige Algorithmen mit homomorpher Kryptografie verschlüsselt berechnen lassen. Auf Basis dieser Grundlage werden einfache Algorithmen und Protokolle in homomorph verschlüsselter Ausprägung entwickelt, die einzelne Probleme, wie den Oblivious Transfer, eine geheime Auswahl aus vielen Elementen oder einen geheimen Vergleich lösen. Diese Basisalgorithmen lassen sich kombiniert zur Lösung komplexerer Probleme anwenden.

Kapitel 4

Homomorph verschlüsselte Suche

In diesem Kapitel wird ein Ansatz zum Suchen auf verschlüsselten Daten unter Anwendung von homomorph verschlüsselten Schaltkreisen mit konstanter Tiefe vorgestellt. Abbildung 4.1 zeigt eine Skizze der Architektur. Die verschlüsselte Suche gehört zu den gängigsten Anwendungsfällen vertraulicher Operationen in verteilten Systemen. Die Suche wird in diesem Kapitel in zwei Ausprägungen betrachtet: zum einen die exakte Suche, d. h. der Suchbegriff ist vollständig und in exakter Abbildung im Suchraum enthalten und der *Fuzzy*-Suche, wobei der Suchbegriff mit gewissen (einstellbaren) Toleranzen im Suchraum Treffer erzeugt.

Die in diesem Kapitel betrachteten Ansätze und Protokolle wurden in [11] wissenschaftlich publiziert.

4.1 Grundlegendes Vorgehen

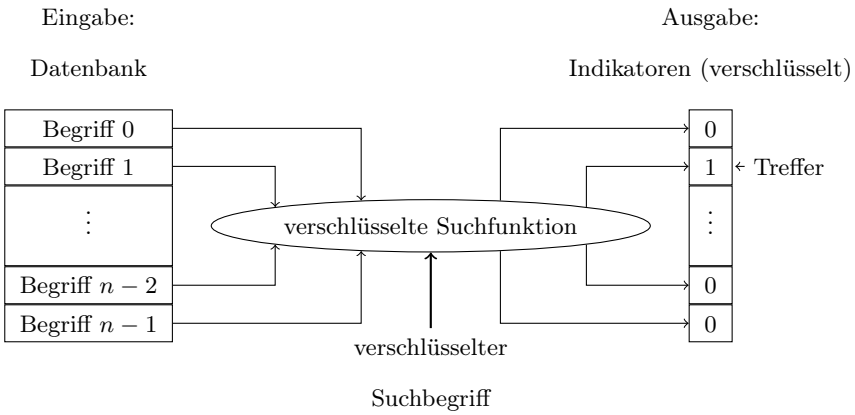


Abbildung 4.1: Generelles Schema für eine Wörterbuchsuche

Die folgenden Komponenten sind Teil der Versuchsanordnung für die Suche:

- Die *Eingabe* ist eine Liste von Wörtern oder ein Datenstrom
- Die *verschlüsselte Suchfunktion* wird als homomorph verschlüsselter Schaltkreis ausgeführt. Dieser wird für jedes Eingabewort ausgeführt und berechnet ein Element des Ausgabevektors.
- Der *verschlüsselte Suchbegriff* dient als Eingabe in den verschlüsselten Suchschaltkreis und ist mit dem öffentlichen Schlüssel verschlüsselt.
- Der *verschlüsselte Ausgabevektor* besitzt an den Stellen, an denen Übereinstimmungen zwischen Suchbegriff und Eingabewort bestehen, eine verschlüsselte 1, an allen anderen Stellen eine verschlüsselte 0. Bei einer unexakten Fuzzy-Suche enthält der Ausgabevektor eine Übereinstimmungsbewertung.

Eine Zelle des Ausgabevektors kann formal als Rückgabewert einer Funktion von Eingabewort (word) und Suchbegriff (searchterm) angegeben werden:

$$\varphi : \begin{array}{c} \Sigma^* \times \mathfrak{C}^* \to \mathfrak{C}^* \\ (word, searchterm) \mapsto \mathcal{C}(word, searchterm) \end{array} \tag{4.1}$$

Weil die Definition von φ unabhängig von der Größe der Datenbank und den darin enthaltenen Wörtern ist, kann der Ausgabevektor effizient parallel berechnet werden. Ein weiterer Vorteil ist, dass die Suchfunktion $\mathcal{C}$ nur vom verschlüsselten Suchbegriff abhängt. Insbesondere ist die Schaltkreistiefe der Funktion also unabhängig von der Größe der Datenbank und im Bezug auf die Eingabewortgröße konstant.

Auf der anderen Seite kann der Ausgabevektor sehr groß werden, sofern die Eingabemenge auch groß ist. Tatsächlich besteht eine lineare Abhängigkeit zwischen der Größe der Eingabedatenbank und der Größe des Ausgabevektors, da dieser auf der Anwendung der Suchfunktion für jedes Eingabewort basiert. Unter der Voraussetzung einer strengen Totalordnung, bei der für die Elemente der Eingabemenge M stets die folgende Ordnung $\forall a, b \in M : a < b \ \vee \ a > b$ gilt, kann gezeigt werden, dass die Platzkomplexität des Ausgabevektors von $\mathcal{O}(n)$ bei einer Datenbankgröße von n auf $\mathcal{O}(1)$ reduziert werden kann. Der dazu erforderliche Schaltkreis hat eine lediglich um 1 erhöhte multiplikative Tiefe.

4.2 Exakte Suche

Aufbauend auf dem generellen Ansatz für verschlüsselte Suche kann nun eine Funktion formuliert werden, welche die exakte Suche durchführt. Der die Funktion implementierende Schaltkreis sollte die folgenden Eigenschaften besitzen:

$$\varphi(word, searchterm) = \begin{cases} 1 & word == searchterm \\ 0 & \text{else} \end{cases} \tag{4.2}$$

Dies kann in einen zeichenbasierten Vergleich überführt werden

$$\varphi(word, searchterm) = \bigwedge_{i=0}^{|searchterm|-1} (word_i =_c searchterm_i) \tag{4.3}$$

Zwei Punkte bedürfen noch einer genaueren Spezifikation:

1. die konkrete Implementierung der zeichenbasierten Vergleichsfunktion $=_c$ und
2. der (häufige) Fall $|searchterm| \neq |word|$.

4.2.1 Vergleich auf Zeichenebene

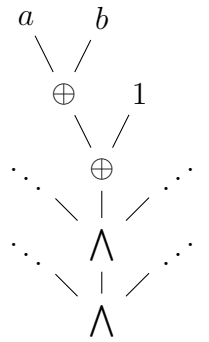
Sei Σ ein endliches Alphabet sowohl für die in der Datenbank enthaltenen Wörter als auch für die Suchbegriffe mit einer strengen Totalordnung $<$ über Σ. Im Falle von $\Sigma = \{a, \ldots, z, A, \ldots, Z\}$ sei dies die Sortierfolge der Datenbankworte. Die bijektive Funktion $bin(\sigma)$ berechnet die Position von σ und ist definiert als

$$\begin{aligned} & \Sigma \to \{0,1\}^{\lceil \log_2 \Sigma \rceil} \\ bin: \quad & \sigma \mapsto \begin{cases} 0 \; if \; \sigma = \min(\Sigma) \\ \left(\max_{\sigma' < \sigma} \{bin(\sigma')\} + 1 \right)_2 \text{ else.} \end{cases} \end{aligned} \tag{4.4}$$

Schließlich wird die Relation $=_c$ definiert als

$$=_c : \quad \left\{ \{\sigma_1, \sigma_2\} \middle| \bigwedge_{i=0}^{\lceil \log_2 \Sigma \rceil - 1} bin(\sigma_1)_i \oplus bin(\sigma_2)_i \oplus 1 \right\} \quad . \tag{4.5}$$

Die Tiefe des aus Gleichung (4.3) resultierenden booleschen Schaltkreises in Verbindung mit dem booleschen Schaltkreis $=_c$ aus Gleichung (4.5) ist



$$\left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \end{array}\right\} \begin{array}{l} 2 \\ \\ \lceil \log_2 \Sigma \rceil \text{ (eq. 4.5)} \\ \\ \lceil \log_2 |search| \rceil \text{ (eq. 4.3)} \end{array} \tag{4.6}$$

$$\Rightarrow \text{depth}(\varphi(word, search)) =$$
$$2 + \lceil \log_2 \Sigma \rceil + \lceil \log_2 |search| \rceil \in \mathcal{O}(\log_2 n)$$

Es ist festzustellen, dass die durch die Funktion *bin* durchgeführte Berechnung lediglich dazu dient, ein Zeichen in seine binäre Repräsentation zu transformieren, und somit ist *bin* nicht als Teil der Suchfunktion anzusehen. Für eine effiziente Transformation kann statt der impliziten Funktion in Gleichung 4.4 besser eine tabellengestützte Transformation durchgeführt werden.

4.2.2 Auffüllen (Padding)

In Gleichung 4.3 wurde die Abhängigkeit $|word| \geq |searchterm|$ eingeführt. Generell können sich bei einer Suche die Längen der Suchwörter und der Datenbankeinträge durchaus unterscheiden.

Definition 4.2.1. Für ein Alphabet Σ und ein Füllzeichen (padding character) $\square$ mit $\Sigma \cap \square = \emptyset$ sei das *Alphabet mit Padding* $\widetilde{\Sigma} = \Sigma \cup \{\square\}$. $\square^n$ wird als Kurznotation für $\underbrace{\square \cdots \square}_{n \text{ mal}}$, eine n-Zeichen Auffüllung verwendet.

Als zusätzliche Sicherheitsmaßnahme kann das Padding verwendet werden, um die Länge eines Suchbegriffs zu verschleiern. Die Länge von *searchterm* ist eine öffentliche Information, auch wenn der Suchbegriff inhaltlich verschlüsselt ist. Außerdem kann auf die Länge eines Suchbegriffs wegen der bestehenden Abhängigkeit zur Größe der Suchfunktion geschlossen werden. Um also die Länge des Suchbegriffs zu verschleiern, kann der Suchanfragensteller vor der Verschlüsselung des Suchbegriffs diesen durch Anfügen einer zufälligen Anzahl von Füllzeichen erweitern.

Die Suche wird auf dem Server durch den folgenden Algorithmus geleistet:

```
1: function search(searchterm st, database db):
2:     length ← |st|
3:     for word in db {
4:         if (|word| ≤ length) {
5:             output ← φ(word + □^(length−|word|), st)
6:         } else {
7:             output ← Encrypt(0)
8:         }
9:     }
```

Trotz der Modifikation des Suchbegriffs durch Padding ist die Tiefe von φ noch immer im Bereich $\log n$, weil die Änderungen lediglich eine Konstante zu Σ und $|searchterm|$ in Gleichung 4.6 hinzufügen.

4.2.3 Reduktion der Ausgabegröße

Zwischen der Größe des Ausgabevektors und der Größe der durchsuchten Datenbank besteht eine lineare Abhängigkeit, somit ist die Platzkomplexität eines Ergebnisses einer Suchanfrage in einer n-elementigen Datenbank im Bereich $\mathcal{O}(n)$. Unter der Voraussetzung einer strikten Totalordnung der Datenbankelemente ergibt sich, dass eine Suchanfrage höchstens einen Treffer liefern kann - oder keinen, sofern der Suchbegriff nicht in der Datenbank enthalten ist. Die strikte Totalordnung bezieht sich dabei nicht auf eine (unabhängig davon mögliche) Sortierung der Datenbankelemente, sondern generell auf die Menge der in der Datenbank vorkommenden Begriffe, d. h. es wird auf dem eindeutigen Primärschlüssel der Datenbank gesucht. Dem Protokoll für einen geheimen Speicherzugriff in Kapitel 3.6 folgend, kann der reduzierte Ausgabevektor bestehend aus den Trefferindikatoren $I_0..I_n$ berechnet werden als

$$\sum_{i=0}^{n} I_i$$

Dies verringert die Platzkomplexität des Ausgabevektors auf $\mathcal{O}(1)$, das bedeutet auf ein einziges Element, welches eine verschlüsselte 1 enthält, sofern der angegebene Suchbegriff in der Datenbank gefunden wurde und eine ver-

schlüsselte 0, wenn dies nicht der Fall ist. Um die Position des Treffers in der Datenbank anzugeben, kann der Ausgabewert berechnet werden als

$$\sum_{i=0}^{n} I_i * i$$

Das Resultat ist dann ein $log_2 n$-Element, welches die verschlüsselte binäre Repräsentation des Treffer-Indexes enthält. Die Reduzierung auf einen verschlüsselten binären Index erhöht die multiplikative Tiefe der Suchfunktion um 1.

Reduktion auf eine festgelegte Ausgabegröße Um der Ausgabemenge des zuvor beschriebenen Konzeptes statt einer booleschen eine höhere Aussagekraft zu verleihen, wird im Folgenden angenommen, dass eine Suche maximal c Elemente aus der Treffermenge liefern soll, dass also c Elemente des Ergebnisvektors $\mathfrak{ind}$ auf den Suchbegriff zutreffen und mit 1 markiert sind. Das Ergebnis wird dann als Bitvektor der Länge $c \cdot s$ ausgedrückt. Dazu wird das Gesamtergebnis bei jedem Treffer (bei dem $\mathfrak{ind}_i = 1$) um s Bits verschoben.

Sei $\mathsf{hamming}(v)$ die Funktion, welche das Hamming-Gewicht von v berechnet (siehe [55, p. 15]). Die Funktion liefert als Ausgabe einen Bitvektor der Länge $\lceil \ln c + 1 \rceil$, wenn höchstens c Treffer in v markiert sind.

Die kumulative Summe von $\mathfrak{ind}$ wird dann als $|R| \times \lceil \ln c + 1 \rceil$-Matrix $(\mathfrak{sum}_{ij})$ erzeugt. Die folgende Funktion kommt dabei zum Einsatz:

$$\mathfrak{sum}_{ij} = \mathsf{hamming}(\mathfrak{ind}_{0..i})_j \wedge \mathfrak{ind}_i \tag{4.7}$$

Dann wird die $|R| \times c$-Matrix besetzt:

$$\mathfrak{mask}_{ij} = \begin{cases} 1 & \sum_{k=0}^{\lceil c \rceil} 2^k \cdot \mathfrak{sum}_{ik} = j \\ 0 & \text{else.} \end{cases} \tag{4.8}$$

Schließlich kann das Ergebnis als $c \times |R|$-Matrix $(\mathfrak{res}_{ij})$ aufgefasst werden, wobei gilt:

$$\mathfrak{res}_{ij} = \bigoplus_{k=0}^{|R|-1} \mathsf{encrypt}(R_k)_j \wedge \mathfrak{mask}_{ki}. \tag{4.9}$$

Abbildung 4.2 zeigt die Schritte für ein einfaches Beispiel[1].

[1] Zur besseren Lesbarkeit der Binärwerte sind die Spalten gespiegelt.

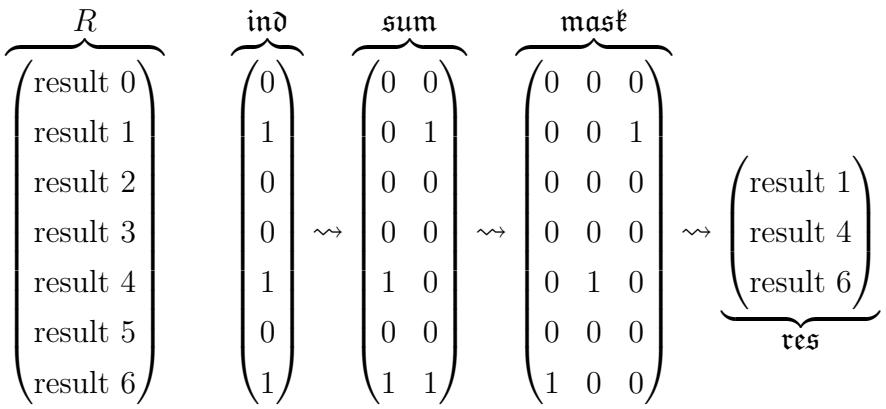


Abbildung 4.2: Schritte zur Ergebniskompression

4.3 Exakte Suche in Datenströmen

Mit einigen wenigen Modifikationen können der Algorithmus in Abschnitt 4.2.2 und die Funktion φ zur Suche von Worten in einem Datenstrom verwendet werden. Abbildung 4.3 zeigt eine schematische Darstellung dieses Ansatzes.

- Jede Position im Datenstrom entspricht einer Zelle im Ausgabevektor.
- Padding ist nicht mehr erforderlich und kann auch nicht mehr angewendet werden. Daher ist die Länge des Suchbegriffs öffentlich. Dies kann durch Einsatz der *Fuzzy*-Suche kompensiert werden, indem n Zeichen des Suchbegriffs durch Padding-Zeichen ersetzt werden und die Suche eine Toleranz von n Fehlern gestattet. Dies führt zu einer gewissermaßen exakten Suche mit geheimer Länge des Suchbegriffs.

4.4 *Fuzzy*-Suche

Während Abschnitt 4.2 eine grundsätzliche Konstruktion zur exakten Suche unter Verwendung einer homomorph verschlüsselten Suchfunktion einführte, erweitert dieser Abschnitt diesen Ansatz wesentlich dadurch, dass nun mehr als ein einziger Vergleich ($=_c$) der Zeichen durchgeführt wird. Dieses Vorgehen resultiert in der Fähigkeit, *unscharfe* (fuzzy) Suchergebnisse zu berechnen. Die Konstruktion der Fuzzy-Suche unterscheidet sich von der exakten Suche hauptsächlich in der Beschaffenheit der Elementarsuchfunktion φ.

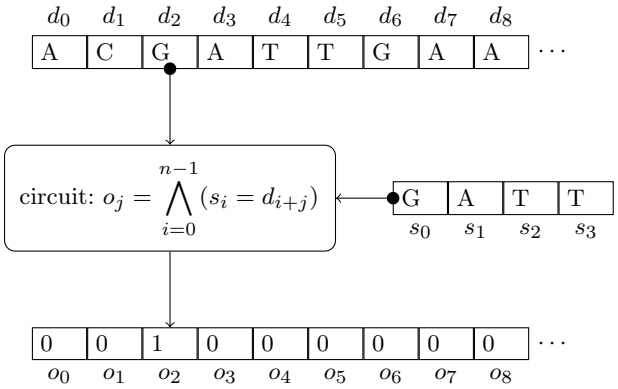


Abbildung 4.3: Suche in einem Datenstrom

4.4.1 Zählen der Abweichungen

Ein erster Ansatz zur Durchführung einer unscharfen Suche kann durch das Abzählen der Abweichungen zwischen dem Suchbegriff und einem gegebenen Begriff aus der durchsuchten Datenbank oder des durchsuchten Datenstroms erreicht werden. Zu diesem Zwecke wird die konjunktive Verknüpfung der zeichenbasierten Vergleichsoperationen in Gleichung 4.3 durch eine Summenfunktion ersetzt:

$$\varphi : \begin{array}{c} \Sigma^* \times \mathfrak{C}^s \to \mathfrak{C}^{\lceil \log_2 s \rceil} \\ (w, s) \mapsto \sum_{i=0}^{|s|-1} (w_i =_c s_i) \end{array} . \tag{4.10}$$

Da die gebildete Summe als Hamming-Gewicht eines Bitvektors ausgedrückt werden kann, führt die in [55, Seite 15] vorgeschlagene Anwendung von *symmetrischen Polynomen* zu einem sehr flachen booleschen Schaltkreis. Das k-te Bit des Hamming-Gewichts eines Vektors $(b_1, \ldots, b_n)$ kann dann direkt als Polynom angegeben werden:

$$e_{2^{k-1}}(b_1, \ldots, b_n) := \sum_{1 \le j_1 < j_2 \cdots j_{2^{k-1}} \le n} b_{j_1} \cdots b_{j_{2^{k-1}}} \quad .$$

Die Auswertung des Polynoms kann direkt in einen booleschen Schaltkreis überführt werden.

$$e_{2^{k-1}}(b_1, \ldots, b_n) := \bigoplus_{1 \le j_1 < j_2 \cdots j_{2^{k-1}} \le n} b_{j_1} \cdots \wedge \cdots b_{j_{2^{k-1}}} \quad .$$

Für den Fall dass die höchste Fehleranzahl err_{max} von vornherein bekannt ist, liegt der Vorteil in der Anwendung symmetrischer Polynome darin, dass die Anzahl der auszuwertenden Polynome auf die ersten $\log_2(err_{max})$ Ziffern des Ausgabevektors beschränkt werden kann.

4.4.2 *Weiche* Abweichungen und Ähnlichkeit

Einige Anwendungsgebiete der Fuzzy-Suche, wie das Zuordnen von kurzen Basensequenzen bei der Gensequenzierung (*short read mapping* [57]), erfordern eine differenziertere Unterscheidung bei der Feststellung der Gleichheit oder Ungleichheit von Zeichen oder Sequenzen. Es wird konkret eine Quantifizierung der weichen Abweichung oder *Ähnlichkeit* zweier Sequenzen benötigt. Im folgenden Beispiel bedeutet in der betreffenden Sequenz eine Abweichung einer Basis (ein Zeichen der DNA) dennoch einen Treffer, falls diese Abweichung eine chemisch verwandte Basis ist. Tatsächlich ist es so, dass die Verwandtschaftsgrade der Basen zueinander bekannt sind und numerisch ausgedrückt werden können.

Sei das DNS-Alphabet $\Sigma = \{A, C, G, T\}$ bestehend aus den vier Basen Adenin, Cytosin, Guanin und Thymin, die die DNS bilden. Die Verwandtschaftsgrade der Basen werden in einer Matrix mit Spur 0 dargestellt, wobei die Übergänge von einer zu einer anderen Basis mit einem Widerstandswert gekennzeichnet sind. Diese sind in der *Penalty*-Tabelle 4.1 zusammengefasst und umso höher, je unwahrscheinlicher eine natürliche Mutation ist.

	G	**A**	**C**	**T**
G	0	1	3	2
A	1	0	1	3
C	3	1	0	2
T	2	3	2	0

Tabelle 4.1: DNS Penalty Tabelle

Für die Vergleichsfunktion wird eine Hilfsfunktion $pen : \Sigma \times \Sigma \to \mathfrak{C}^*$ benötigt, die genau den Widerstandswert für die Konversion eines gegebenen Basenpaares ermittelt. Da diese Funktion letzlich auch in einem booleschen Schaltkreis

ausgeführt werden muss, muss sie selbst in einen solchen überführt werden. Die dazu erforderlichen Schritte sind in Abbildung 4.4 dargestellt. Dies sind

1. Jede Basis c in den Zeilen- und Spaltenköpfen der Tabelle wird als $bin(c)$ notiert. Die Widerstandswerte werden ebenfalls in Binärform notiert.

2. Jede Binärziffer der Widerstandswerte wird in einer eigenen Tabelle notiert, die nur die Binärziffern gleicher Wertigkeit enthält.

3. Für jede der entstandenen Tabellen wird ein minimaler boolescher Schaltkreis mit Hilfe eines Karnaugh-Plans erstellt.

Schaltkreistiefe von *pen*. Sei ohne Beschränkung der Allgemeinheit der aus dem Karnaugh-Plan entwickelte Schaltkreis in der konjunktiven Normalform (KNF). Die oberste Konjunktion resultiert in logischen UND-Verknüpfungen der Tiefe $\log_2(|\Sigma|)$. Da nur logische AND- und XOR-Gatter im homomorph verschlüsselten Schaltkreis gestattet sind, wird jede Disjunktion $a \vee b$ (ODER) als zusammengesetzte Operation $(a \oplus b) \oplus (a \wedge b)$ ausgedrückt, wodurch sich die effektive Schaltkreistiefe vergrößert. Jede Disjunktion hat daher die Tiefe $\log_2(2|\Sigma|)$. Die gesamte Schaltkreistiefe der Funktion *pen* ist

$$\log_2(|\Sigma|) + \log_2(2|\Sigma|) \in \mathcal{O}(\log|\Sigma|)$$

und entspricht der Tiefe des zeichenbasierten Vergleichs $=_c$ in Abschnitt 4.2.1.

Durch die Ersetzung des zeichenbasierten Vergleichs mit der Widerstandsfunktion *pen* ist es erforderlich, dass die Widerstandswerte der einzelnen Zeichen zu einer Summe (der Quantifizierung der Ähnlichkeit) des betreffenden Suchbegriffs addiert werden, wie in Abschnitt 4.4.1 gezeigt. Dies resultiert in der folgenden Funktion:

$$\varphi : \begin{array}{c} \Sigma^* \times \mathfrak{C}^s \to \mathfrak{C}^{\lceil \log_2 s \rceil} \\ (w, s) \mapsto \sum_{i=0}^{|s|-1} pen(w_i, s_i) \end{array} . \tag{4.11}$$

	G	A	C	T
G	0	1	3	2
A	1	0	1	3
C	3	1	0	2
T	2	3	2	0

bin

❶

	00	**01**	**11**	**10**
00	00	01	11	10
01	01	00	01	11
11	11	01	00	10
10	10	11	10	00

❷

	00	**01**	**11**	**10**
00	0	0	1	1
01	0	0	0	1
11	1	0	0	1
10	1	1	1	0

❸

$$o_1 = (a_2 \vee b_2) \wedge (\overline{a_1} \vee \overline{b_1}) \wedge (\overline{a_2} \vee a_1 \vee \overline{b_2} \vee b1)$$

❷

	00	**01**	**11**	**10**
00	0	1	1	0
01	1	0	1	1
11	1	1	0	0
10	0	1	0	0

❸

$$o_0 = (\overline{a_2} \vee \overline{b_2}) \wedge (a_0 \vee b_0 \vee b_1) \wedge (b_0 \vee a_0 \vee a_1) \wedge (\overline{a_0} \vee a_1 \vee \overline{b_0} \vee b_1)$$

Abbildung 4.4: Konstruktion der *pen* Funktion: Transformation der Penalty Tabelle in einen booleschen Schaltkreis durch Karnaugh-Plan

4.4.3 Reguläre Ausdrücke

Um den Bereich der Fuzzy-Suche abzuschließen, sollen noch kurz die regulären Ausdrücke betrachtet werden. Dazu wird das Alphabet Σ des Suchbegriffs durch zwei Zeichen mit besonderer Semantik erweitert:

1. | (Alternative): “a|b” entspricht “a” and “b”.

2. $*$ (Kleene Stern): “a$*$” entspricht “”, “a”, . . . , “a $\cdots$ a”.

Der Einsatz von booleschen Schaltkreisen konstanter Tiefe bedingt, dass die Anzahl der Parameter (im Zusammenhang dieses Abschnittes ist das die Länge des Suchbegriffs) a priori bekannt ist. Das bedeutet, dass die Länge des Suchbegriffs nicht durch diesen selbst beeinflusst werden darf, und somit kann eine potenziell unbegrenzte Expansion durch den Kleene Stern $*$ mit dem hier behandelten Ansatz nicht abgebildet werden.

Die Alternative hingegen kann auf die quantifizierten Ähnlichkeiten aus Abschnitt 4.4.2 reduziert werden. Dies gilt für den Fall, dass im Term $(a|b)$ die betreffenden a und b auf einzelne Zeichen in Σ abgebildet werden und keine Expansion erfolgt. Jedes Vorkommen von $(a|b)$ im Suchbegriff kann dann durch die folgende Zeile

	$\cdots$	a	b	$\cdots$	
$(a\|b)$	$\cdots$	1	0	0	1	$\cdots$

in der Penalty Tabelle ersetzt werden und wird so durch die *pen*-Funktion verarbeitet, wie im vorausgehenden Abschnitt beschrieben.

4.5 Ergebnisse

Dieses Kapitel zeigte, wie Suchbegriffe in einer Menge durch homomorph verschlüsselte Funktionen exakt oder weich (fuzzy) gefunden werden können. Die Größe der Funktion φ, die im homomorph verschlüsselten Raum ausgeführt werden muss, ist in jedem der vorgestellten Fälle im Bereich der Platzkomplexität $\mathcal{O}(\log_2 |\Sigma|)$. Das bedeutet, dass die Größe des Schaltkreises von vornherein bekannt ist und insbesondere *nicht* von der Größe der durchsuchten Datenbank abhängt (wohl aber die Zeitkomplexität).

Kapitel 5

Ein homomorph verschlüsseltes Prozessormodell

5.1 Designziele

Die verschlüsselte Programmausführung mittels homomorpher Kryptografie basiert auf der Überführung eines Programms in eine Repräsentation aus booleschen Schaltkreisen. Dies verringert die erforderliche strukturelle Komplexität des Kryptosystems, weil lediglich recht einfache Basisoperationen wie AND und XOR oder NAND dargestellt werden müssen. Die Schaltkreisabbildungen lassen sich nach dem Schritt der Arithmetisierung mit den entsprechenden Mitteln des Kryptosystems verschlüsseln und werden anschließend dem Ausführenden als Datenpaket übergeben. Dieser kann mit der unverschlüsselten Verarbeitungsvorschrift, dem booleschen Schaltnetz des Anwendungsprogramms, die verschlüsselten Operanden in der richtigen Reihenfolge homomorph addieren bzw. multiplizieren. Das so berechnete, verschlüsselte Ergebnis kann nur der Auftraggeber mit seinem geheimen Schlüssel entschlüsseln. Der Programmausführer kann hier zwar den verwendeten Algorithmus nachvollziehen, da er ihm lesbar vorliegt. Allerdings kann er weder Eingabe- noch Ausgabedaten lesen. Um in diesem Szenario auch ein verschlüsseltes Programm ausführen zu können, begeben wir uns auf die niedrigere Abstraktionsstufe der Prozessorebene. Die unverschlüsselte Verarbeitungsvorschrift implementiert in diesem Fall die Schaltkreise eines Mikroprozessormodells. Das einfache Schichtenmodell dieser Architektur ist in Abbildung 5.1 dargestellt. Der verschlüsselte Teil enthält

das eigentliche binäre Anwendungsprogramm samt Daten. Da die unverschlüsselte Prozessor- und Speicherschaltung in jedem Takt immer gleichförmig abgearbeitet werden muss, sind wegen der verschlüsselten Operanden von außen keine Folgerung auf im Inneren ablaufende Operationen, Speicherzugriffe, Programmsprünge oder –schleifen möglich.

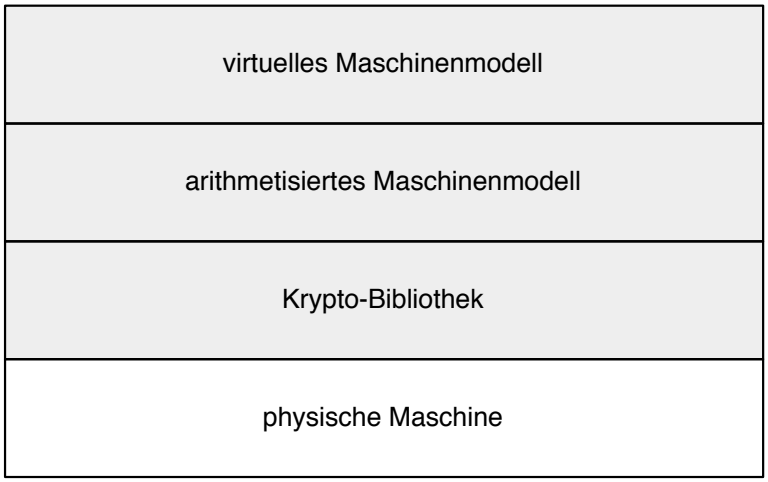


Abbildung 5.1: Architektur-Schichtenmodell

Der Preis für die universale Einsetzbarkeit eines solchen Systems ist vor allem die benötigte Rechenzeit, was auf zwei Faktoren zurückzuführen ist. Zum einen erfordert das unterliegende Kryptosystem aufgrund seiner mathematischen Beschaffenheit eine recht große Zusatzrechenleistung. Diese dient der zyklischen Laufzeitkorrektur von systembedingten Eigenschaften, wie der Genauigkeit einzelner Chiffretexte. Zum anderen benötigt der Speicherzugriff aufgrund der seriellen Schaltkreissimulation verhältnismäßig viel Zeit. Tatsächlich steigt hier die Zugriffszeit auf eine einzelne verschlüsselte Speicherzelle linear mit der gesamten verschlüsselten Speichergröße an.

In diesem Kapitel wird ein verschlüsseltes Prozessormodell vorgestellt. Dabei handelt es sich zunächst um eine arithmetisierte Version der Schaltkreise einer CPU, die mittels homomorpher Kryptografie verschlüsselt werden und somit die vollständig verschlüsselte Simulation des zugrundeliegenden Prozessors ermöglichen. *Die Grundlagen zu diesem Kapitel wurden in [14, 15] wissenschaftlich publiziert. Außerdem wurden Patente [12, 13] angemeldet.*

Aufgrund der Laufzeitbeschränkungen behandelt dieses Kapitel auch eine Konzeptskizze zur Umsetzung der CPU-Simulation in Hardwarekomponenten.

Dazu werden einige Schritte für ein weiterführendes Projekt zur Anschlussforschung vorgeschlagen. Die Hardware-Umsetzung ist jedoch nicht Gegenstand dieser Arbeit.

5.2 Zur Nutzung bestehender Architekturen

Das Systemdesign ist aufgrund der seriellen Schaltkreissimulation in der Software-Implementierung auf eine geringe Anzahl von Schaltelementen ausgelegt. So sind keine Konzepte wie Pipelining, Superskalarität oder Caching [36] integriert worden, die im Falle einer physischen CPU zur Beschleunigung der Verarbeitung beitragen. Im Kontext der unparallelisierten Schaltkreissimulation durch die softwareimplemenierte CPU hätte dies den gegenteiligen Effekt, da die Befehlsstränge die Funktionseinheiten letztendlich doch seriell durchlaufen und die Implementierung der Beschleunigungsmaßnahmen zu einer erhöhten Anzahl von Schaltelementen führt.

Dies lässt sich bereits an einer einfachen Schaltung wie einem Addierwerk nachvollziehen. Ein recht einfacher *Ripple-Carry-Adder* (RCA), also ein Serienaddierer, bei dem der Übertrag von der unteren bis zur oberen Addierstufe durchgereicht wird, besitzt nur eine geringe Komplexität bei linearer Laufzeit in Abhängigkeit der Wortgröße. Der Flächenbedarf, d. h. die tatsächliche Anzahl der zur Hardware-Implementierung erforderlichen Schaltgatter ist ebenfalls linear in Abhängigkeit der Wortgröße. Im Gegensatz dazu besitzt ein *Carry-Look-Ahead-Adder* (CLA), also ein Addierwerk, bei dem die Überträge der einzelnen Addierstufen parallel berechnet werden eine nahezu konstant geringe Laufzeit für kleine Wortbreiten. Dieses Verhalten wird allerdings mit einer deutlich erhöhten Anzahl an erforderlichen Schaltelementen erkauft. So ist der Flächenbedarf eines CLA durch ein kubisches Verhältnis zur Wortbreite gekennzeichnet.

Das Kernproblem bei komplexen Schaltungen ist im Bezug auf eine homomorphe Verschlüsselung nicht die Ausführungszeit der Schaltgatter, bzw. der Schaltfunktionen selbst, sondern der durch eine höhere Anzahl von angewendeten Schaltfunktionen gleichfalls erhöhte Rechenbedarf für die Laufzeitkorrektur der Operanden. Diese *Recrypt*-Problematik wurde in Kapitel 3 im Rahmen der unbegrenzt homomorphen Kryptosysteme besprochen.

5.3 System-Konzept & Schaltkreisverschlüsselung

Das Konzept zur Simulation einer verschlüsselten CPU setzt sich aus mehreren Teilaspekten zusammen, die zu der System-Architektur zusammengeführt werden. Im Wesentlichen handelt es sich dabei um das homomorphe Verschlüsselungsschema, ferner ein Verfahren zur Arithmetisierung von Schaltkreisen und schließlich konkrete Schaltkreise und Funktionsgruppen, welche die Funktionalität von CPU und Speicher abbilden. Ein System wie das hier beschriebene zeichnet sich vor allem dadurch aus, dass sowohl Programme als auch Daten durch den verschlüsselten Speicher und die verschlüsselte CPU geschützt sind. Durch die Abbildung eines integrierten Gesamtsystems auf verschlüsselte Schaltkreise lässt sich auch die Charakteristik des Speicherzugriffs verbergen, sodass beispielsweise die Analyse der statistischen Zugriffshäufigkeit bestimmter Speicherbereiche verhindert werden kann[1].

Zur Verschlüsselung eines booleschen Schaltkreises kann gemäß der Umsetzungen in Kapitel 4 ein homomorphes Schema verwendet werden, wenn es die Multiplikation und die Addition im Kryptoraum beherrscht. Diese beiden Operationen können auf die booleschen Funktionen AND und XOR abgebildet werden, womit sich alle weiteren booleschen Funktionen ableiten lassen. Dadurch können Schaltkreise von beliebiger Funktionalität zusammengestellt werden.

Definition 5.1. In booleschen Ausdrücken bezeichnet der Operator $\oplus$ die XOR-Operation, der Operator $\otimes$ steht für die AND-Operation.

Die XOR-Operation besitzt die Charakteristik der binären Addition ohne Übertrag, dagegen gleicht die AND-Operation der binären Multiplikation.

Beispiel. Der boolesche Ausdruck $r = (a \oplus b) \oplus (a \wedge b)$, der in der booleschen OR-Operation resultiert, kann in Ganzzahl-Arithmetik als $r = (a+b)+(a*b)$ ausgedrückt werden, wobei a und b Ganzzahl-Repräsentationen von Bitwerten mit äquivalenten Paritäten sind.

[1] siehe zur sog. *Obliviousness* auch [32]

Definition 5.2. Von hier an wird für die zusammengesetzte OR-Operation in Ganzzahl-Arithmetik der Operator $\circ$ verwendet. Die Operation ist definiert als $a \circ b = (a + b) + (a * b)$.

5.4 Architektur-Komponenten

Dieser Abschnitt stellt die Mikroarchitektur einer Modellmaschine vor, mit der die Tragfähigkeit des Konzeptes der verschlüsselten Programmausführung nachgewiesen wird. Die Mikroarchitektur-Komponenten können auch zur Umsetzung anderer Modelle verwendet werden. Im weiteren Verlauf dieses Abschnittes und im Abschnitt 5.5 werden die Basis-Komponenten zur Abbildung einer Befehlssatz-Architektur (Instruction Set Architecture, ISA) zusammengesetzt.

5.4.1 Memory Unit (MU) & Speicherzellen

Abbildung 5.2 zeigt einen grundlegenden Demultiplexer-Schaltkreis, welcher als Speicherschaltkreis verwendet werden kann. In diesem Diagramm werden die gespeicherten Werte über die m-Leitungen statisch in die Schaltung eingebracht. Diese Darstellung lässt sich leicht in eine Software-Simulation umsetzen. Das Ausgabe-Bit dieser einzelnen Speicherbit-Spalte lässt sich wie folgt berechnen:

$$c = (\neg a_0 \wedge \neg a_1 \wedge m_0) \vee (a_0 \wedge \neg a_1 \wedge m_1) \vee (\neg a_0 \wedge a_1 \wedge m_2) \vee (a_0 \wedge a_1 \wedge m_3).$$

Durch Erweiterung der Funktion auf die erforderliche Anzahl von Adressleitungen und Speicherzeilen kann ein Speicherschaltkreis in beliebiger Größe modelliert werden. Nun wird die boolesche Schaltlogik in eine arithmetische Repräsentation überführt, wobei der folgende Ausdruck entsteht:

$row_0 = ((a_0 + 1) * (a_1 + 1) * m_0)$
$row_1 = (a_0 * (a_1 + 1) * m_1)$
$row_2 = ((a_0 + 1) * a_1 * m_2)$

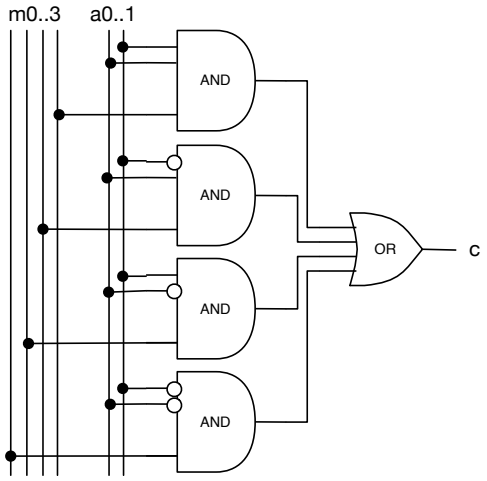


Abbildung 5.2: Speicherschaltkreis

$row_3 = (a_0 * a_1 * m_3)$

$c = row_0 \circ row_1 \circ row_2 \circ row_3$

Beispiel. Seien $m_0..m_3$ die Bits einer Speicherspalte mit den Werten $\{1, 0, 1, 0\}$ und sei $a_0..a_1$ die Bitsequenz {0,1} als Repräsentation der Adresse 2 in Binärform. Der arithmetische Ausdruck zur Berechnung des Zugriffsergebnisses ist dann

$row_0 = ((0 + 1) * (1 + 1) * 1) = 0$
$row_1 = (0 * (1 + 1) * 0) = 0$
$row_2 = ((0 + 1) * 1 * 1) = 1$
$row_3 = (0 * 1 * 0) = 0$

$r = 0 \circ 0 \circ 1 \circ 0 = 1$

Im speziellen Fall eines Speicherzugriffs, bzw. eines Zugriffs der genau ein Element auswählt, kann die abschließende OR-Verknüpfung durch eine XOR-Verknüpfung ersetzt werden. Dadurch lässt sich gemäß der Definition 5.2 der OR-Operation je OR eine homomorphe Multiplikation einsparen.

Werden die Schaltwerte nun verschlüsselt, so kann mit verschlüsselten Adressen auf verschlüsselte Speicherbereiche zugegriffen werden, d. h. die Zugriffsfunktion verrät weder Zugriffsadresse noch den Inhalt. Unter der Annahme eines probabilistischen Kryptosystems wie dem von Smart et al., bei dem dem gleichen Klartext in zwei unterschiedlichen Verschlüsselungsvorgängen mit hoher Wahrscheinlichkeit unterschiedliche Chiffretexte zugeordnet werden, ist zu beobachten, dass der Zugriff auf den Speicher mit unterschiedlichen Repräsentationen derselben Adresse auch zu unterschiedlichen (jedoch im Bezug auf die Entschlüsselung äquivalenten) Repräsentationen der Speicherausgabe führen.
Beispiel. Seien $m_0..m_3$ die Bits einer Speicherspalte mit den Werten $\{1, 0, 1, 0\}$ und sei $a_0..a_1$ die Bitsequenz {0,1} als Repräsentation der Adresse 2 in Binärform. Unter Zuhilfenahme des homomorphen Kryptoschemas $\mathfrak{H}$ aus Kapitel 3.3 werden die Operanden verschlüsselt. Dabei wird dieselbe Technik angewendet, die bereits in Abschnitt 3.6 verwendet wurde, um die einfachen Beispielalgorithmen zu formulieren. Da das verwendete Kryptoschema $\mathfrak{H}$ in den Beispielen dort und ebenso in diesem Beispiel Zahlen aus einem Wertebereich $\{2^\eta, p\}$ mit $\eta > 1$ statt aus einem binären Bereich $\{0, 1\}$ verwendet, soll noch kurz skizziert werden, wie sich mit Ganzzahl-Arithmetik boolesche Algebra ausführen lässt. Die Tabelle 5.1 zeigt analog zur Tabelle 3.3 die Abbildung der booleschen Funktionen XOR und AND auf modulo 2-Arithmetik anhand der Paritäten *gerade* und *ungerade* der beteiligten Operanden.

(a) + *Paritäten*, $\oplus$

	g	**u**
g	g	u
u	u	g

(b) · *Paritäten*, $\wedge$

	g	**u**
g	g	g
u	g	u

Tabelle 5.1: Überführung boolescher Operationen in Paritäten von + und ·

Zur Abbildung von 0-Bits kommen dann gerade, ganzzahlige Klartexte zum Einsatz. 1-Bits werden durch ungerade, ganzzahlige Klartexte dargestellt. Für das Beispiel werden für $m_0..m_3$ die Klartext-Repräsentationen $m'_0..m'_3 \leftarrow \{5, 4, 9, 6\}$ per Zufall ausgewählt. Für die Adress-Bits a_0, a_1 werden die Repräsentanten a'_0, a'_1 mit $\{8, 3\}$ belegt. Die Verschlüsselung mit dem $\mathfrak{H}$-Schlüssel $p = 911$ und dem Zufallswert $r_0 = 4$ ergibt für den verschlüsselten Speicherin-

halt $\mathfrak{m}_0$:

$$\mathfrak{m}_0 \leftarrow m'_0 + r_0 p \Rightarrow \mathfrak{m}_0 \leftarrow 5 + 4 \cdot 911 \Rightarrow \mathfrak{m}_0 \leftarrow 81$$

und für zufällige $r_1..r_3 \leftarrow \{6, 5, 2\}$ ergeben sich die verschlüsselten Speicherinhalten $\mathfrak{m}_1..\mathfrak{m}_3$:

$$\mathfrak{m}_1..\mathfrak{m}_3 \leftarrow \{118, 104, 44\}$$

Und für weitere $r_0, r_1 \leftarrow \{7, 4\}$ ergibt sich:

$$\mathfrak{a}_0, \mathfrak{a}_1 \leftarrow \{141, 79\}$$

Mithin errechnet sich das Beispiel von oben in verschlüsselter Form wie folgt:

$row_0 = ((6385 + 1) * (3647 + 1) * 3649) = 23296128$
$row_1 = (6385 * (3647 + 1) * 5470) = 127409865600$
$row_2 = ((6385 + 1) * 3647 * 4564) = 106294382488$
$row_3 = (6385 * 3647 * 1828) = 42566981660$

$$c = \underbrace{23296128}_{row_0} + \underbrace{127409865600}_{row_1} + \underbrace{106294382488}_{row_2} + \underbrace{42566981660}_{row_3} = 276294525876$$

Entschlüsselt *mod* p ergibt sich 551 und *mod* 2 das binäre Ergebnis 1. Dies ist auch der Inhalt des m-Speichervektors an Indexposition 2, wie durch den Adressvektor a adressiert.

Um einer Speicherzelle einen neuen Wert zuzuweisen, wird dessen Repräsentation in die Zugriffsfunktion als Parameter i (*input*) gegeben. Der neue Zellenwert kann dann durch die folgende Gleichung berechnet werden:

$$m_{new} = (row \wedge i) \vee (\neg row \wedge m)$$

wobei row ein nach dem oben gezeigten Prinzip generiertes row-select Sig|-nal ist. Dieses ist im Falle einer durch eine Adresse ausgewählten Speicherzeile logisch 1, generiert für die Zuweisung ebenfalls eine logische 1, falls i gesetzt ist und den bisherigen Inhalt der Speicherzelle, wenn die Adresse die aktuell bearbeitete Zeile nicht auswählt. Auch wenn eine Speicherzelle nicht durch die

Zugriffsfunktion adressiert wird, berechnet die Funktion dennoch eine neue Repräsentation des bisherigen Inhalts, sodass nach einem Zugriffszyklus das gesamte Speicherabbild neu geschrieben wird. Um zwischen Lese- und Schreibzugriff zu unterscheiden, kann ein entsprechendes Flag in die Zugriffsfunktion eingebracht werden, wodurch die Datenrichtung definiert wird. Unterscheidung bedeutet in diesem Zusammenhang keine Fallunterscheidung im logischen, sondern im arithmetischen Sinne, da die Datenrichtung lediglich in die gesamte Berechnung einfließt und das Endergebnis nach Durchführung aller Schritte beeinflusst. Die vollständige Zugriffsfunktion für den Lese- und Schreibzugriff im Adressraum $\mathfrak{A}$ lautet

$$\begin{aligned} \forall a \in \mathfrak{A} : m_a &\leftarrow (row_a \wedge write \wedge reg) \vee \\ &\quad\ (row_a \wedge read \wedge m_a) \vee \\ &\quad\ (\neg row_a \wedge m_a) \\ reg &\leftarrow (row_a \wedge read \wedge m_a) \vee \\ &\quad\ (row_a \wedge write \wedge reg) \vee \\ &\quad\ (\neg row_a \wedge reg) \end{aligned} \tag{5.1}$$

Satz. Da die Speicherzugriffsfunktion bei jedem Zugriff die logische Kombination aller verfügbaren Zellen berechnet, hängt die Komplexität des Speicherzugriffs maßgeblich von der Größe ϕ des Speichers ab und wird durch eine annähernd lineare Funktion beschrieben. Die Zahl der zu berechnenden booleschen Gatterfunktionen $B_{\{1,2,3\}} = \{\neg, \wedge, \vee\}$ während eines Lesezugriffs ist $f(\phi) = (\phi * B_1) + (2 * \phi * B_2) + ((\phi - 1) * B_3)$.

5.4.2 Arithmetic-Logical Unit (ALU)

Zur Modellierung der ALU kommen die gleichen Konzepte zum Einsatz wie zuvor beim verschlüsselten Speicher. Abbildung 5.3 zeigt eine einfache 1-Bit ALU, die Funktionen zu Addition und logischen Verknüpfungen zweier Operandenbits bereitstellt. Die Einheit besteht in der Umsetzung aus recht einfachen Schaltkreisen, deren Funktionen durchgehend auf die zwei jeweils anliegenden Operanden a und b angewendet werden. Vergleichbar sind diese Funktionsergebnisse mit den statischen Speicherzellen in der Speicherschaltung. Technisch identisch mit der Demultiplexer-Einheit in der Adressauswahl des Speichers wird das Zielergebnis der ALU mittels eines Opcodes ausgewählt. Dieser be-

steht im vorliegenden Beispiel aus zwei Bits, sodass vier Operationen codiert werden können. Im Einzelnen sind dies für die Opcode-Bits o_0, o_1 die Operationen $\{0,0\} \mathrel{\widehat{=}} add, \{1,0\} \mathrel{\widehat{=}} and, \{0,1\} \mathrel{\widehat{=}} xor, \{1,1\} \mathrel{\widehat{=}} not$.

Auf Basis dieser Zuordnung kann die ALU mit den folgenden Gleichungen in boolescher Logik modelliert werden:
$c_{add} = fulladder(a,b)$, $c_{and} = a \wedge b$, $c_{xor} = a \oplus\ b$, $c_{not} = \neg a$

Die folgende Sequenz berechnet mit diesen Gleichungen das Ergebnis c einer gegebenen Operation:
$c = (c_{add} \wedge (\neg o_0 \wedge \neg o_1)) \vee, (c_{and} \wedge (o_0 \wedge \neg o_1)) \vee, (c_{xor} \wedge (\neg o_0 \wedge o_1)) \vee, (c_{not} \wedge (o_0 \wedge o_1))$

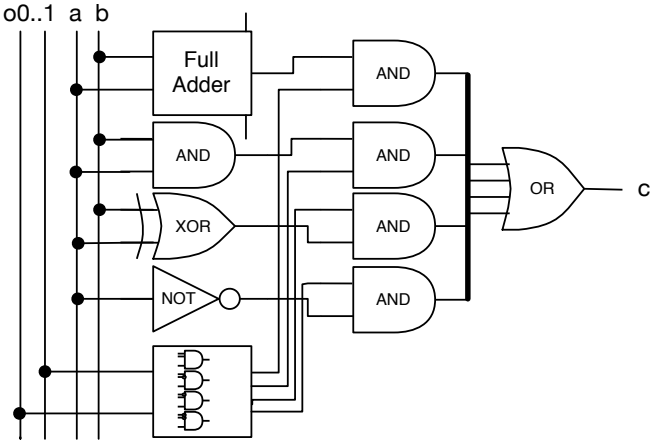


Abbildung 5.3: 1-bit ALU Schaltung

Wie gezeigt wurde, sind Speicherzugriff und die Operationsauswahl einer ALU technisch vergleichbar. Die Transformation in verschlüsselbare ganzzahlige Arithmetik kann auch vom zuvor gezeigten Konzept der Speicherschaltung abgeleitet werden.

5.4.3 Control Unit (CU)

Die bis hierher vorgestellten Konzepte sind im Prinzip unabhängig von der Umgebung, in der sie eingesetzt werden. Das bedeutet, dass ein Speicherschaltkreis nicht in Verbindung mit einer CPU eingesetzt werden muss, sondern jeder Komponente, die in der Lage ist, binäre Adress-Signale zu erzeugen als Speicher dienen kann. Das gilt im übertragenen Sinne auch für eine ALU.

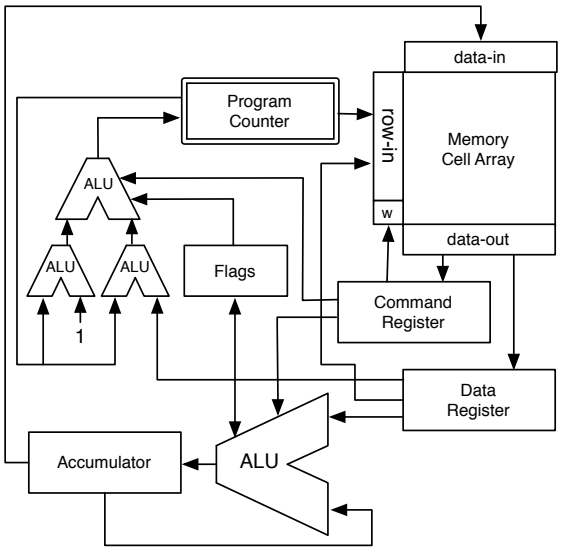


Abbildung 5.4: CPU Schema (Datenpfad)

Die Komponente, die einen Systemtyp definiert und die Zusammenschaltung der einzelnen Komponenten strukturiert, ist die Steuereinheit einer Maschine. Sie ist somit das wichtigste Element und bildet das zentrale Verbindungselement für alle angebundenen Einzelteile und regelt sowohl Datenfluss als auch Programmablauf. Die Steuereinheit ist dabei ein zustandsbehafteter Automat, der in jedem Takt die gleichen atomaren Operationen ausführt und durch die Parametrisierung dieser Operationen in Form von Opcodes und Operanden gesteuert wird. Dabei liest die Steuereinheit nach dem Maschinenstart zunächst eine definierte Speicherstelle und ermittelt mit Hilfe des Decoders das auszuführende Kommando. Sollten zur Abarbeitung noch weitere Operanden erforderlich sein, so werden diese aus dem Speicher in entsprechende Register geladen. Dazu aktiviert der Decoder mit Hilfe von Selektoren und Demultiplexern die erforderlichen funktionalen Einheiten und trägt so zur Berechnung des nächsten Systemzustandes bei. Dieser Folgezustand besteht zum einen aus dem Ergebnis eines arithmetischen oder logischen Kommandos, d. h. es liegt ein Ergebnis in Form eines Registerinhaltes und in Form von Flags vor. Zum anderen berechnet die Steuereinheit die Adresse des als nächstes zu bearbeitenden Kommandos. Dies ist im Falle eines linearen Programmablaufs das im Speicher dem aktuell bearbeiteten folgende und im Falle eines Sprunges das durch einen Registerinhalt adressierte Kommando.

Dabei ist die Durchführung beliebiger Programmsprünge eine grundsätzlich wichtige Eigenschaft für jede Programmausführung. Dies ist erforderlich, um komplexe, kontextabhängige und nicht-lineare Programmabläufe abbilden zu können. Anhand der Abbildung 5.4, die den Datenpfad des hier vorgestellten CPU-Modells zeigt, soll kurz in das Gesamtkonzept und die Steuereinheit im Speziellen eingegangen werden. Wie klar zu erkennen ist, handelt es sich bei der Modell-CPU um eine Akkumulator gestützte Architektur, d. h. es ist nur ein Arbeitsregister vorgesehen, das gleichzeitig Zielregister für arithmetische Operationen der ALU und eines der Quellenregister für die Operanden ist. Der Speicher ist in kombinierter Harvard / von-Neuman Architektur ausgeführt. Das bedeutet, dass einerseits Programm- und Datenspeicher getrennt sind, andererseits liegt der Opcode in den unteren 5 Bits des 13 Bit langen Datenwortes, und so können sowohl Opcode als auch Operand in einem einzigen Zugriff geladen werden. Bei einem Schreibzugriff werden jedoch nur die oberen 8 Datenbits mit dem Inhalt des Akkumulators überschrieben, sodass selbstmodifizierender Code ermöglicht wird. Diese Maßnahmen vereinfachen das Design der Steuereinheit und verringern massiv die erforderliche Schaltungsgröße, weil beispielsweise komplexere Adressierungsarten wie die indirekte oder die indizierte durch Codemodifikation während der Laufzeit verwendet werden können.

Der folgende Absatz beschreibt einen Taktzyklus der Modell-CPU anhand der Datenpfad-Abbildung. Eine wesentliche weiter gefasste Einführung in Schaltkreis- und CPU-Design findet sich in Hennessy & Patterson [36] und Patterson & Hennessy [48]. Ausgehend vom Programmzählregister (Program Counter) wird dieses anfangs mit der Startadresse des Anwendungsprogrammes initialisiert und veranlasst so das Laden des entsprechenden Speicherwortes in das Daten- und das Instruktionsregister (DR für *data register*, CR für *command register*). Das Datenregister dient neben dem Inhalt des Akkumulators als zweiter Operand für die Hauptrecheneinheit, während die Information im Instruktionsregister vorwiegend Kontrollzwecken dient. So wird aus dem Opcode z. B. das entsprechende Steuersignal für die Recheneinheit dekodiert sowie die Adresse des nächsten Kommandos berechnet. In diese Berechnung fließen jeweils auch die Flags mit ein.

Das Programmzählregister ist auch das Zielregister bei Sprungoperationen.

Bei kontextunabhängigen Sprüngen (Jumps) wird die Zieladresse einfach in das Programmzählregister hineinkopiert, während dieser Vorgang bei kontextabhängigen Sprüngen oder Verzweigungen (Branches) durch die Statusinformationen des Prozessors, den Flags, beeinflusst wird. Beispiele dafür sind z. B. das Zero-Flag, das anzeigt, dass die letzte arithmetische Operation eine numerische 0 ergab oder dass ein Vergleich die numerische Gleichwertigkeit zweier Operanden ergab. Ein weiteres wichtiges Flag ist das Übertrags- oder Carry-Flag, welches anzeigt, dass die letzte arithmetische Operation einen Übertrag ergab, z. B. bei einer Addition. Die Eigenschaft, im verschlüsselten Raum Sprünge und Verzweigungen auszuführen, unterscheidet das in dieser Arbeit vorgestellte Konzept signifikant von anderen Ansätzen (siehe auch Abschnitt 2). Der Programmfluss wird im Diagramm 5.4 von der ALU-Gruppe im linken Teil gesteuert. Hier wird eine statische 1 im Falle eines linearen Programmablaufs zum Programmzählregister addiert, während im Falle eines Sprunges der Inhalt des Datenregisters DR addiert wird. Bei einer kontextabhängigen Verzweigung wird zusätzlich mit Hilfe des Flag-Registers entschieden, ob das Programm linear oder eben mit Verzweigung fortgesetzt wird.

Sei F die Menge der zur Verfügung stehenden Flags, PC das Programmzählregister (Program Counter), DR das Datenregister und CR das Befehlsregister. Die Funktionen $jmp(CR)$, $bcc(CR)$, und $bz(CR)$ seien als boolesche Funktionen ausgeführt und liefern eine positive Ausgabe (*wahr*), sofern im Befehlsregister der entsprechende Opcode für den jeweiligen Sprungbefehl steht. Die Adresse des als nächstes auszuführenden Befehls lässt sich durch folgende Schaltfunktion ermitteln:

$$\forall x : x \in \{0..wordsize-1\},\ PC_x = (jmp(CR) \wedge DR_x)$$
$$\vee\ (bcc(CR) \wedge DR_x \wedge \neg F_{carry}) \vee (bz(CR) \wedge DR_x \wedge F_{zero})$$
$$\vee\ (\neg jmp(CR) \wedge \neg bcc(CR) \wedge \neg bz(CR) \wedge (PC+1)_x).$$

Das CPU-Schema in Abbildung 5.4 zeigt außerdem, wie die einzelnen Architekturkomponenten zusammengeschaltet werden können, um einen Prozessor abzubilden. Diese Methode zur Konstruktion einer CPU ist der erste Ansatz der es erlaubt, sowohl Programme als auch Daten während der gesamten Dauer der Verarbeitung im verschlüsselten Zustand zu belassen. Dennoch können beliebige Programme, bzw. Programme mit beliebigen algorithmischen Kon-

strukten ausgeführt werden.

5.5 Software-Implementierung

Das im vorangegangenen Kapitel vorgestellte verschlüsselbare Prozessormodell wird in diesem Kapitel mit einer Implementierung überprüft. *Die Implementierungsansätze und die Performance-Messungen wurden in [10, 11] wissenschaftlich publiziert.*

Alle Laufzeitmessungen erfolgen im Zusammenhang mit einem Sicherheitsparameter λ, der im Wesentlichen die Schlüssellänge des unterliegenden Kryptosystems darstellt. Details sind in Kapitel 3 nachzulesen.

5.5.1 Laufzeitumgebung

Zur Umsetzung des Konzeptes in Software wurden unterschiedliche Umgebungen gewählt, um die Universalität nachzuweisen. Zum einen existiert eine Implementierung in Java, wobei hier aber nur exemplarisch die Überführung der arithmetisierten Schaltfunktionen der einzelnen Komponenten in native Java-Calls umgesetzt wurde. Da die Schicht der Prozessor- und Speicherschaltkreise von der unterliegenden Kryptografie vollständig entkoppelt ist, kann hier bei Verfügbarkeit eine beliebige Krypto-Bibliothek eingesetzt werden.

Die vollständige Implementierung liegt in C vor, wobei dies vor allem der Verfügbarkeit leistungsfähiger mathematischer Bibliotheken zuzuschreiben ist. So liegen für zahlreiche Funktionen der schnellen Polynom-Arithmetik und zahlentheoretischer Funktionen die entsprechenden Implementierungen in der freien Bibliothek *FLINT*[2] vor. Diese wiederum nutzt die freien Bibliotheken *MPFR*[3], *MPIR*[4] und *GMP*[5].

Als Betriebssystemplattformen wurden ebenfalls unterschiedliche Produkte gewählt. Zum einen eine Linux-Plattform (Ubuntu 11) sowie MacOS X 10.7.

[2]Fast Library for Number Theory, http://www.flintlib.org

[3]GNU multiple-precision floating-point computations with correct rounding, http://www.mpfr.org

[4]Multiple Precision Integers and Rationals, http://www.mpir.org

[5]The GNU Multiple Precision Arithmetic Library, http://gmplib.org

5.5.2 Komponenten-Implementierung

Die einzelnen Komponenten des verschlüsselten Prozessormodells werden bewusst einfach gehalten, sodass die Überführung von der Schaltlogik in die verschlüsselten Schaltkreise leichter nachvollziehbar wird. Andererseits ist auch die Kontrolle über die tatsächlich erzeugten Schaltfunktionen größer, als wenn eines der gängigen Hilfsmittel zur halbmaschinellen Schaltungsgenerierung verwendet würde. Die Umsetzung eines komplexen Hardware-Modells in Verilog oder VHDL[6] ist Gegenstand des in Abschnitt 5.6 skizzierten Forschungsfolgeprojektes, in dem Teile des Prozessormodells für die Umsetzung auf einem rekonfigurierbaren Hardwarebaustein (FPGA[7]) realisiert werden kann.

Speicherzugriff

Tabelle 5.2 zeigt die Performancemessungen der Implementierung der Zugriffsfunktion. Die Zeiten wurden als $\frac{\Delta clock()}{CLOCKS\_PER\_SEC}$ mit Hilfe der entsprechenden Funktionen und Makros aus der Standard-clib gemessen. Wie die Messungen zeigen, ist die Zeit, die für einen Lesezugriff benötigt wird, größer als die für einen Schreibzugriff. Dies hat den einfach zu erklärenden Grund, dass bei einem Lesezugriff das Rauschen aller Speicherzellen im Zielregister akkumuliert wird und daher eine große Anzahl von recrypt-Operationen erforderlich ist. Dagegen werden die Repräsentationen der Speicherzellen jeweils nur mit dem Rauschen des Quellregisters verknüpft, und somit ist es möglich, dass für einen Schreibzugriff keine einzige recrypt-Operation fällig wird. Zu beachten ist auch der Aspekt, dass die Zugriffe in einem *eingeschwungenen* System generell länger dauern, als in einem, in dem erst wenige Speicherzugriffe stattgefunden haben. Dies wird besonders bei relativ großen Speichern unterhalb des Maximums deutlich (64 oder 128 Speicherzeilen in einem System von 256 Zeilen).

[6] Verilog und VHDL sind Schaltungsbeschreibungssprachen die insbesondere für die maschinelle Umsetzung in Hardware verwendet werden

[7] auf einem Field Programmable Gate Array können beliebige Schaltungen aus vielen Gattern und Funktionsbausteinen bestehend realisiert werden

Speicherzeilen	$\lambda = 384$	$\lambda = 512$	$\lambda = 768$	$\lambda = 1024$
8	1 / <1	1 / <1	1 / <1	1 / <1
16	4 / <1 y<	4 / <1	5 / <1	5 / <1
32	17 / <1	19 / <1	21 / <1	25 / <1
64	9-18 / 4	9-19 / 4	11-22 / 5	13-25 / 6
128	21-63 / 9	22-68 / 9	25-76 / 11	30-90 / 12
256	51 / 15	53 / 15	62 / 17	71 / 21

Tabelle 5.2: Speicherzugriff in Sekunden (lesen/schreiben)

Das Interface für die Speicherzugriffsfunktion MA ist

$$\overline{reg} \leftarrow MA(\overline{adr}, \overline{reg}, read)$$

wobei reg das Quell- oder Zielregister ist, adr die zugegriffene Adresse und $read$ das Datenrichtungs-Flag. In der hier beschriebenen konkreten Implementierung des Systems wird die Zugriffsfunktion innerhalb der Control-Unit aufgerufen, wobei offensichtlich ist, welche Richtung der Datentransfer hat (siehe auch Abschnitt 5.4.3 für weitere Details). Für diesen Fall kann die Zugriffsfunktion also in zwei spezialisierte Funktionen aufgeteilt werden, um weitere Performance-Gewinne im Vergleich zur universalen Zugriffsfunktion zu erzielen. Die beiden Funktionen implementieren zusammen Gleichung 5.1 über einem Speicherarray `A` wie in den Listings 5.1 und 5.2 dargestellt.

Listing 5.1: Speicherzugriff (lesend)

```
1  reg MAr(adr)
2  {
3    for(i = 0..ROWS)
4    {
5      sel = ROWSELECT(i,adr);
6      for(j = 0..WORDSIZE)
7      {
8        reg.j  = (A[i].j * sel)+(reg.j  * !sel);
9      }
10   }
11 }
```

Listing 5.2: Speicherzugriff (schreibend)

```
1  MAw(adr,reg)
2  {
3    for(i = 0..ROWS)
4    {
5      sel = ROWSELECT(i,adr);
6      for(j = 0..WORDSIZE)
7      {
8        A[i].j = ( reg.j * sel)+(A[i].j * !sel);
9      }
10   }
11 }
```

Hier bezeichnet der Operator | die zusammengesetzte OR-Operation und $X.y$ repräsentiert das y-te Bit des Speicherwortes X. Die Funktion $ROW\ SELECT$ implementiert die bereits erwähnte Zeilenauswahl, wie oben erläutert und ergibt eine logische 1, wenn die verschlüsselten Repräsentanten der binär dargestellten Speicheradresse adr dem Zeilen-Iterator i entspricht. Listing 5.3 zeigt den entsprechenden Pseuso-Code.

Listing 5.3: Generierung des row-select

```
1  sel ROWSELECT(i,adr)
2  {
3    sel=ENCRYPT(1);
4    for(j = 0..ADRSIZE)
5    {
6      if(int.j==1)
7        temp=ENCRYPT(1)*adr[j];
8      else
9        temp=ENCRYPT(1)*!adr[j];
10
11     sel=sel*temp;
12   }
13 }
```

Dieses Beispiel zeigt stellvertretend für etliche Stellen der CPU-Implementierung, dass während der Ausführung Konstanten in die Berechnung mit einfließen müssen (in diesem Falle die Konstante '1'). Dies kann für Startwerte von Iterationen oder für Negations-Operationen erforderlich sein. Durch die Public-Key-Eigenschaft des unterliegenden Kryptosystems kann dies aber leicht erreicht werden, da die ausführende Instanz im Besitz des öffentlichen Schlüssels ist und somit beliebige verschlüsselte Konstanten erzeugen kann.

Arithmetisch-Logische Einheit (ALU)

Die ALU bearbeitet zunächst nur Operanden von einem einzigen Bit. Bei der Integration der ALU in das Gesamtsystem ergibt sich im Gegensatz zum Speichermodell durch die Ausweitung eines Speicherwortes auf mehrere Bits eine Abhängigkeit zwischen benachbarten Operandenbits. Bei der Addition muss für die korrekte Bearbeitung eine Statusinformation zwischen den zusammengeschalteten 1-Bit ALUs weitergereicht werden, nämlich der Übertrag der einzelnen Additionsstufen. Ein erstes Interface zu einer 1-Bit ALU könnte demnach wie folgt definiert werden:

$$(res, carry) \leftarrow ALU1(\overline{opcode}, op1, op2, carry)$$

Um nun Maschinenworte von n Bits verarbeiten zu können, werden n Instanzen der definierten ALU1 in Reihe geschaltet, sodass sie einen Serienaddierer[8] ergeben, bei dem der Übertrag der Reihe nach von der untersten in die oberste Additionsstufe durchgereicht wird. Dadurch besitzt dieses Rechenwerk lineare Laufzeit im Bezug auf die Operandenlänge. Natürlich existieren weitaus effizientere Addierwerke wie beispielsweise Paralleladdierer mit Übertragsumleitung[9], Paralleladdierer mit Übertragsvorausberechnung[10] oder auch das von-Neuman-Addierwerk, bei denen im günstigsten Fall logarithmische Laufzeit erzielt werden kann. Dies trifft jedoch nur auf eine quasi-parallele Ausführung des wesentlich erhöhten Schaltungsaufwandes in einer tatsächlich elektrisch verbundenen und geschalteten Implementierung zu. Da das hier vorgestellte Konzept zunächst auf einer seriellen Software-Simulation beruht, kann durch komplexere Schaltungen in aller Regel keine Ausführungsgeschwindigkeit gewonnen werden, da hier die Ausführungszeit direkt und linear von der Anzahl der verarbeiteten Gatter abhängt. Die so entstandene n-Bit ALU kann durch das erweiterte Interface beschrieben werden:

$$(\overline{res}, carry, zero) \leftarrow ALU(\overline{opcode}, \overline{op1}, \overline{op2}, carry)$$

[8] auch Ripple-Carry Adder
[9] auch Carry-Skip Adder
[10] auch Carry-Look-Ahead Adder

In der Implementierung werden die Maschinenworte beginnend beim LSB[11] bis zum MSB[12] bearbeitet. Den Übertrag zur untersten Addierstufe bildet das System-Übertragsflag, der Ergebnisübertrag der höchsten Addierstufe wird wieder ins System-Übertragsflag geschaltet. Weiterhin ist die ALU potenziell die Quelle vieler anderer Steuerungssignale, wie dem Zero-Flag, dem Overflow-Flag oder dem Sign-Flag, die sich zur Steuerung des Programmablaufs eignen. Das Zero-Flag signalisiert, wenn das Ergebnis der letzten arithmetischen Operation numerisch 0 war. Dadurch können kontextabhängig weitere Operationen ausgelöst oder Programmsprünge initiiert werden. Vergleichsoperationen von Rechenwerken werden ebenfalls als arithmetische Operationen implementiert (als temporäre Subtraktion), sodass das Zero-Flag beispielsweise auch einen positiven Vergleich (Gleichheit) signalisiert. Durch die Eigenschaft, den Systemzustand abzubilden und die sich daraus ergebende Eigenschaft, auf den Programmablauf Einfluss zu nehmen, sind die Flags ein wichtiges Bindeglied zur Control-Unit. Das folgende Code-Fragment in Listing 5.4 zeigt die Implementierung der n-Bit ALU.

Listing 5.4: n-Bit ALU

```
1 (res,carry) ALU(opcode,op1,op2,carry)
2 {
3   for(i = 0..n) //little endian operands
4   {
5     (res[i],carry,zero)=ALU1(opcode,op1[i],op2[i],carry);
6   }
7 }
```

Die Performance der ALU-Implementierung für unterschiedliche Operandengrößen ist in Abbildung 5.3 dargestellt.

Operandengröße	$\lambda = 384$	$\lambda = 512$	$\lambda = 768$	$\lambda = 1024$
8 Bits	0.07 s	0.08 s	0.09 s	0.1s
16 Bits	0.14 s	0.16 s	0.19 s	0.22 s
32 Bits	0.28 s	0.32 s	0.38 s	0.45 s

Tabelle 5.3: ALU Laufzeiten

[11] least significant (geringstwertiges) Bit
[12] most significant (höchstwertiges) bit

Control-Unit (CU)

Die Steuereinheit wurde von ihrer Bedeutung her bereits beschrieben. Dieser Abschnitt erläutert im Detail die Funktionsweise der für diese Arbeit implementierten CU. Dazu werden zunächst die einzelnen Schritte anhand des Pseudo-Codes in Listing 5.5 erläutert.

Listing 5.5: Steuereinheit als Zustandsautomat

```
1  CONTROL()
2  {
3          register ac = 0; //accumulator
4          register pc = 0; //program counter
5          flag carry = 0;
6          flag zero = 0;
7          extern flag brk = 0;
8
9          while(!brk)
10         {
11           temp = MAr(pc); //immediate addressing
12           opi = GETOPERAND(temp);
13           cmd = GETOPCODE(temp);
14
15           temp2 = MAr(opi); //absolute addressing
16           opa = GETOPERAND(temp2);
17
18           //ALU operations
19           (ac_new_i,carry_new_i) = ALU(0,ac,opi); //imm
20           (ac_new_a,carry_new_a) = ALU(0,ac,opa); //abs
21
22           //LOAD
23           ac_new_li = opi; //imm
24           ac_new_la = opa; //abs
25
26           //STORE
27           mem_new = ((ac & cmd==ST) | (opa & cmd!=ST))
28           MAw(opi,mem_new);
29
30           //UPDATE AC
31           ac = (ac_new_i & cmd in{OR,XOR,AND,ADD}) |
32                  (ac_new_a &cmd in{ORa,XORa,ANDa,ADDa} |
33                  (ac_new_li & cmd==L) |
34                  (ac_new_la & cmd==La);
35
36           //UPDATE FLAGS
37           carry = (carry_new_i & cmd==ADD) |
38                   (carry_new_a & cmd==ADDa);
39           zero = (zero_new_i & cmd in{{OR,XOR,AND,ADD}) |
40                  (zero_new_a & cmd in{ORa,XORa,ANDa,ADDa})|
41                  (zero_new_li & cmd==L)|
42                  (zero_new_la & cmd==La);
```

```
43
44              //JUMP & BRANCH
45              pc = ((opi & cmd==JMP) |
46                    (opi & cmd==BCC && carry==0) |
47                    (opi & cmd==BZ && zero==1) |
48                    (pc+1 & cmd notin{JMP,BCC,BZ}));
49          }
50  }
```

Erläuterung

Zeilen 3-7 Die Arbeitsregister des Zustandsautomaten werden initialisiert. Eine von 0 abweichende Startadresse kann hier festgelegt werden. Das Unterbrechungssignal `brk` ist als externe Quelle modelliert.

Zeile 9 Die Control Unit läuft gewissermaßen in einer Endlosschleife, solange kein Unterbrechungssignal anliegt.

Zeile 11 Die Fetch-Phase beginnt mit dem ersten Lesen des Speichers an der Programmzähleradresse. Dies bildet gleichzeitig die Adressierungsart *unmittelbar* (immediate) für das gelesene Datum ab.

Zeile 12 Der geladene Operand wird als `opi` mit der Hilfsfunktion `GETOPERAND` aus dem Speicherleseregister in das Datenregister für die unmittelbare Adressierung geladen.

Zeile 13 Der geladene Operator wird mit der Hilfsfunktion `GETOPERATOR` aus dem Speicherleseregister in das Befehlsregister übernommen.

Zeile 15 Die Fetch-Phase wird mit einem Speicherzugriff mit der Adresse `opi` fortgesetzt.

Zeile 16 Der gelesene Operand wird als `opa` aus dem Speicherleseregister in das Datenregister für die direkte Adressierung geladen.

Zeile 19 In der Execute-Phase wird die ALU zur Berechnung des arithmetischen Ergebnisses des unmittelbar adressierten Operanden aufgerufen. Ebenso werden die Flags für das Ergebnis ermittelt.

Zeile 20 Das ALU-Ergebnis für die absolute Adressierung wird samt Flags für diesen Operanden ermittelt.

Zeilen 23-24 In der Load-Phase werden die unmittelbar und direkt adressierten Operanden temporär in Hilfsregistern gespeichert.

Zeile 27 Das Argument für den schreibenden Speicherzugriff in der Store-Phase wird gemäß der Gleichung 5.1 ermittelt.

Zeilen 31-34 Der Akkumulator wird anhand der temporären Ergebnisse und des tatsächlich anliegenden Opcodes neu geladen.

Zeilen 37-42 Die Flags werden ebenfalls gemäß des vorherigen Punktes gesetzt.

Zeilen 45-48 Abschließend wird das Befehlszählregister anhand der temporären Ergebnisse und der neu gesetzten Flags auf den als nächstes zu ladenden Befehl gesetzt.

Die Steuereinheit nutzt die Hilfsschaltkreise `GETOPERAND(word)` und `GETOPCODE(word)`, um ein Speicherwort in seine Daten- und Befehlsbestandteile zu teilen. Dies kommt einer kombinierten von Neuman/Harvard-Architektur gleich und vereinfacht wesentlich den Zugriff auf einen kompletten Befehl mit Operand in einem einzigen Speicherlesezyklus. Das in der Beispielarchitektur implementierte Instruction Set ist im Anhang in Tabelle A.1 zusammengefasst und wird dort im Abschnitt A im Detail erklärt.

Wie im Pseudocode der Steuereinheit gezeigt, greift jeder Maschinenzyklus dreimal auf den Speicher zu, um gemäß des *Oblivious Access*-Paradigma das Zugriffsmuster zu verbergen:

- die erste Operation liest das Wort *opi* an der aktuellen Stelle des Programmzählers *pc*
- der Operand des geholten Wortes ist eine potenzielle Ladeadresse, deren Inhalt als alternativer operand *opa* geladen wird
- der Operand *opi* ist gleichfalls eine potenzielle Speicheradresse für die *opa* oder reg Werte

In der Performance der Steuereinheit summieren sich prinzipiell die Laufzeiten der abhängigen Unterkomponenten, die während eines Maschinenzyklus'

Speicherzeilen	$\lambda = 384$	$\lambda = 512$	$\lambda = 768$	$\lambda = 1024$
8	5 s	5 s	6 s	7 s
16	11 s	12 s	14 s	16 s
32	35 s	37 s	42 s	49 s
64	39 s	42 s	48 s	55 s
128	97 s	104 s	117 s	137 s
256	119 s	123 s	144 s	166 s

Tabelle 5.4: CPU Zykluszeit (pro Maschinenzyklus)

zur Anwendung kommen. Die Laufzeitmessungen auf Basis der Softwareimplemenierung (C) und der Testplattform (Intel Core 2 Duo 2.4, Mac OS X) werden in der Tabelle 5.4 zusammengefasst.

5.5.3 Werkzeuge

Die Implementierung des Konzeptes umfasst einen prototypischen Assembler, der Maschinencode gemäß der im Anhang A definierten Befehlssatz-Architektur generiert. Es handelt sich dabei um einen 2-Pass Assembler mit symbolischen Adressen. Die beiden Assembler-Direktiven `INITAC <data>` und `INITPC <addr>` steuern die Initialisierung des Akkumulators und die Startadresse des Programms.

5.6 Hardware-Implementierung

Ein naheliegender Ansatz zur Leistungssteigerung der relativ begrenzten Leistungsfähigkeit der vorgestellten allgemeinen, prozessorbasierten Lösung zur Ausführung homomorph verschlüsselter Programme ist eine Implementierung in Hardware. Das konkrete Ziel eines solchen Vorgehens ist die Beschleunigung der in der seriellen Softwaresimulation systembedingt nacheinander ausgeführten Schaltelemente vor allem beim Speicherzugriff. Die Leistungssteigerung basiert also auf einer massiv parallelen Ausführung der Schaltungselemente, wobei zunächst die Umsetzung auf einem FPGA empfohlen wird. Dies ist bereits als Anschlussarbeit an diese Arbeit geplant und wird im folgenden Un-

terabschnitt als Umsetzungsskizze aus [9] entnommen zusammengefasst.

5.6.1 Umsetzungsskizze

Das verwendete homomorphe Kryptographieschema erlaubt die Anwendung der logischen Operationen AND sowie XOR auf die verschlüsselten Daten. Diese Operationen werden im Folgenden zur Differenzierung *Krypto-AND-* und *Krypto-XOR-Operation* genannt. Analog heißt ein verschlüsseltes Bit *Kryptobit*. Die Gatter, die eine Krypto-AND- bzw. eine Krypto-XOR-Operation realisieren, heißen *Krypto-AND-* bzw. *Krypto-XOR-Gatter*. Die kombinatorischen Teile des virtuellen Prozessors sind aus diesen Basisgattern zusammenzusetzen. Es ist zu beachten, dass ein Kryptobit die Bitbreite des Chiffretextes besitzt. Diese bewegt sich für eine realisitsche Anwendung in der Größenordnung 1024 Bits. Ein Prozessor, der ein verschlüsseltes Programm auf verschlüsselten Daten ausführt (*Kryptoprozessor*), kann dadurch erzeugt werden, dass in einem bestehenden Prozessor für unverschlüsselte Programme und unverschlüsselte Daten aufgebaut aus AND- und XOR-Gattern, diese durch die entsprechenden Kryptogatter ausgetauscht werden. Zusätzlich müssen zur Reduktion des akkumulierten Rauschens in regelmäßigen Abständen Recrypt-Einheiten in den Datenfluss zwischen die Kryptogatter eingefügt werden. Das Ergebnis wäre eine konventionelle Prozessorimplementierung, bei der alle Einheiten komplett parallel realisiert sind. Diese Einheiten sind aus Grundgattern aufgebaut, welche nur eine geringe Komplexität von wenigen Transistoren aufweisen. Ein Krypto-AND-Gatter jedoch ist ein Addierer und ein Krypto-XOR-Gatter ist ein Multiplizierer mit angefügter Division zur Restberechnung jeweils mit Ganzzahlen der Bitbreite des Schlüssels als Operanden. Die Recrypt-Einheit selbst ist auch aus Addierern, Multiplizierern und Dividierern dieser Bitbreite aufgebaut. Schon für eine sehr einfach geartete Prozessorarchitektur, deren Kosten in Grundgattern möglichst gering ausfällt, bedeutet das beschriebene Ersetzungsschema eine so immense Explosion der Kosten bzgl. der erforderlichen Transistoren und der erforderlichen Chipfläche, dass jeglicher verfügbare Rahmen einer physikalischen Realisierung gesprengt wird. Weiterhin wäre der Kryptoprozessor auf eine maximale Schlüssellänge limitiert. Wünschenswert dagegen ist eine dynamisch veränderbare Schlüssellänge.

Der gangbare Weg für eine Implementierung ist wie auch bei einer Software-

Simulation eine Serialisierung des Kryptoprozessors. Abhängig von den verfügbaren Ressourcen, werden die einzelnen Funktionseinheiten (FE) mehrfach implementiert. Ein einzelner Takt des virtuellen Kryptoprozessors wird dann durch die sequenzielle Mehrfachnutzung der wenigen parallel vorhandenen FE simuliert. Es ist also eine Simulationsarchitektur zu erarbeiten, welche in der Hardwarebeschreibungssprache VHDL synthesefähig umgesetzt wird. Diese Simulationsarchitektur soll derart skalierbar sein, dass zusätzliche Ressourcen in einer erhöhten genutzten Parallelität und damit in einer kürzeren Ausführungszeit resultieren - d. h. in einer kleineren Anzahl von wirklichen Takten pro virtuellem Takt. Dabei muss die Kompatibilität des simulierten, virtuellen Kryptoprozessors zwischen allen Ausprägungen der Simulationsarchitektur gewährleistet werden. Hierzu wird die Ablaufsteuerung programmierbar gestaltet. Eine Sequenz wird durch eine Abfolge von *Mikroinstruktionen* realisiert und bildet eine Instruktion des Kryptoprozessors - eine *Makroinstruktion*. Das Verhältnis vom virtuellen Kryptoprozessor zu Simulationsarchitektur ist in Abbildung 5.5 dargestellt. Eine Erweiterung der zu Grunde liegenden Simulationsarchitektur wird durch eine Adaption der Gesamtheit der Mikroinstruktionen - dem *Mikrocode* - nutzbar gemacht. Die Granularität und benötigte Flexibilität der Mikroinstruktionen, welche mit der Komplexität der Steuerung des Datenflusses einher geht, soll im Projekt erforscht werden. Weiterhin sind die Auswirkung der relativen Anzahl der unterschiedlichen FE zueinander auf deren mögliche Auslastung und der Gesamtanzahl von FE auf die Laufzeit zu beantwortende Fragestellungen.

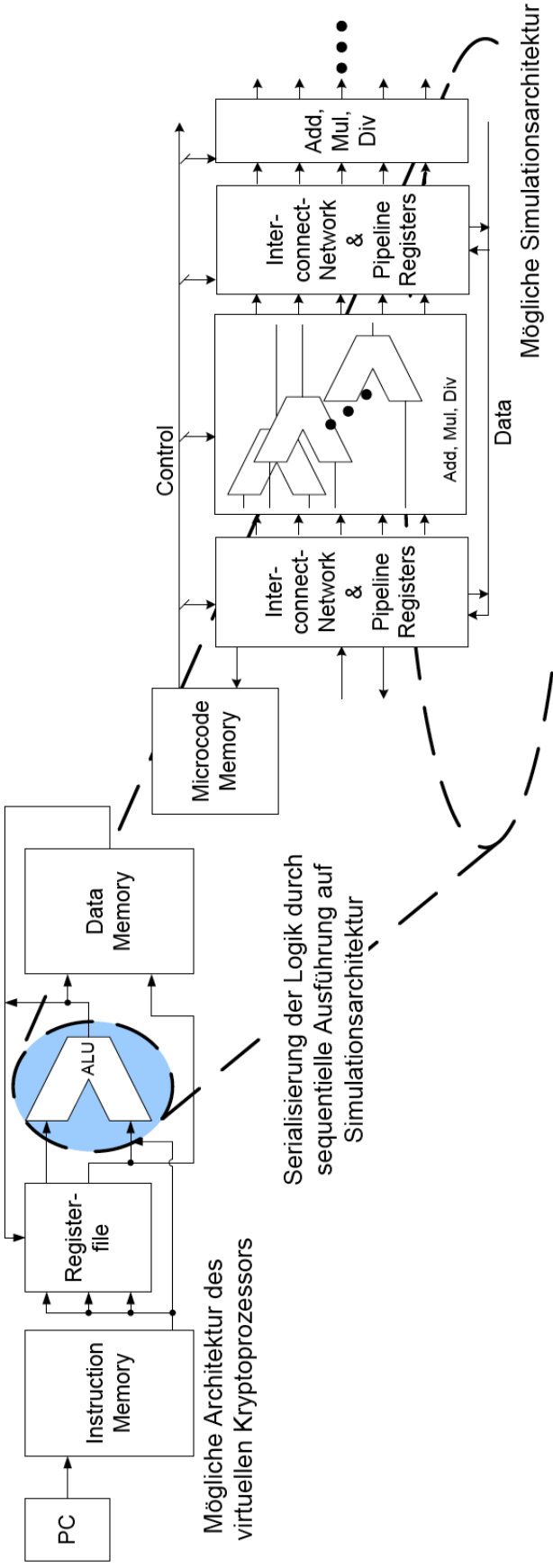


Abbildung 5.5: Architektur des virtuellen Kryptoprozessors und dessen Abbildung auf die Simulationsarchitektur [9]

Neben der Serialisierung des Kryptoprozessors werden auch die Additionen, Multiplikationen und Divisionen der großen Ganzzahlen serialisiert. Sie werden in Subwörter mit fester Bitbreite zerlegt, wobei alle Bits eines Subworts parallel und eine variable Anzahl von Subwörtern seriell verarbeitet wird. Dadurch kann die Schlüssellänge ohne Obergrenze als Vielfaches der Subwortbitbreite zur Laufzeit gesetzt werden. Größere Schlüssellänge wirken sich jedoch negativ auf die Laufzeit aus. Die Erarbeitung einer günstigen FE jeweils für die Addition, Multiplikation und Division sind Forschungsgegenstand des Projekts. Weitere damit verbundene Herausforderungen stellen die Anbindung an den Speicher, die Interkonnektivität jeweils zwischen den einzelnen FE und der internen Register sowie das Zusammenspiel des internen Pipelining der FE mit dem Pipelining auf höherer Ebene und der Serialisierung des virtuellen Kryptoprozessors dar.

Eine möglichst einfache Struktur des virtuellen Kryptoprozessors und das damit verbundene, spezifizierte Verhalten auf die verfügbaren, eingegebenen Instruktionen, genannt *Instruktionssatzarchitektur* (ISA), wird zu einer kleineren Anzahl von wirklichen Takten pro virtuellem Takt führen. Jedoch wird sich auch die Anzahl der benötigten Kryptoinstruktionen für ein auszuführendes Programm erhöhen. Außerdem ist eine Auswirkung auf die konsumierte Energie zu erwarten. Das Optimum dieses Trade-Offs ist anwendungsabhängig. Zur Exploration dieses Entwurfsraums sollen anhand typischer Benchmarks sowie einer Beispielanwendung verschiedene ISA evaluiert werden. Ein zu implementierender Instruktionssatzsimulator (ISS) simuliert das funktionale Verhalten des Kryptoprozessors in Software. Basierend auf ersten Ergebnissen aus der Erarbeitung der Simulationsarchitektur der FEs sollen Modelle aufgestellt werden, welche bei vorgegebenen Kosten und auszuführendem Programm im ISS zur Schätzung von Laufzeit und Energieverbrauch verwendet werden können. Diese Untersuchung kann bereits vor der kompletten Fertigstellung der Implementierung der Simulationsarchitektur nach Erstellung des ISS und der Modelle frühzeitig im Projekt erfolgen. Die Programmierbarkeit der Simulationsarchitektur mit Hilfe des Mikrocodes ermöglicht, den bezüglich Anwendung und Optimierungsziel optimalen ISA zu realisieren. Für die Evaluation des ISA ist es nicht erforderlich, dass die verarbeiteten Daten oder das auszuführende Programm verschlüsselt sind, noch müssen andere Maßnahmen

zum Schutz gegen das Ausspähen getroffen werden, wodurch die Ausführungszeit herabgesetzt werden kann. Diese Merkmale werden jedoch für die funktionale Verifikation der Hardware-Umsetzung des Kryptoprozessors benötigt und sollen deshalb in einem weiteren ISS implementiert werden.

Da dem Kryptoprozessor zur Ausführungszeit eines verschlüsselten Programms lediglich der öffentliche Schlüssel vorliegt, können wegen der verschlüsselten Adressen keine direkten Speicherzugriffe vorgenommen werden. Das Ergebnis eines Zugriffs wird also durch eine logische Funktion über den Adressbits aller Speicherzeilen, den Datenbits der jeweiligen Speicherstellen und den Bits der zuzugreifenden Adresse gebildet. Die Funktion entspricht dabei einem binären Demultiplexer. Durch die Tatsache, dass der Kryptoprozessor zum Speicherzugriff zwangsweise alle verschlüsselten Speicherelemente lesen und in der Zugriffsfunktion verarbeiten muss, entsteht gleichzeitig ein Zugriffsverfahren, bei dem je zwei Zugriffe von außen nicht voneinander zu unterscheiden sind. Dieses Prinzip lässt sich auf eine *Oblivious Random Access Machine* (ORAM [31, 32, 50]) reduzieren.

Der Kryptoprozessor verarbeitet die ausgeführte Makroinstruktionsfolge in einem durch ihn interpretierbaren Format. Diese *Maschinensprache* stellt aber keine geeignete Form zur Programmierung seitens des Menschen dar. Praktisch kann der Kryptoprozessor als nicht programmierbar gelten. Deshalb wird ein *Assembler* benötigt, der eine menschen-lesbare Darstellung der Makroinstruktionen, die *Assemblersprache*, in die Maschinensprache übersetzt. Der Assembler ist spezifisch für eine ISA, sodass ein Algorithmus für jede ISA neu geschrieben werden muss. Höhere Abstraktionsgrade bieten Hochsprachen wie C, C++ oder Fortran, die erlauben, Quellcode in kompakterer und verständlicherer Form unabhängig von der ISA zu schreiben. Ein *Compiler* übersetzt unter Kenntnis der ISA den Hochsprachencode in Assemblercode. Um den Kryptoprozessor praktisch benutzbar zu machen, ist eine Verarbeitungskette, die einen Assembler und Compiler beinhalten zum automatischen Übersetzen von Hochsprachen in Maschinensprache unerlässlich. Weitere erforderliche Werkzeuge bei der Anwendungsentwicklung sind ein Debugger zum schrittweisen Nachvollziehen des ausgeführten Quellcodes und ein Packer, der das erzeugte Programm in Maschinensprache und die Eingangsdaten in ein verschlüsseltes Speicherabbild für den Kryptoprozessor überführt. Für das Auslesen der

Ergebnisse wird ein Depacker benötigt, der das ausgelesene Speicherabbild entschlüsselt und die Daten extrahiert.

5.6.2 Vorschlag zur Umsetzung

Der folgende Unterabschnitt ist ein Vorschlag für ein konkretes Arbeitsprogramm zur Umsetzung der vorstehenden Skizze.

A. Prozessor zum Ausführen verschlüsselter Maschinensprache

- Konzeption einer skalierbaren, programmierbaren Simulationsarchitektur zum seriellen Addieren, Multiplizieren und Dividieren: Die in Bezug auf die Anzahl der Addierer, Multiplizierer und Dividierer sowie der Subwortbitbreite parametrisierbare Simulationsarchitektur soll konzipiert werden. Durch die auf diese Weise erzielbare Skalierbarkeit wird die Abbildung auf FPGAs mit unterschiedlichen Mengen an Ressourcen ermöglicht.

- Speicheranbindung: Die für die Gesamtsicherheit des Systems wichtigen Speicherzugriffsmuster, aus denen keine Informationen ausgespäht werden können, soll umgesetzt werden. Die Anbindung an den Speicher ist entsprechend der benötigten Bandbreite der Simulationsarchitektur zu entwerfen. Es liegt damit eine gegenseitige Abhängigkeit mit dem vorausgehenden Arbeitspaket vor.

- ISA-Evaluation: Ein generischer ISS für verschiedene ISAs ist zu implementieren. Anhand der Simulationsarchitektur werden Laufzeit und Energiebedarf einer Makroinstruktion modelliert. Auf Basis dieser Modelle werden Schätzer in den ISS integriert, die eine quantitative Bewertung der ISA ermöglichen. Einzelne Schätzungen werden später zur Validierung des Verfahrens kritisch mit tatsächlich ermittelten Werten verglichen.

B. Toolchain (basierend auf GNU-Toolchain)

Aufgrund der verschiedenen zu evaluierenden, unterschiedlichen ISA dürfen die in den folgenden drei Unterpunkten beschriebenen einzelnen Werkzeuge nicht

fest für eine ISA entworfen werden. Stattdessen müssen Generatoren implementiert werden, welche die veränderlichen Teile dieser Werkzeuge abhängig von der gegebenen ISA automatisch erstellen.

- Binutils: Diverse Werkzeuge inklusive des Assemblers, die zum Übersetzen von Assemblercode in Maschinensprache erforderlich sind, werden realisiert. Zusätzlich wird das anschließende Zusammenführen von Teilen (linken) ermöglicht sowie die Konvertierung der binären Maschinensprache in verschiedene Darstellungen.

- Compiler (basierend auf GNU Compiler Collection): Der Teil des Compilers, der die Übersetzung der internen Beschreibung des Programmablaufs abhängig von der jeweiligen ISA in Assemblersprache übernimmt (das Backend), soll implementiert werden.

- Debugger (basierend auf GNU Debugger): Der Debugger zum schrittweisen Nachverfolgen der ausgeführten Befehle sowie des internen Zustands des virtuellen Kryptoprozessors soll implementiert werden. Dabei wird das Programm auf einem in Software simulierten Kryptoprozessor oder aber auf dem Experimentier-/Demonstrationssystem ausgeführt. Der interne Zustand des Prozessors soll per JTAG-Schnittstelle fortlaufend ausgelesen werden können.

- Encrypt- und Decrypt-Werkzeuge: Der Packer und Depacker werden umgesetzt. Zusätzlich Werkzeuge zum Hoch- bzw. Herunterladen der Speicherabbilder.

5.7 Ausgangssignalisierung

Ein inhärentes Problem des in diesem Kapitel dargestellten Ansatzes einer verschlüsselten CPU bzw. eines verschlüsselten Containers ist die systembedingte *kryptografische Abgeschlossenheit*. D. h. gemäß der Beweisführung zur Sicherheit der kryptografischen Basis (siehe Abschnitt 3.3.3) kann ein im Innern des Containers ablaufendes Programm keine unverschlüsselten Ressourcen verwenden und somit keine unverschlüsselten Signale generieren.

Diese Fähigkeit ist jedoch zur Nutzung in einem realen Umfeld wünschenswert, da sich hieraus eine Fülle von erweiterten Anwendungsfällen erschließen lässt. In der Regel muss der Container mindestens dann ein erkennbares Signal generieren, wenn das ablaufende Programm terminiert. Andernfalls könnte der ausführenden Ressource nicht mitgeteilt werden, wann die Arbeit eingestellt werden kann. Außerdem lässt sich der Sachverhalt sehr anschaulich am Beispiel einer industriellen Steuerungsanlage, die einen Roboterarm kontrolliert, für die Hardware-Implementierung darstellen: die für die Steuerungssoftware erforderlichen Standardwerte sind zusammen mit der Software selbst bereits im verschlüsselten Container auf der Anlage enthalten. Mess- und Sensorwerte werden von außen eingespeist, um die Steuerung zur Laufzeit zu parametrisieren. Die berechneten Steuerwerte, die üblicherweise von Signalen auf einem Steuerbus durch Treiberbausteine in Schaltströme umgewandelt werden, müssen jedoch im Klartext vorliegen, um der angesteuerten Hardware korrekte Befehle erteilen zu können. Die Gefahr, die sich aus dieser Anforderung ergibt, ist natürlich die Kompromittierung des gesamten Systems, wenn die Entschlüsselbarkeit eines einzelnen Elementes gewährleistet werden soll. Dieser Abschnitt weist nach, dass ein solcher Mechanismus auf Basis der verfügbaren Kryptosysteme die Sicherheit der verschlüsselten Maschine gefährdet.

Zur Verdeutlichung dieser Problematik dient das in diesem Kapitel bereits im Ansatz eingeführte Szenario einer geheimen Berechnung zwischen zwei Parteien: Partei A will eine Berechnung delegieren und besitzt den dazu erforderlichen Programmcode und die zu verarbeitenden Daten. Vor dem Transfer der Implementierung seiner Software und der Daten überführt A diese Informationen in ein verschlüsseltes Maschinenimage M. Diese virtuelle Maschine ist durch einen Startzustand S_0 gekennzeichnet, welcher das initiale Maschinenimage und die Inhalte aller Maschinenregister und Flags enthält. Die ausführende Einheit B hat Zugriff zu der öffentlich verfügbaren Implementierung der virtuellen Maschine zur Ausführung des verschlüsselten Maschinenimages und lädt die verschlüsselten Inhalte des Images S_0 an die erforderlichen Positionen der virtuellen Maschine. B führt das verschlüsselte Image in der virtuellen Maschine aus, indem die Hauptschleife des als Zustandsautomat implementierten Maschine eine bestimmte Anzahl von Durchläufen aufruft. B liefert nach der Ausführung a) das gesamte Maschinenimage oder b) eine definierte

Untermenge des Maschinenimages an A zurück. A entschlüsselt das Ergebnis der Berechnung.

Die Analyse einer durch B entschlüsselbaren Information aus dem verschlüsselten Maschinenimage M erfolgt im Honest-But-Curious Angreifermodell. Das bedeutet, dass B sich grundsätzlich an das Protokoll zwischen A und B hält, also nicht destruktiv handelt. Allerdings versucht B mehr Informationen aus dem Maschinenimage zu erhalten, als durch A vorgesehen ist.

Theorem. Die verschlüsselte Maschine $\mathcal{M}$ mit einem öffentlichen, beschränkten Entschlüsselungsorakel $\mathcal{E}$ kann aktiv Signale generieren, die außerhalb des verschlüsselten Containers durch eine öffentliche Funktion F entschlüsselt oder decodiert werden kann. Das Orakel $\mathcal{E}$ ist auf die Weise beschränkt, dass es nur die durch $\mathcal{M}$ dafür vorgesehenen Chiffretexte entschlüsseln kann. Dies impliziert, dass die semantische Sicherheit jedes unterliegenden Public-Key-basierten Kryptosystems gebrochen werden kann. Darüberhinaus kann ein Angreifer $\mathcal{A}$ den gesamten verschlüsselten Maschinenzustand mit in $O(m \cdot n^3)$ entschlüsseln.

Beweis. Die verschlüsselte Maschine $\mathcal{M}$ mit einem Startzustand S_0 (Speicher, Register und Flags) der Größe n und einer öffentlichen Verschlüsselungsfunktion E kann aktiv Signale innerhalb des verschlüsselten Containers generieren, die außerhalb desselben durch die öffentliche Funktion F entschlüsselt oder decodiert werden können.

Nach der in diesem Kapitel vorgestellten Konstruktion einer verschlüsselten Maschine hat ein Maschinenzyklus einer Maschine mit n Speicherstellen aufgrund des Speicherzugriffsprinzips eine Zeitkomplexität von $O(n)$. Die Maschine generiert innerhalb von m Maschinenzyklen ein außerhalb des verschlüsselten Containers entschlüsselbares oder decodierbares Signal s. Es existiert eine öffentliche Funktion F

$$s \leftarrow F(\mathcal{M}(\mathcal{S}_0))$$

mit konstanter Laufzeit m. Diese Funktion wird im folgenden Experiment eingesetzt:

$Exp\{$

$$
\begin{aligned}
&\forall\ a \in \{0..m-1\} \\
&\quad \forall\ b \in \mathcal{S}_a \\
&\qquad \mathcal{S}_a[b] \leftarrow E(0) \\
&\qquad if(F(\mathcal{M}(\mathcal{S}_a)) = 0 \\
&\qquad\quad \mathcal{S}_a[b] \leftarrow E(1) \\
&\qquad\quad if(F(\mathcal{M}(\mathcal{S}_a)) = 1 \\
&\qquad\qquad return\{a, b\} \\
&\}
\end{aligned}
$$

Im Besitz des gesamten Maschinenzustandes iteriert $\mathcal{A}$ über alle verschlüsselten Positionen von Bitrepräsentationen des Maschinenzustandes $\mathcal{S}_m$ und fügt die verschlüsselte Repräsentation eines ihm bekannten Bitwerts ein. Liefert die Funktion F den eingefügten Bitwert, so kann das Prozedere mit dem anderen Binärwert an der gefundenen Stelle des Maschinenimages durchgeführt werden. Führt auch der zweite Test zum Erfolg, so kann der Vorgang mit jeder Bitrepräsentation des verschlüsselten Maschinenimages wiederholt werden und führt zum entschlüsselten Maschinenimage $\mathcal{D}$

$$\forall b \in \mathcal{S}_0, \mathcal{D}_b \leftarrow F(\mathcal{M}(b)).$$

■

Lösungsansatz

Das Problem der Ausgangssignalisierung beruht auf der Tatsache, dass alle Chiffretexte mit demselben Schlüssel entschlüsselt werden können und somit keine Selektion möglich ist. Die Lösung zu der Problemstellung ist ein homomorphes Kryptoschema, bei dem Chiffretexte mit unterschiedlichen Schlüsseln entschlüsselt werden müssen, die Chiffretexte jedoch untereinander im Bezug auf den zugrundeliegenden Homomorphismus interoperabel sind.

Sei $\mathcal{H}$ ein homomorphes Kryptosystem mit der Verschlüsselungsfunktion $\mathcal{E}$, der Entschlüsselungsfunktion $\mathcal{D}$ und *interoperablen Schlüsselgruppen* $\mathcal{G}_0..\mathcal{G}_n$. Eine Schlüsselgruppe bestehe aus mindestens drei Schlüsseln $\mathcal{K}_0, \mathcal{K}_1, \mathcal{K}_2$, sodass für zwei Klartexte a und b der algebraische Homomorphismus gilt:

$$a' \leftarrow \mathcal{E}(a, \mathcal{K}_0), b' \leftarrow \mathcal{E}(b, \mathcal{K}_1)$$

$$\mathcal{D}(a' \oplus b', \mathcal{K}_2) = a + b, \mathcal{D}(a' \otimes b', \mathcal{K}_2) = a \cdot b$$

Weiterhin gelte Folgendes:

$$\mathcal{D}(a', \mathcal{K}_0) = a, \mathcal{D}(a', \mathcal{K}_1) = \emptyset, \mathcal{D}(a', \mathcal{K}_2) = \emptyset$$

$$\mathcal{D}(b', \mathcal{K}_0) = \emptyset, \mathcal{D}(b', \mathcal{K}_0) = b, \mathcal{D}(b', \mathcal{K}_2) = \emptyset$$

$$\mathcal{D}(a' \oplus b', \mathcal{K}_0) = \emptyset, \mathcal{D}(a' \oplus b', \mathcal{K}_1) = \emptyset, \mathcal{D}(a' \oplus b', \mathcal{K}_2) = a + b$$

$$\mathcal{D}(a' \otimes b', \mathcal{K}_0) = \emptyset, \mathcal{D}(a' \otimes b', \mathcal{K}_1) = \emptyset, \mathcal{D}(a' \otimes b', \mathcal{K}_2) = a \cdot b$$

Ein nach diesem Prinzip operierendes Kryptosystem ist in der Lage, nur bestimmte Chiffretexte mit Hilfe eines speziellen Schlüssels zu entschlüsseln, ohne die Sicherheit der anderen Chiffretexte zu gefährden.

5.8 Ergebnisse

In diesem Kapitel wurde die Konstruktion eines homomorph verschlüsselten Maschinenmodells vorgestellt und im Detail ausgearbeitet. Es dient der vollständigen Abstraktion der Kryptografie und gestattet es, Sicherheitsfunktionen vor der Anwendungsschicht zu verbergen. Dies führt zu einer sinnvollen Unterscheidung zwischen funktionalem Code der Business-Logik einer Anwendung und nicht-funktionalen Anforderungen an Verschlüsselung. Wie die Messungen an den prototypischen Implementierungen zeigen, erfordert die vollständige Abstraktion in einer verschlüsselten Maschine einen erheblichen Mehraufwand zur Berechnung der verschlüsselten Programmfunktionen, sodass unter Verwendung der gegenwärtigen Kryptoschemata eher kleine Funktionen verschlüsselt werden können und dennoch eine vertretbare Leistung erzielen. Um diesem Umstand zu begegnen, wird in diesem Kapitel eine Hardware-Umsetzung auf Basis von FPGA vorgeschlagen, mit der sich eine Beschleunigung erzielen lässt. Außerdem adressiert dieses Kapitel das Problem der Ausgangssignalisierung einer verschlüsselten Maschine. Es wird nachgewiesen, dass mit den verfügbaren Kryptoschemata keine Maschine konstruiert werden kann, die selektiv Informationen von außen verschlüsselbar zur Verfügung stellen kann, ohne den gesamten Inhalt der verschlüsselten Maschine zu gefährden.

Kapitel 6

Homomorph verschlüsselte Hybrid-Systeme

Die vorangegangenen Kapitel haben gezeigt, dass der Ansatz einer homomorph verschlüsselten Maschine technisch möglich ist, die Leistungsfähigkeit in Verbindung mit den zur Zeit verfügbaren Kryptosystemen jedoch das Einsatzgebiet deutlich einschränkt. Allerdings wurde nachgewiesen, dass bei bestimmten Problemstellungen und angepasster Protokollstruktur eine für reale Anwendungen akzeptable Performance erzielt werden kann, ohne die Sicherheit der resultierenden Lösung wesentlich negativ im Vergleich zur vollständig verschlüsselten Maschine zu beeinflussen. Um diese aus unverschlüsselten und verschlüsselten Programmteilen zusammengesetzten Algorithmen formal zu fassen, führt dieses Kapitel abschließend den Begriff der *hybriden Systeme* ein. Diese stellen einen wesentlichen Ansatzpunkt für weitergehende Forschung im Bereich der praktisch anwendbaren homomorphen Kryptografie dar. Die in diesem Kapitel dargestellten Ergebnisse wurden als studentische Arbeit unter meiner Betreuung erstellt.

Hybride Algorithmen sind deshalb eine valide Alternative zur vollständigen homomorphen Verschlüsselung, weil in der Praxis oftmals Daten verarbeitet werden, die entweder öffentlich zugänglich sind oder in dem speziellen Fall nicht besonders schützenswert sind. Diese Daten können also in einem *unverschlüsselten* Teil des Algorithmus' verarbeitet werden, ohne dass die Sicherheit des Gesamtsystems gefährdet wird.

6.1 Definition

Dieser Abschnitt führt eine formale Definition des Begriffs der hybriden Algorithmen im Sinne der hier betrachteten homomorphen Verschlüsselung ein.

Definition 6.1.1. Seien x und $\mathfrak{x}$ der unverschlüsselte und der verschlüsselte Teil eines Algorithmus', C ein Schaltkreis im homomorph verschlüsselten Raum, f eine (totale) Funktion und pk der öffentliche Schlüssel des unterliegenden Kryptosystems.

Ein *hybrider Algorithmus* ist dann definiert als ein Algorithmus, der durch eine Turing-Maschine mit Eingabe $(x, \mathfrak{x}, f, \mathsf{pk})$ wie folgt berechnet werden kann:

1: $(C, y) \leftarrow f(x)$ // unverschlüsselter Teil, der Eingabe für $\mathfrak{x}$ erzeugt
2: $\mathfrak{out} \leftarrow \mathsf{Evaluate}(C, (\mathfrak{x}, \mathsf{Encrypt}(y, \mathsf{pk})), \mathsf{pk})$ // verschlüsselter Teil
3: **return** $\mathfrak{out}$

Für real einsetzbare Anwendungen ist die Funktion f nach oben polynomiell in der Größe der Eingabe begrenzt. Das bedeutet, dass C ebenfalls polynomiell in der Größe der Eingabe begrenzt ist, da C durch die Funktion f konstruiert wird. Aus den Gründen, die im Kapitel 5 für die eingeschränkte Leistungsfähigkeit der verschlüsselten CPU aufgezeigt werden, ist der verschlüsselte Teil des Algorithmus' wesentlich rechenintensiver als der unverschlüsselte Teil.

6.2 Analyse

Der Zweck eines hybriden Algorithmus' ist der Schutz der Vertraulichkeit der verschlüsselten Eingabedaten $\mathfrak{x}$ und der verschlüsselten Ausgabedaten $\mathfrak{out}$. Aus der Definition geht hervor, dass die Ausgabe $\mathfrak{out}$ das Resultat der Ausführung der Schaltkreisrepräsentation C über der Eingabe $(\mathfrak{x}, \mathsf{Encrypt}(y, \mathsf{pk}))$ ist. Für die Analyse wird angenommen, dass das unterliegende Kryptosystem gemäß der Analysen in Gentry et al. [27] und Smart et al. [55] sicher ist. Weiterhin wird angenommen, dass die Schaltkreisrepräsentation C gemäß der folgenden Definition nicht *degeneriert* ist:

$$C \text{ ist degeneriert} :\Leftrightarrow \exists i : \forall (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) :$$
$$f_C(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) = f_C(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n). \quad (6.1)$$

In einer nicht degenerierten Schaltkreisrepräsentation trägt jeder Teil der Eingabe zur Ausgabe bei. Das bedeutet, dass das verschlüsselte Ergebnis $\mathfrak{out}$ nicht aus der Struktur der Schaltkreisrepräsentation und der unverschlüsselten Ausgabe y geschlossen werden kann. Der entsprechende Beweis, dass weder $\mathfrak{x}$ noch $\mathfrak{out}$ aus der Eingabe x und der Ausgabe y abgeleitet werden kann, muss für eine konkrete Anwendung unter Berücksichtigung der Konstruktion der Schaltkreisrepräsentation, der Struktur der Eingabeparameter und der Charakteristika der Funktion f erbracht werden. Für eine exemplarische Anwendung von Bloom-Filtern zur Suche mit geheimen Suchargumenten wird eine Analyse in dem folgenden Abschnitt skizziert. Dies ist ein Feld für weitergehende Forschung.

6.3 Hybride Suche mit verschlüsselten Anfragen

Das hier eingeführte Hybrid-System für die Indexsuche beruht neben homomorpher Kryptographie auf Bloom-Filtern, bei denen sowohl die Suchanfragen als auch die Daten auf Hashes reduziert werden, wobei letztere in einer Baumstruktur zur effizienteren Suche geordnet werden. Hierdurch ergibt sich eine logarithmische Zeitkomplexität. Die während eines Suchvorgangs gewählten Pfade des Suchbaums werden im Kryptoraum akkumuliert, um ein Reverse-Engineering der Hashes zu verhindern.

Eine Grundlage für die Suche bilden zählende („counting“) Bloom-Filter. Ein zählender Bloom-Filter ist ein m-Vektor aus $\mathbb{N}^m$. Sei $\big(f_1(x), \ldots, f_n(x)\big)$ ein Tupel von n Hashfunktionen mit $f_i : \Sigma^* \to [0, m] \subset \mathbb{N}$, wobei Σ^* das Arbeitsalphabet ist, aus dem die Suchdaten sowie die Anfrage stammen. Der

Markierungsvektor b ist definiert als

$$b(f(x)) = (\dots, 0, \underbrace{1}_{\text{Stelle } f(x)}, 0, \dots) \in [0,1]^m$$

$$\mathcal{B} : \Sigma^* \to \mathbb{N}^m$$

$$x \mapsto \sum_{i=1}^{n} b(f_i(x)) \qquad \text{Vektoraddition}$$

Als nächstes wird der Bloom-Filter $\mathcal{B}$ auf eine Menge von Eingaben $\{x_1, \dots, x_k\}$ erweitert:

$$\widehat{\mathcal{B}} : \wp(\Sigma^*) \to \mathbb{N}^m \qquad \{x_1, \dots, x_k\} \mapsto \sum_{x \in \{x_1, \dots, x_k\}} \mathcal{B}(x)$$

Dann ist

$$\forall i \in [0, m-1] : \mathcal{B}(x)[i] \leq \widehat{\mathcal{B}}(\{x_1, \dots, x_k\})[i] \Rightarrow x \in \{x_1, \dots, x_k\} \tag{6.2}$$

mit großer Wahrscheinlichkeit, sowie

$$\exists i \in [0, m-1] : \mathcal{B}(x)[i] > \widehat{\mathcal{B}}(\{x_1, \dots, x_k\})[i] \Rightarrow x \notin \{x_1, \dots, x_k\} \quad . \tag{6.3}$$

Die Wahrscheinlichkeit in Gleichung 6.2 hängt von der Wahl der Parameter n und m sowie der Mächtigkeit der Eingabe von $\widehat{\mathcal{B}}$ ab und kann durch Kaskadierung mehrerer Bloom-Filter verbessert werden (Chazelle et al.: *Bloomier Filters* [16]). Um nun in einer geordneten Menge $X = (x_1, \dots, x_n)$ genau den Index i mit $x = x_i$ zu finden wird das Konzept der Bloom-Filter auf einen binären Suchbaum übertragen. Dabei repräsentiert der Wurzelknoten den Bloom-Filter der kompletten Menge X, der linke bzw. rechte Nachfolger enthält den Filter der linken respektive rechten Hälfte von X. Dies wird rekursiv fortgesetzt, bis ein Blatt nur noch ein einzelnes Wort repräsentiert. Durch wiederholtes Überprüfen der Bedingungen in Gleichung 6.2 und 6.3 kann in $\mathcal{O}(\log n)$ eine Menge von Treffern $T = \{\, i \mid \mathcal{B}(x) = \widehat{\mathcal{B}}(\{x_i\})\}$ aufgebaut werden. Diese Menge repräsentiert dann die Treffer der Suchanfrage x.

Hier zeigt sich auch, weshalb der Zusammenhang über eine Wahrscheinlichkeit in Gleichung 6.2 ausreicht: Bezüglich der Inklusion ($\in$) können nur „false positives“ entstehen. Je weiter der Baum traversiert wird, desto weniger Elemente werden jedoch von $\widehat{\mathcal{B}}$ abgebildet bis schließlich $\widehat{\mathcal{B}}$ dem Bloom-Filter

$\mathcal{B}$ eines Blattes entspricht. Letztlich ist die Wahrscheinlichkeit für ein „false positive“ dann:

$$P[\mathcal{B}(x_1) = \mathcal{B}(x_2) \mid x_1 \neq x_2] = \frac{1}{n \cdot m} \tag{6.4}$$

und damit insbesondere unabhängig von der Größe des Baumes.

Bis auf die Tatsache, dass anstatt eines Suchstrings x nur der Hash $\mathcal{B}(x)$ benötigt wurde, wurde in dem Verfahren noch nichts verschlüsselt. Hierfür wird von der Anfrage $\mathcal{B}(x)$ noch ein Zufallsfilter $\mathcal{B}_r$ substrahiert, wobei $\mathcal{B}_r \in [0,1]^m$ an $n \ll m$ zufälligen Stellen mit 1 belegt ist, an denen auch $\mathcal{B}(x)$ mit 1 belegt ist. Dieser Filter $\mathcal{B}_r$ wird mit homomorpher Kryptographie als $\mathcal{B}_\mathfrak{c}$ verschlüsselt und bildet zusammen mit dem überlagerten Suchwort $\mathcal{B}(x)$ die komplette Suchanfrage.

Die Treffermenge T enthält somit auch Treffer, die nur aufgrund von Belegungen aus $\mathcal{B}_r$ Treffer sind. Dieser Bloom-Filter verhält sich somit ähnlich wie eine Menge von „Ablenkungs-Suchwörtern“, in denen die Suchanfrage versteckt wird.

- Genau wie die „Ablenkungs-Suchwörter“ wird auch in diesem Ansatz die Ergebnismenge erweitert. Da in Gleichung 6.2 bei der Überprüfung nur die Indizes relevant sind, die in $\mathcal{B}(x)$ mit 1 belegt sind, besitzt die Subtraktion von $\mathcal{B}_r$ genau diese Eigenschaft und verhält sich damit bezüglich der Vertraulichkeit ähnlich wie eine Menge von „Ablenkungs-Suchwörtern“.

- Weiterhin beinhaltet die Antwort der Suche auch die Treffer der Ablenkung. Beim Durchlaufen des Suchbaumes werden daher durch einen Schaltkreis im Kryptoraum mittels homomorpher Operationen die Ergebnisse markiert, die unabhängig von dem Ablenkungswort zur Suchanfrage passen. Dies ist möglich, da der verschlüsselte Filter $\mathcal{B}_\mathfrak{c}$ ebenfalls vorliegt. Konkret wird folgender Schaltkreis als arithmetischer Ausdruck ausgewertet:

$$\prod_{i \text{ mit } \widehat{\mathcal{B}}(X)[i]=0} (\mathcal{B}_\mathfrak{c}[i] + 1) = \begin{cases} 1 & \text{Treffer} \\ 0 & \text{Ergebnis abhängig von Ablenkungswort} \end{cases} \tag{6.5}$$

  Dieser Schaltkreis besitzt somit nur logarithmische Tiefe abhängig von der Anzahl der Hashfunktionen n. Dies ist im allgemeinen ein Parameter

des Systems, also nicht von der Länge einer konkreten Suchanfrage oder der Menge der zu durchsuchenden Daten abhängig.

Das beschriebene Markierungsverfahren ist wichtig für das Hybrid-System, da ansonsten zusätzlich die Bloom-Filter aller Ergebnisse mit übertragen werden müssten und somit die Effizienz beeinträchtigt würde.

Zusammenfassend erfüllt der Filter $\mathcal{B}_r$ also zwei sicherheitstechnische Funktionen:

1. Es macht die Rekonstruktion von x aus $\mathcal{B}(x)$ unmöglich, da aus dem Bloom-Filter Einträge gelöscht wurden.

2. Es erschwert den Rückschluss von der Ergebnismenge auf die Suchanfrage, da erstere einen großen Anteil an Ergebnissen beinhaltet, die nur aufgrund des Ablenkungswortes als Ergebnisse aufgenommen wurden.

Ferner wird mittels $\mathcal{B}_\mathfrak{c}$ das Markieren der korrekten Ergebnisse im Kryptoraum ermöglicht.

2006 veröffentlichte Bruce Schneier eine Kritik an der in [38] beschriebenen Methode[1]. Der hier beschriebene Ansatz teilt diese Schwachstellen jedoch nicht:

- Da dieser Ansatz auf Bit-Ebene operiert, wird kein (begrenztes und vorhersehbares) Wörterbuch für die Ablenkungswörter benötigt.

- Da nur eine Suchanfrage gesendet wird, vergrößert sich die Laufzeit nicht um einen Faktor n der Anzahl der Ablenkungsanfragen. Ferner sind dadurch statistische Analysen basierend auf der Frequenz der Anfragen nicht möglich.

Damit stellt dieser Ansatz eine Verbesserung gegenüber [38] dar.

Eine Beispielanwendung ist die Suche in einer Gendatenbank. Das zentrale Schutzziel ist die Vertraulichkeit der Suchdaten sowie der daraus generierten Treffer in Form möglicher Diagnosen, letztendlich also potentielle Krankheitsbilder des Patienten. Eingabe ist ein wie oben beschrieben berechneter Bloom-Filter einer Gen-Sequenz eines Patienten sowie der verschlüsselte Filter $\mathcal{B}_\mathfrak{c}$. Die

[1] `http://www.schneier.com/blog/archives/2006/08/trackmenot_1.html`

Suchmaschine durchläuft mit der Anfrage den Suchbaum und konstruiert eine Indexliste. Zudem wird für jeden Index im Kryptoraum markiert, ob es ein Treffer ist oder nicht. Die tatsächlichen Treffer können nur vom Benutzer der Suchmaschine durch Entschlüsselung der Markierungen ermittelt werden.

Dieses Beispiel verdeutlicht das Potential von Hybrid-Systemen in der Anwendung von homomorpher Kryptografie für effiziente geheime Programmausführung.

Kapitel 7

Homomorph verschlüsselte Finanztransaktionen

Dieses Kapitel beschreibt einige weitere homomorph verschlüsselte Protokolle, die zusammengefasst als Basis für elektronischen Zahlungsverkehr mit verdeckten Beträgen eingesetzt werden können. Die folgenden Abschnitte führen in das Thema sowohl des klassischen elektronischen Zahlungsverkehrs als auch in neue elektronische Zahlungssysteme ein, die vorwiegend für den Handel mit digitalen Gütern und für sog. *Micropayment*, also Beträgen im Bereich um 5-10 € Verwendung finden.

Im weiteren Verlauf dieses Kapitels wird ein Bankenszenario beschrieben, in dem die Parteien über eine zentrale Verrechnungs- oder *Clearing*-Stelle Transaktionen in geheimer Höhe durchführen können. Dieses Beispiel fokussiert dabei die Geheimhaltung der Salden der einzelnen Teilnehmer sowie die Geheimhaltung der Transaktionsbeträge vor der Verrechnungsstelle. Die Anonymität der Teilnehmer ist nicht Gegenstand der vorliegenden Betrachtungen, kann aber durch eine Agentur hergestellt werden.

7.1 Bargeldloser Zahlungsverkehr

Im gegenwärtigen Geschäftsverkehr sinkt der Anteil der Bargeldzahlungen im Einzelhandel stetig. Waren im Jahre 1994 laut einer Studie des Einzelhandels-Forschungsinstituts EHI noch 80% der Transaktionen bargeldbehaftet, so sank diese Zahl 2008 auf 61% und 2010 auf 59%. Dabei verlor die Bargeldzahlung

ihre Anteile am gesamten Einzelhandelsvolumen hauptsächlich zugunsten der bargeldlosen Kartenzahlung [53].

Der Vorteil von Bargeld gegenüber dem bargeldlosen Zahlungsverkehr und insbesondere dem klassischen Buchgeld, welches durch Überweisungen oder Lastschriften übertragen wird, ist die erzielbare Anonymität bei Barzahlung. Euro-Bargeld ist das offizielle Zahlungsmittel der Euro-Währungsunion, was bedeutet, dass diese Zahlungsform von jedermann im Euro-Währungsraum in unbegrenzter Höhe mit schuldbefreiender Wirkung akzeptiert werden muss[1]. Außerdem ist Bargeld nach deutschem Recht den Inhaberpapieren gleichgestellt, sodass eine Übertragung des Eigentums praktisch nicht eingeschränkt werden kann und bei einem Geschäft in jedem Falle zustande kommen kann[2].

Da bei der zunehmenden Zahl von Online-Transaktionen natürlich kein Bargeld fließen kann, wird hier hauptsächlich im klassischen bargeldlosen Zahlungsverkehr operiert. Diese gleichsam zunehmende Nachvollziehbarkeit von Geldflüssen belastet aber die oftmals gewünschte Anonymität von Geschäften, weswegen seit geraumer Zeit nach Lösungen gesucht wird, die die Anonymität eines elektronischen Bargeld-Äquivalents bieten.

7.1.1 Bitcoin

Bitcoin ist eine der ersten Implementierungen des Konzeptes der *Kryptowährung*, welches auf einen Bericht von Wei Dai auf der *Cypherpunks* Mailinglist 1998 zurückgeht. Weitere Arbeiten an dem Konzept sind in Nakamoto [46] zu finden. Die Philosophie hinter Bitcoin versteht jedes beliebige Gut, sei es ein physisches Objekt oder auch ein elektronisches Datum, welches zur Bezahlung anderer Güter oder Dienstleistungen in einem möglicherweise geografisch beschränkten oder in einem sozioökonimischen Kontext in irgendeiner Weise akzeptabel ist, als Geld. Dabei setzt Bitcoin eine kryptografische Grundlage zur Generierung und zum Transfer von Geld voraus, statt einer zentralen Einrichtung (Bank) für die Durchführung dieser Aufgaben zu vertrauen. Die Darstellungen in diesem Unterabschnitt sind aus den Quellen *bitcoin.org* und

[1] §14 Bundesbankgesetz

[2] im Speziellen bedeutet das auch, dass bspw. zuvor gestohlenes Bargeld gutgläubig bei einem Geschäft schuldbefreiend akzeptiert werden kann und dann in das Eigentum des Empfängers übergeht

bitcoin.it entlehnt und beschreiben zusammengefasst die zentralen Punkte des Verfahrens.

Erzeugung von Bitcoins

Um einen realen Wert für die Bitcoins zu erzeugen, müssen diese in irgendeiner Weise künstlich verknappt werden. Neue Bitcoins werden daher mit relativ hohem Rechenaufwand und nach einem mathematischen Verfahren erzeugt. Dem Verfahren liegt ein mathematisches Problem zugrunde, dessen Komplexität so angepasst wird, dass die Erzeugung von Bitcoins durch alle am Verfahren teilnehmenden Einheiten konstant ist. Dazu verwenden die Teilnehmer eine Software, deren Parametrisierung für die Erzeugung neuer Bitcoins jeweils so angepasst wird, dass durchschnittlich alle 10 Minuten eine Werteinheit gefunden werden kann. Der Teilnehmer, der eine gültige Lösung für das mathematische Problem gefunden hat (d. h. er hat eine Münze durch sog. Bitcoin-Mining erzeugt), kann alle anderen Teilnehmer des Verfahrens darüber informieren, indem er der Gemeinschaft einen *Block* mit der Lösung und anderen Informationen, wie kürzlich stattgefundenen Transaktionen, zur Verfügung stellt.

Derzeit enthalten Blöcke 50 Bitcoins. Dies ist auch die *Blockprämie*, die für die Lösung eines Blocks durch die Gemeinschaft an denjenigen, der diese findet, gewährt wird. Dies ist ein Anreizmechanismus, der Teilnehmer zur Erzeugung von Bitcoins veranlassen soll, aber auch als Gegenwert für den betriebenen technischen Aufwand gedacht ist. Die Anzahl der gültigen Bitcoins, die in einem Zeitraum berechenbar sind, wird sich nach der derzeit gültigen Vorhersage etwa alle vier Jahre halbieren. Dabei können andere Teilnehmer des Verfahrens leicht prüfen, ob Bitcoins mit gültigen Parametern generiert worden sind, sodass gefälschte oder ungültige Bitcoins identifiziert und verworfen werden können. Die Prüf- und Sicherheitsmechanismen zur Beurteilung von Bitcoins führt dazu, dass nur etwa 21 Millionen Bitcoins die Voraussetzungen des zugrundeliegenden mathematischen Problems und der Prüfmechanismen erfüllen können.

Da die Blockprämie im Laufe der Zeit verringert wird, wird erwartet, dass Bitcoin-Miner in der Zukunft dazu übergehen werden, die vorhandene technische Ausrüstung zur Abwicklung kostenpflichtiger Transaktionen zu nutzen. Dabei wird von einem Zahlungsempfänger eine geringe Transaktionsgebühr ge-

zahlt, deren Höhe die Geschwindigkeit der Transaktionsabwicklung bestimmt.

Zahlungsvorgang

Um zu garantieren, dass keine dritte Partei in die Lage versetzt wird, die Bitcoins anderer Teilnehmer auszugeben, indem Transaktionen in deren Namen durchgeführt werden, verwendet das Bitcoin-System Public-Key-Kryptografie zur Erstellung und Verifikation digitaler Signaturen. Jeder Bitcoin-Teilnehmer kann je ein Schlüsselpaar mit einer oder mehreren Adressen verknüpfen und in seiner *digitalen Brieftasche*, einem kryptografisch gesicherten Datensatz, zusammen mit signierten Bitcoins verwalten. Nur der Besitzer der zugehörigen geheimen Schlüsselteile kann Bitcoins aus seiner Brieftasche ausgeben, indem er die entsprechende Transaktion signiert, jeder andere Teilnehmer der Bitcoin-Gemeinschaft kann jedoch die Signatur der Transaktion anhand des öffentlichen Schlüsselteils der Transaktionsteilnehmer prüfen.

Der Ablauf einer Transaktion zwischen Alice und Bob gestaltet sich wie folgt:

- Bob sendet seine Bitcoin-Adresse, aus der der öffentliche Schlüssel abgeleitet werden kann, an Alice.
- Alice erstellt eine Transaktionsnachricht, die den öffentlichen Schlüssel Bobs und die zu übertragende Summe enthält.
- Alice signiert die Transaktion mit ihrem geheimen Schlüsselteil.
- Alice stellt die Transaktionsnachricht der Bitcoin-Gemeinschaft öffentlich zur Einsicht zur Verfügung

Die Transaktion ist nun öffentlich und sofern die beteiligten Adressen Alice und Bob zugeordnet werden können, lautet der Informationsgehalt: Alice und Bob einigten sich auf die Übertragung einer bestimmten Summe.

Bob kann die erhaltenen Bitcoins nun an Charlie überweisen. Dazu verfährt er analog zu Alice zuvor:

- Charlie sendet seine Bitcoin-Adresse an Bob
- Bob erzeugt eine Transaktionsnachricht mit dem öffentlichen Schlüssel Charlies und der Transaktionssumme.

- Bob signiert die Transaktionsnachricht mit seinem geheimen Schlüssel.
- Bob veröffentlicht die Transaktionsnachricht in der Bitcoin-Gemeinschaft.

Eine dritte Partei kann die Eigentümerschaft eines Bitcoins nicht verändern, da andernfalls die Signatur der zuletzt ausgeführten Transaktion oder die Signatur einer digitalen Geldbörse ungültig werden würde. Dies kann nur durch eine Transaktion mit der Signatur des privaten Schlüssels des jeweils letzten Eigentümers erfolgen. Ein Empfänger eines Bitcoins signiert somit auch den Eigentümerverlauf, der einem Bitcoin als Abfolge digitaler Signaturen anhaftet.

Mehrfachverwendung eines Bitcoins

Das oben skizzierte Protokoll hindert Alice nicht daran, den verwendeten Bitcoin in einer weiteren Transaktion zu verwenden. Der folgende Ablauf soll dies verhindern und ist die primäre Innovation des Bitcoin-Systems.

- Die Transaktionsnachrichten werden in der Bitcoin-Gemeinschaft veröffentlicht und erreichen so alle (oder zumindest möglichst viele) Teilnehmer.
- Eine ständig wachsende Verkettung von Blöcken, die alle Transaktionsnachrichten enthält, wird bei allen Bitcoin-Teilnehmern verwaltet (jeder besitzt eine vollständige Kopie).
- Um in der Transaktionskette akzeptiert zu werden, müssen neue Transaktionsblöcke kryptografisch gültig sein und einen Nachweis enthalten, dass es sich ursprünglich um einen der von einem Miner rechtmäßig generierten Blöcke handelt.
- Die Blöcke werden durch ein kumulatives Hash-Verfahren so verkettet, dass bei der Änderung eines Blocks alle folgenden Blöcke neu berechnet werden müssen.
- Wenn für eine Blockkette mehrere gültige Erweiterungen existieren, so werden alle bis auf die längste Kette für ungültig erklärt. Nur die längste Kette wird mit weiteren Transaktionen erweitert. Die längste Kette ist

dabei die Kette, die mit dem höchsten gemeinschaftlichen, rechnerischen Aufwand erstellt wurde.

Wenn Bob eine in einem Block befindliche Transaktion von Alice prüft und diese sich in der längsten der vorhandenen Verkettung befindet (wobei der Vorgang der Verkettung mit rechnerischem Aufwand verbunden ist), so kann er davon ausgehen, dass diese Transaktion von der Gemeinschaft als gültig akzeptiert wurde und Alice mit dem betreffenden Bitcoin keine weitere gültige Transaktion durchführen kann. Um dennoch eine zweite Transaktion mit dem selben Bitcoin zu fälschen, müsste Alice eine Verkettung erstellen, deren rechnerischer Aufwand höher ist, als die kumulative Rechenleistung aller anderen Bitcoin-Teilnehmer.

Anonymität

Das Bitcoin-Netzwerk erfordert keine unmittelbar personenbeziehbaren Konten, E-Mail Adressen oder sonstige Benutzernamen oder Passwörter, um Bitcoins zu verwalten oder Transaktionen zu initiieren. Jedes *Konto* ist durch eine generische Adresse und ein kryptografisches Schlüsselpaar gekennzeichnet. Das Geld auf dem jeweiligen Konto (genauer: in der betreffenden digitalen Geldbörse) gehört stets demjenigen, der mit dem zugehörigen privaten Schlüssel Signaturen ausstellen kann. Die Schlüssel brauchen nicht zentral registriert zu werden, da sie ausschließlich zur Signatur von Transaktionen, bzw. zum Prüfen von Signaturen verwendet werden. Beteiligte an einer Transaktion brauchen den jeweiligen Gegenüber nicht identifizieren zu können, so wie dies bei einer Barzahlung mit realem Bargeld auch der Fall ist.

Jeder Bitcoin-Teilnehmer kann beliebig viele Bitcoin-Adressen der Form `1Bg3Qivmp9L3scH6Rq1BDqomSCYk7X84K1` führen, die jeweils mit einer digitalen Geldbörse verknüpft sind, sodass die Zuordnung einer bestimmten Adresse zu einer Person weiter verschleiert wird. Um seine Privatsphäre zu schützen, kann der Empfänger einer Transaktion beispielsweise für jeden Empfang eine neue Adresse generieren. Ein späterer Empfänger des Bitcoins in der Transaktionskette kann auf diese Weise auch keine Rückschlüsse auf vorherige Besitzer ziehen, ohne weitere Informationen heranzuziehen. Eine Analyse der Anonymität im Bitcoin-System findet sich in Reid [52].

7.1.2 ukash

Neben elektronischen Barzahlungssystemen ohne zentrale Garantiestelle existieren unterschiedliche kommerzielle Unternehmen, die elektronisches Bargeld mit anonymem Charakter herausgeben.

Die Fa. *ukash* ist eine Tochtergesellschaft der britischen Smart Voucher Ltd. und vertreibt das ebenfalls als ukash bezeichnete Bezahlsystem für E-Commerce. Dabei besteht das Zahlungsmittel aus einem elektronischen *pre-paid Voucher*, also einem vorab zu zahlenden elektronischen Coupon, der durch eine 19-stellige Zeichenkette identifiziert wird. Es besteht eine Wertobergrenze von 500 £ bzw. 750 € je Coupon und mittlerweile sind etliche Varianten auf dem Markt, etwa *ukash Neo*, einem MasterCard-Äquivalent, mit dem bei allen MasterCard-Akzeptanzstellen gezahlt werden kann oder *ukash Air* zur Zahlung unter Zuhilfenahme von Smartphones.

Hauptkritikpunkt bei ukash ist die Deckung von kriminellen Zahlungsvorgängen, wie sie z. B. bei sog. *Scare-Ware* eingesetzt wird. Der Computer eines Benutzers wird bei einer entsprechenden Infektion mit einer Zahlungsaufforderung über ukash blockiert. Dies geschieht unter Vortäuschung falscher Tatsachen, indem scheinbar in den Namen von Wirtschaftsunternehmen wie der GEMA behauptet wird, es sei rechtswidriges Medienmaterial auf dem Computer entdeckt worden und der Benutzer könne nur durch die Zahlung einer Anzeige entgehen. Obwohl ukash im Speziellen Ziel dieses Vorwurfs ist, sind prinzipiell auch andere anonyme Zahlungssysteme Gegenstand der Kritik. Vor allem gemessen an der Rechtslage hat Bargeld denselben anonymisierenden Effekt, es kann jedoch nicht so einfach übertragen werden.

Durch die einfache Übertragbarkeit der Identifikationsnummer eines ukash-Vouchers oder elektronischen Bargeldes im Allgemeinen kann der Zahlungsfluss jedoch kaum beeinflusst werden und kann auf vielen unterschiedlichen Kommunikationskanälen erfolgen.

7.2 Ein Bankenszenario

Die Protagonisten des Szenarios sind Alice und Bob, die als die beiden Bankkunden auftreten sowie die Verrechnungsstelle Charlie als Bank. Charlie interessieren dabei nicht die Salden der von ihm verwaltetet Konten, d. h. er führt

keine Überprüfung der Höhe der Kontostände aus. Er muss lediglich garantieren, dass die von ihm durchgeführten Transaktionen gedeckt sind und dass alle der von ihm verwalteten Konten kreditorisch geführt werden. D. h. dass die Konten dauerhaft Guthaben aufweisen müssen und nicht überzogen werden können. Dazu muss Charlie bei jeder Transaktion sicherstellen, dass der für ein Konto aus der Transaktion resultierende Saldo positiv ist.

Das Szenario beinhaltet also zwei wesentliche Problemstellungen, die in der Folge mit einer Kombination aus jeweils asymmetrischer homomorpher und klassischer (RSA-) Kryptografie gelöst werden.

Definition 7.2.1 (Gläubiger)**.** Alice und Bob sind Gläubiger von Charlie, der für sie die Konten verwaltet. Alle Parteien besitzen zwei Schlüsselpaare, nämlich ein homomorphes Schlüsselpaar mit geheimem Schlüssel $\mathcal{H}$ und öffentlichem Schlüssel $\mathcal{H}'$, sowie ein RSA-Schlüsselpaar $\mathcal{R}$ und $\mathcal{R}'$. Dabei bezeichnet z. B. $\mathcal{R}'_{Alice}$ den öffentlichen RSA-Schlüssel von Alice.

Definition 7.2.2 (Clearingstelle)**.** Charlie ist die zentrale Clearingstelle und besitzt ebenfalls ein homomorphes Schlüsselpaar $\{\mathcal{H}_{Charlie}, \mathcal{H}'_{Charlie}\}$.

Definition 7.2.3 (Notation)**.** Ein homomorph verschlüsselter Wert 100 wird z. B. als $[100]_{\mathcal{H}'_{Bob}}$ mit dem verwendeten Schlüssel als Index notiert. Ein signierter Wert wird beispielsweise als $\{100\}_{\mathcal{R}_{Bob}}$ mit dem verwendeten Schlüssel als Index notiert. Ein Wert 100 verschlüsselt unter Alices homomorphen öffentlichen Schlüssel und von Bobs privaten RSA-Schlüssel signiert, wird als $\{[100]_{\mathcal{H}'_{Alice}}\}_{\mathcal{R}_{Bob}}$ angegeben.

Definition 7.2.4 (Konto)**.** Die von Charlie verwalteten Konten K bestehen aus den öffentlichen Schlüsseln des Gläubigers und dem Saldo $\overline{S}$ des Kontos. $\overline{S}$ ist ein Vektor mit dem binär kodierten Saldo und mit dem öffentlichen Schlüssel $\mathcal{H}'$ des Gläubigers verschlüsselt. $K_x := \{\mathcal{R}'_x, \mathcal{H}'_x, [\overline{S}]_{\mathcal{H}'_x} := \{[s_0]_{\mathcal{H}'_x}, [s_1]_{\mathcal{H}'_x}, ..., [s_n]_{\mathcal{H}'_x}\}\}$

Dieser Abschnitt gliedert sich in die Behandlung der folgenden Komponenten des Bankenszenarios:

1. Kontoeröffnung bzw. initiale Saldengenerierung
2. Transaktionsprotokolle
3. Kontoauflösung bzw. Saldenliquidierung

7.3 Initiale Saldengenerierung

Der Übergang von einer existierenden Währung in einen Gegenwert auf einem verschlüsselten Konto erfolgt über eine vertrauenswürdige Inkassostelle namens Trent[3] nach Wahl des Gläubigers. Zu diesem Zweck wird ein herausgehobener Gläubiger der Bank eingerichtet, der die Tauschtransaktion treuhänderisch vermittelt und so in einer Transaktionskette Initial-Guthaben von Charlie zum Gläubiger transferiert.

Definition 7.3.1 (Inkassostelle)**.** Trent ist eine vertrauenswürdige dritte Partei und übernimmt die Umwandlung von herkömmlichen Gegenwerten wie Geld in verschlüsselte Buchungsbeträge. Trent dient als treuhänderische Vermittlung zwischen einem Gläubiger und Charlie. Trent besitzt für die Transaktionen mit Charlie ein eigenes Schlüsselpaar $\{\mathcal{H}_{Trent}, \mathcal{H}'_{Trent}\}$.

Der Ablauf gliedert sich wie folgt:

1. Charlie generiert einen Initialsaldo auf einem eigenen Konto unter $\mathcal{H}'_{Charlie}$.
2. Alice und Trent eröffnen Konten bei Charlie. Dieser initialisiert die Salden mit $[0]_{H'_{Alice}}$ und $[0]_{H'_{Trent}}$.
3. Alice möchte einen Betrag a auf ihr Konto bei Charlie buchen und zahlt den Gegenwert bei Trent ein.
4. Trent überweist mit Hilfe des Transaktionsprotokolls (s. u.) den Betrag auf Alices Konto bei Charlie.

[3]Trent ist im klassischen Alice & Bob-Szenario eine neutrale und vertrauenswürdige Partei (trusted party)

Auf diese Weise kann Charlie zwar eine Überweisung auf Alices Konto feststellen, bleibt jedoch im Unklaren über die Höhe der Überweisung. Wegen der verwendeten Subprotokolle bleibt die gesamte Geldmenge nachweisbar stabil, und ebenso bleiben alle beteiligten Konten im Guthabenbereich.

7.4 Transaktionsprotokolle

Die Interaktionen zwischen den Entitäten des Bankenszenarios werden über Transaktionsprotokolle abgewickelt, in denen die Salden übertragen und Prüfungen über die Integrität der Überweisungen durchgeführt werden.

7.4.1 Überweisung durchführen

Die Transaktion wird im Rahmen eines Protokolls zwischen den drei beteiligten Parteien abgewickelt. Die Transaktion kann als Überweisung, bestehend aus einer Abbuchung beim Schuldner und einer Gutschrift beim Gläubiger der Transaktion, definiert werden. Im Beispiel ist Alice die Schuldnerin, Bob der Gläubiger der Transaktion. Diese wird über die Clearingstelle Charlie abgerechnet. Der die Finanztransaktion auslösende Geschäftsvorfall zwischen Alice und Bob sowie die Einigung über den zu übertragenden Betrag werden im Vorfelde unabhängig von Charlie ausgehandelt.

Protokoll `Transaktion`

Alice, Bob und Charlie führen die folgenden Schritte aus, um die Transaktion abzuschließen:

1. Alice und Bob handeln eine Überweisung über einen bestimmten Betrag an Geldeinheiten aus.

2. Alice generiert den Buchungsbetrag für Bobs Konto als verschlüsselten Vektor $\overline{B}$ über den Elementen des binär codierten Betrages, signiert diesen als $\{[\overline{B}]_{\mathcal{H}'_{Bob}}\}_{\mathcal{R}_{Alice}}$ und überträgt ihn zu Bob.

3. Bob prüft die Summe mit $\mathcal{H}_{Bob}$ und signiert das gesamte Aggregat als $\{\{[\overline{B}]_{\mathcal{H}'_{Bob}}\}_{\mathcal{R}_{Alice}}\}_{\mathcal{R}_{Bob}}$. Er generiert analog den Buchungsbetrag für Alice als $\{[\overline{A}]_{\mathcal{H}'_{Alice}}\}_{\mathcal{R}_{Bob}}$ und überträgt beides an Alice.

4. Alice prüft den von Bob generierten Buchungsbetrag mit $\mathcal{H}_{Alice}$, signiert das Aggregat als $\{\{[\overline{A}]_{\mathcal{H}'_{Alice}}\}_{\mathcal{R}_{Bob}}\}_{\mathcal{R}_{Alice}}$ und überträgt beide doppelt signierten Buchungsbeträge an Charlie.

5. Charlie prüft die Signaturen der Buchungsbeträge und führt mit Alice und Bob das `Betragsgleichheit`-Protokoll aus. Bei Erfolg führt er mit Alice das `Vorzeichen`-Protokoll aus, sodann eine Teiltransaktion, in der er den Betrag von Alices Konto homomorph subtrahiert und mit dem Ergebnisvorzeichen nochmals das `Vorzeichen`-Protokoll mit Alice ausführt.

6. Wenn Alices Saldo noch im Guthabenbereich ist, vollendet Charlie die Transaktion durch Gutschrift des Betrages auf Bobs Konto.

7.4.2 Nachweis eines Vorzeichens

Eine fundamentale Problemstellung des hier behandelten Szenarios ist der Nachweis des Guthabens auf einem Konto, genauer gesagt: der Nachweis, dass das Vorzeichen des Guthabens positiv, die binäre Codierung des Vorzeichens also 0 ist. Dieser Nachweis muss Charlie vorliegen, damit er die Deckung einer Transaktion garantieren kann. Die Gläubiger (im Beispiel Alice) wollen Charlie aber nicht den gesamten Saldo des Kontos offenlegen. Das Protokoll muss von Charlie auch verwendet werden, um die Überweisungsbeträge der Transaktionsteilnehmer zu prüfen. Sonst wäre es denkbar, dass beide Parteien kollaborativ einen negativen Betrag vereinbaren und somit beide Konten mit einer Gutschrift begünstigt werden würden. Charlie könnte dies nicht kontrollieren, da er beim Schuldner stets eine Belastung (Subtraktion) und beim Zahlungsempfänger stets eine Gutschrift (Addition) mit den verschlüsselten Beträgen vornimmt.

Die folgenden Unterabschnitte entwickeln eine Näherung an eine Lösung, die die Privatsphäre auf der einen und die Integrität der Kontoführung auf der anderen Seite berücksichtigt. Dabei soll der iterative Prozess der Lösungsfindung einige grundsätzliche Überlegungen beim Design eines hybrid-homomorphen Protokolls vermitteln.

Protokoll `Vorzeichen1`

Charlie und Alice führen das folgende Protokoll mit dem Vorzeichenbit $[s]_{\mathcal{H}'_{Alice}}$ und einem Sicherheitsparameter λ aus:

1. Charlie generiert ein Array der Länge λ und füllt dieses zufällig mit unterschiedlichen Werten $[0]_{\mathcal{H}'_{Alice}}$ und $[1]_{\mathcal{H}'_{Alice}}$. An einer Stelle i setzt er das Vorzeichen $[s]_{\mathcal{H}'_{Alice}} + r \ mod\ 2$ mit $r \in \{[0]_{\mathcal{H}'_{Alice}}, [1]_{\mathcal{H}'_{Alice}}\}$ von Alices Saldo ein und übergibt ihr das Array.

2. Alice entschlüsselt das Array und übergibt die unverschlüsselten Werte an Charlie zurück.

3. Charlie prüft, ob das Element an Stelle i den Wert $0 + r \ mod\ 2$ hat und schließt bei Übereinstimmung, dass Alices Konto ein Guthaben aufweist.

Diskussion: Da Charlie das Vorzeichenbit mit einem Zufallsbit randomisiert, kann Alice nur mit einer Wahrscheinlichkeit $\frac{1}{\lambda}$ ein gefälschtes Ergebnis generieren, d. h. ein Array erzeugen, das an der Stelle i, die ihr nicht bekannt ist, den Wert $0 + r \in \{0, 1\} \ mod\ 2$ besitzt, wobei r ihr ebenfalls nicht bekannt ist.

Das Problem bei diesem Vorgehen ist allerdings die fehlende Privatsphäre für Alice, denn Charlie könnte leicht die Saldobits in das Array mischen und anschließend den von Alice selbst entschlüsselten Saldo lesen.

Protokoll `Vorzeichen2`

Charlie und Alice führen das folgende Protokoll mit dem Vorzeichenbit $[s]_{\mathcal{H}'_{Alice}}$ und einem Sicherheitsparameter λ aus. Die zweite Protokollvariante führt einen Schritt zur Stärkung der Privatsphäre Alices ein:

1. Charlie generiert ein Array der Länge λ und füllt dieses zufällig mit unterschiedlichen Werten $[0]_{\mathcal{H}'_{Alice}}$ und $[1]_{\mathcal{H}'_{Alice}}$. An einer Stelle i setzt er das Vorzeichen $[s]_{\mathcal{H}'_{Alice}} + r \ mod\ 2$ mit $r \in \{[0]_{\mathcal{H}'_{Alice}}, [1]_{\mathcal{H}'_{Alice}}\}$ von Alices Saldo ein und übergibt ihr das Array.

2. Alice entschlüsselt das Array mit $\mathcal{H}_{Alice}$ und verschlüsselt jeden Wert mit einem eigenen symmetrischen Einmalschlüssel. Sie übergibt die symmetrisch verschlüsselten Werte an Charlie zurück.

3. Charlie darf nun eine Stelle wählen, zu der Alice ihm den symmetrischen Schlüssel verrät und kann so die Stelle wählen, an der er das Vorzeichenbit eingefügt hat.

Diskussion: Dieses Protokoll bewahrt Alice davor, dass Charlie mehr als ein Bit des Kontostandes erfährt. Sollte Charlie nicht das Vorzeichenbit, sondern ein anderes entschlüsseln lassen, so kann er möglicherweise die Deckung der Transaktion nicht mehr garantieren und würde so seine Glaubwürdigkeit verlieren. Andererseits ist dieses Verfahren für Charlie ausgesprochen ungünstig, da er durch die Verdeckung der anderen Bits nicht mehr prüfen kann, ob Alice tatsächlich alle Bits richtig entschlüsselt hat. So könnte Alice einfach ein Array der Größe λ von $[0]_{H'_{Alice}}$ erzeugen, symmetrisch verschlüsselt an Charlie senden und dadurch eine Erfolgsquote von durchschnittlich $\frac{1}{2}$ für einen Betrug erreichen. Charlie muss also in die Lage versetzt werden, mehrere Bits abzufragen, darunter das Vorzeichenbit. Die Wahrscheinlichkeit, dass Alice erfolgreich betrügen kann, fällt bei n zusätzlichen entschlüsselten Bits auf $\frac{1}{2^n}$ bei zunehmender Gefährdung Alices Privatsphäre.

Protokoll `Vorzeichen3`

Charlie und Alice führen das folgende Protokoll mit dem Vorzeichenbit $[s]_{\mathcal{H}'_{Alice}}$ und einem Sicherheitsparameter λ aus:

1. Charlie generiert ein Array der Länge λ und füllt dies an zufälligen Stellen genau zur Hälfte mit unterschiedlichen $[1]_{\mathcal{H}'_{Alice}}$. An einer unbesetzten Stelle i des Arrays setzt er das Vorzeichen $[s]_{\mathcal{H}'_{Alice}}$ ein und füllt den Rest mit unterschiedlichen $[0]_{\mathcal{H}'_{Alice}}$ auf. Das Array wird an Alice übertragen.

2. Alice entschlüsselt die Elemente des Arrays und überträgt es zurück an Charlie.

3. Charlie prüft die Stelle i des entschlüsselten Arrays und schließt bei Vorliegen einer 0, dass Alices Konto ein Guthaben aufweist.

Diskussion: Charlie und Alice vereinbaren ein Hamming-Gewicht des Arrays von $\frac{\lambda}{2}$ aus zwei Gründen. Wenn Alice betrügen will, weil ihre Überweisung nicht gedeckt ist, so ist das Hamming-Gewicht des von ihr entschlüsselten Arrays $\frac{\lambda}{2}+1$. Sie müsste dann diejenige 1 im Array finden, die das Vorzeichenbit

enthält und dort eine 0 einsetzen. Die Wahrscheinlichkeit, dass ihr das gelingt ist $\frac{2}{\lambda}$. In jedem Fall kann Charlie eine Transaktion bei einem Hamming-Gewicht ungleich $\frac{\lambda}{2}$ nicht abschließen. Will Charlie statt des Vorzeichenbits ein Saldobit abfragen, so kann er durchschnittlich nur in jedem zweiten Fall ein Array mit dem vereinbarten Hamming-Gewicht $\frac{\lambda}{2}$ generieren, weil er den Wert des abzufragenden Bits nicht kennt. In jedem Fall ist die Vertrauenswürdigkeit Charlies mit einem einzigen aufgedeckten Betrugsversuch zerstört.

Protokoll `Vorzeichen_final`

Die abschließende Protokollvariante ist gleichzeitig die effektivste und bietet sowohl für Alice als auch für Charlie eine vergleichsweise hohe Sicherheit. Charlie und Alice führen das folgende Protokoll mit dem Vorzeichenbit $[s]_{\mathcal{H}'_{Alice}}$ und einem Sicherheitsparameter λ aus:

1. Charlie generiert ein Array der Länge λ und füllt dies mit unterschiedlichen $[1]_{\mathcal{H}'_{Alice}}$. An einer bestimmten Stelle i fügt er das Vorzeichenbit $[s]_{\mathcal{H}'_{Alice}}$ von Alices Kontosaldo ein und fordert Alice dazu auf, die Stelle mit dem Vorzeichen zu benennen.

2. Alice entschlüsselt das Array und sendet den Index i' der von ihr ermittelten Stelle an Charlie.

3. Charlie prüft, ob $i = i'$.

Diskussion: Sofern der Kontostand von Alices Konto im Guthaben geführt wird, kann sie leicht die Stelle mit dem Vorzeichenbit 0 ermitteln. Andernfalls ist das Konto offenbar überzogen und die Wahrscheinlichkeit, dass Alice die richtige Stelle tippt ist $\frac{1}{\lambda}$.

7.4.3 Nachweis der Betragsgleichheit

Ein weiteres wichtiges Protokoll im Rahmen der Finanztransaktion ist die Sicherstellung der Gleichheit der gebuchten Beträge durch Charlie. Eine Gesetzmäßigkeit der doppelten Buchführung ist die Verbuchung gleicher Beträge (Soll und Haben) durch einen Buchungssatz. Die Gleichheit der beiden Beträge ist deshalb erforderlich, weil eine Ungleichheit zu einer Veränderung der

Bilanzsumme im Gesamtsystem führen würde. Im konkreten Fall wäre die Konsequenz, dass durch eine im Sinne der doppelten Buchführung fehlerhafte Buchung die Geldmenge verändert wird. Charlie muss also als Bestandteil seiner Deckungsgarantie für die Bilanz diese Prüfung durchführen können, ohne die konkreten Beträge zu kennen. Abbildung 7.1 zeigt schematisch den Ablauf der Betragsprüfung.

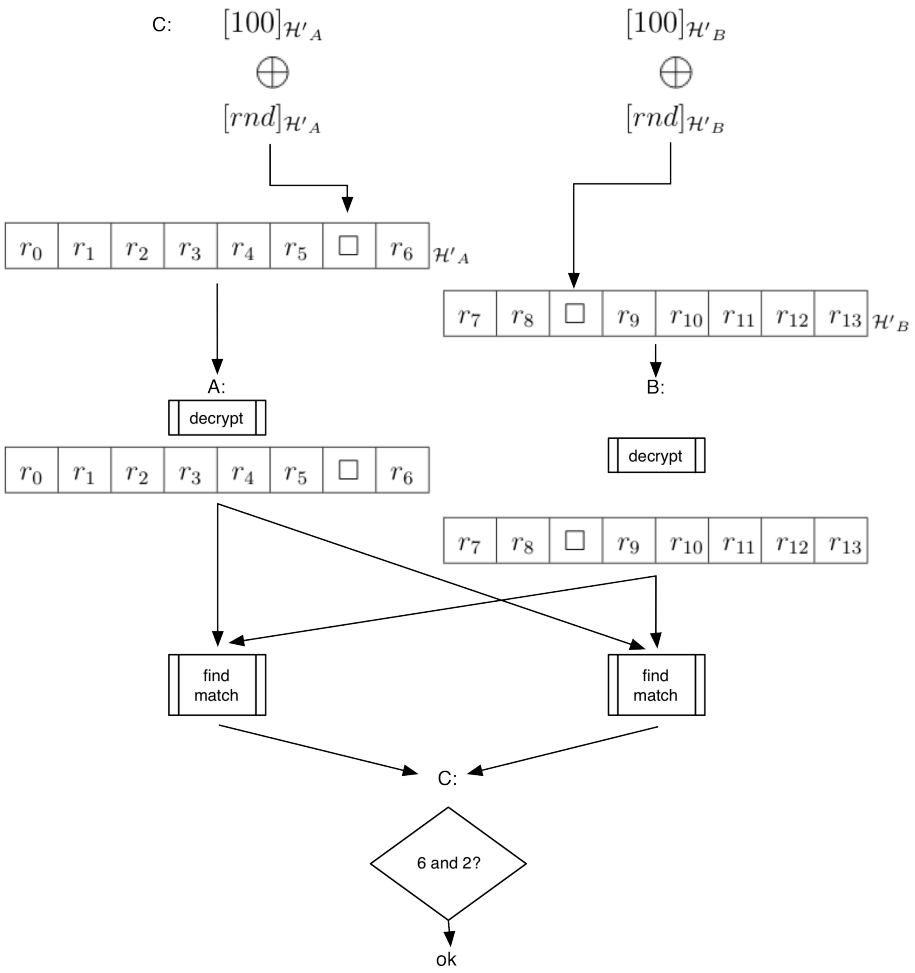


Abbildung 7.1: Ablauf der Betragsprüfung

Protokoll `Betragsgleichheit`

Charlie führt das folgende Protokoll mit Alice und Bob bei einem Sicherheitsparameter λ aus:

1. Charlie generiert bei Bitlänge l eines Kontosaldos S Zufallsbits $r_{0..l-1} \in \{0,1\}$ als R und besetzt in einem Array A der Länge λ eine zufällige Stelle i mit $PRP_\eta([S]_{\mathcal{H}'_{Alice}} \oplus [R]_{\mathcal{H}'_{Alice}})$ und einem zweiten Array B der Länge λ ebenfalls eine zufällige Stelle j mit $PRP_\eta([S]_{\mathcal{H}'_{Bob}} \oplus [R]_{\mathcal{H}'_{Bob}})$. Die verbleibenden Stellen jedes Arrays füllt Charlie mit $[\{0,1\}^l]_{\mathcal{H}'_x}$, sodass alle Elemente in A und B paarweise unterschiedlich sind.

2. Charlie überträgt $[A]_{\mathcal{H}'_{Alice}}$ an Alice und $[B]_{\mathcal{H}'_{Bob}}$ an Bob.

3. Alice und Bob entschlüsseln die Elemente und Alice sendet A an Bob, Bob sendet B an Alice.

4. Alice sucht nach einer Übereinstimmung in A und B und sendet die Indexpositionen a_{Alice} und b_{Alice} an Charlie, Bob verfährt analog.

5. Charlie prüft, ob $i = a_{Alice} = a_{Bob}$ und $j = b_{Alice} = b_{Bob}$ und schließt bei Gleichheit auf übereinstimmende Buchungsbeträge.

Diskussion: Charlie ist im Besitz beider Kontosalden jeweils mit $\mathcal{H}'_x$ des Kontoinhabers verschlüsselt. Er kann durch homomorphe *mod* 2-Addition einer zufälligen Bitsequenz (einem One-Time-Pad), die unter demselben Schlüssel $\mathcal{H}'_x$ verschlüsselt ist, wiederum den Saldo im homomorphen Raum XOR-verschlüsseln. Die Stelle A_i des Arrays für Alice wird beispielsweise wie folgt mit dem verschlüsselten Saldo besetzt:

$$[A]_{\mathcal{H}'_{Alice_{i_j}}} \leftarrow [S]_{\mathcal{H}'_{Alice_j}} \oplus [R]_{\mathcal{H}'_{Alice_j}}, 0 \leq j < l$$

Die anderen Stellen des Arrays A werden mit zufälligen Bitsequenzen besetzt, wobei leicht nachzuweisen ist, dass der mit One-Time-Pad verschlüsselte Saldo durch Alice nicht von den Zufallssequenzen unterschieden werden kann[4]. Um einen kollaborativen Angriff von Alice und Bob zu verhindern, wird der verschlüsselte Saldo zufällig durch die Funktion PRP_η permutiert.

[4]Shannon bewies bereits in den 1940er Jahren, dass das One-Time-Pad informationstheoretisch perfekte Sicherheit bietet. Eine Ableitung des Beweises wird in dieser Arbeit zum Beweis von Lemma 3.3.6 verwendet

Angriff ohne Permutation: Die im Anschluss an die One-Time-Pad-Verschlüsselung stattfindende Permutation verhindert den folgenden Angriff. Alice generiert eine Buchung über den Betrag a, Bob über den Betrag b mit $a \neq b$. Charlie generiert die Arrays

$$[A]_{\mathcal{H}'_{Alice}} \leftarrow [\{R_1^a, R_2^a, ..., a \oplus R, ..., R_\lambda^a\}]_{\mathcal{H}'_{Alice}}$$

und

$$[B]_{\mathcal{H}'_{Bob}} \leftarrow [\{R_1^b, R_2^b, ..., b \oplus R, ..., R_\lambda^b\}]_{\mathcal{H}'_{Bob}}$$

Anstatt einer Suche nach gleichen Bitmustern (die der Annahme $a \neq b$ zufolge nicht erfolgreich sein kann), führen Alice und Bob über allen Paaren die folgende Suche aus:

$$\forall x \in A, y \in B : x \oplus y = a \oplus b$$

Diese Beziehung gilt, weil

$$(a \oplus R) \oplus (b \oplus R) \;\Leftrightarrow\; a \oplus b$$

Auf diese Weise könnten sie trotz unterschiedlicher Beträge die Stellen der vermeintlichen Übereinstimmung konsistent angeben. Durch die Permutation wird verhindert, dass der Term $a \oplus b$ verwendet werden kann.

7.5 Ergebnisse

Dieses Kapitel entwickelt Lösungen zur Abbildung eines Szenarios, in dem Finanztransaktionen mit verschlüsselten Salden abgebildet werden können. Dazu werden hybrid-homomorphe Protokolle vorgestellt, die einzelne Problemstellungen, wie die Ermittlung eines verschlüsselten Vorzeichens oder die Gleichheit zweier verschlüsselter Beträge lösen und zusammen zur Lösung der definierten, komplexen Problemstellung beitragen. Bei der Formulierung der Protokolle werden grundsätzliche Erwägungen für die Konstruktion von hybriden Algorithmen aufgezeigt, die das Spannungsfeld zwischen erforderlicher Vertraulichkeit und erzielbarer Leistung vorgibt.

Kapitel 8

Zusammenfassung und Ausblick

Die vorliegende Arbeit untersucht unterschiedliche Ansätze zum Schutz aktiver Programme und Funktionen zur Laufzeit. Die Basis für die erarbeiteten Konzepte ist die homomorphe Kryptografie, welche die Ausführung elementarer Binäroperationen im verschlüsselten Raum gestattet. Für die Abbildung einfacher Funktionen wird ein begrenzt homomorphes Schema über der Zahlenmenge $\mathbb{N}$ vorgestellt und formal analysiert. Darüberhinaus wird die im Rahmen dieser Arbeit entstandene, erste öffentlich verfügbare Umsetzung des unbegrenzt homomorphen Smart-Vercauteren Kryptosystems vorgestellt und auf ihre Eigenschaften hin untersucht.

Im Hauptteil dieser Arbeit wird aufbauend auf den erarbeiteten verschlüsselten Elementarfunktionen der wichtige Anwendungsfall der verschlüsselten Suche formalisiert und unter Berücksichtigung typischer Randbedingungen in Form von exakter und unscharfer (fuzzy) sowie der Suche in Datenströmen exemplarisch umgesetzt. Erstmals wird in dieser Arbeit ein vollständig verschlüsseltes Maschinenmodell konzipiert. Dieses besteht zum einen aus verschlüsseltem Speicher, auf den mit verschlüsselten Adressen zugegriffen werden kann. Zum anderen wird ein Prozessormodell umgesetzt, das die Ausführung von Programmen im verschlüsselten Speicher gestattet, sodass das Programm und die Daten, auf denen es operiert, zu jedem Zeitpunkt verschlüsselt sind und durch den Ausführenden nicht entschlüsselt werden können. Dieses Modell lässt sich als Grundkonzept für die vertrauliche Programmausführung in verteilten Systemen, wie dem Cloud-Computing, verwenden.

Ausgehend von der ermittelten Leistungsfähigkeit der verschlüsselten Ma-

schinenabstraktion stellt diese Arbeit eine Strategie zur Beschleunigung von homomorph verschlüsselten Funktionen vor. Bei diesen hybriden Systemen wird nur ein kleiner Teil der Operationen verschlüsselt, wohingegen der weitaus größere Teil unverschlüsselt abläuft und so zur Laufzeitverbesserung beiträgt. Die Herausforderung bei hybrid-homomorph verschlüsselten Systemen ist die Konstruktion geeigneter Protokolle, die trotz der Verwendung unverschlüsselter Operationen die Vertraulichkeit der Gesamtfunktion sicherstellen können. Ein praktisches Beispiel zur systematischen Annäherung an ein solches Protokoll wird anhand verschlüsselter Finanztransaktionen dargestellt.

Ausblick

Die in dieser Arbeit vorgestellten Ergebnisse liefern eine Grundlage für weiterführende Forschungsvorhaben im Bereich der sicheren Delegation von Rechenleistung in unsicheren Umgebungen. Das vorgestellte verschlüsselte Maschinenmodell ist in seiner Leistungsfähigkeit begrenzt, sodass hier Potenzial zur Verbesserung in unterschiedlichen Richtungen vorhanden ist. Zum einen hat die Architektur der Prozessor- und Speicherschaltungsrepräsentationen unmittelbaren Einfluss auf das Laufzeitverhalten der verschlüsselten Maschine, sodass hier weitere Untersuchungen zur Bestimmung eines möglichen Optimums in Abhängigkeit zu bestimmenden Randbedingungen erforderlich sind. Zum anderen ist das Verhalten einer Hardware-Umsetzung des Konzeptes zu untersuchen. Hierfür wurden in Kapitel 5.6 bereits sehr konkrete Vorschläge für ein Folgeforschungsprojekt zusammengestellt.

Der Bereich der hybriden Systeme bietet weiteres Potenzial für Anschlussforschung. Durch die Verknüpfung von verschlüsselten und unverschlüsselten Algorithmen ergeben sich hier neben der Grundlagenforschung in der Kryptografie insbesondere Herausforderungen aus dem Bereich des Software-Engineerings. Vor allem Anwendungsarchitekturen und Protokolle für teilverschlüsselte Algorithmen sind hier von Bedeutung, sodass dieser Forschungszweig interdisziplinär geprägt und durch die Kombination von Vorgehensmustern aus unterschiedlichen Bereichen der Informatik gekennzeichnet ist.

Diese Arbeit stellt durch die vorgestellten Konzepte und Lösungen einen Beitrag zur vertraulichen Delegation von Rechenleistungen in potenziell unsi-

cheren Umgebungen dar und soll so die bisher unzureichend adressierte Sicherheit im verteilten Rechnen und im Cloud-Computing signifikant verbessern. Auf diese Arbeit aufbauende Forschungsprojekte können durch eine weitere Verbesserung der vertraulichen Delegation in verteilten Systemen ebenso zum Ausbau bewährter und zur Erschließung vieler neuer Anwendungsfälle in diesem Bereich beitragen.

Anhang A

Instruktionssatzarchitektur

A.1 Wortstruktur

Ein Speicherwort der Modellarchitektur steht im Little-Endian Format und besteht aus 5 Bits Instruktionsteil und 8 Bit Datenteil, wie in Abbildung A.1 dargestellt. Bei einem lesenden Speicherzugriff wird das Operandenregister (Akkumulator) mit dem Datenteil des Speicherwortes geladen (Bits 5-12). Bei einem schreibenden Speicherzugriff wird der Datenteil des adressierten Speicherwortes mit dem Registerinhalt beschrieben.

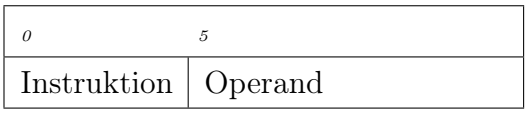


Abbildung A.1: Struktur des Maschinenwortes

A.2 Adressierungsarten

Der Befehlssatz des Beispielprozessors ist in Tabelle A.1 zusammengefasst und wird an dieser Stelle im Detail vorgestellt. Es werden die folgenden beiden Adressierungsarten nativ unterstützt:

- *unmittelbar:* der Operand ist dem Befehl unmittelbar im Maschinencode angefügt und kann aus dem Datenteil des Maschinenwortes entnommen werden

- *direkt:* der Operand steht direkt an der Adresse, die dem Maschinenbefehl im Datenteil angefügt ist und durch den FETCH_OPERAND Zyklus geholt wird

Befehle, die beide Modi unterstützen, verwenden Bit 4 als Indikator für den entsprechenden Modus. Durch die Maschinenwortstruktur ergibt sich, dass Speicheradressen und Operanden im Wertebereich $\{0, 255\}$ codiert werden können. Ebenfalls durch die Struktur des Maschinenwortes und des Speicherzugriffs ergibt sich, dass die beiden folgenden Adressierungsarten durch Befehlsfolgen emuliert werden können:

- *indirekt:* die endgültige Adresse steht an der Adresse, die durch den Datenteil des Befehlswortes angegeben wird
- *indiziert:* die endgültige Adresse ergibt sich durch die im Datenteil des Befehlswortes angegebenen Adresse plus einem Index

Diese *erweiterten Adressierungsarten* werden durch Selbstmodifikation des Codes erzielt. Listing A.1 zeigt ein Beispiel für indirekte Adressierung, Listing A.2 eines für indizierte Adressierung.

Listing A.1: indirekte Adressierung

```
1
2           La adr
3           ST load
4   load    La 0 ;wird ueberschrieben mit <target>
5
6   ...
7   adr     L <target>
```

Listing A.2: indizierte Adressierung

```
1
2           La load
3           CLC
4           ADDa index
5           ST load
6   load    La 0 ;wird ueberschrieben mit <offset>
7
8   ...
9   index   L <offset>
```

Das Prozessormodell verfügt über drei Flags:

Mnemonic	*Flags*	*Beschreibung*
L	Z,M	Register laden
ST	-	Register schreiben
ADD	Z,M,C	zum Register addieren
OR	Z,M	bitweises ODER
XOR	Z,M	bitweises XOR
AND	Z,M	bitweises UND
ROR	Z,M,C	rechts rotieren durch Übertrag
ROL	Z,M,C	links rotieren durch Übertrag
CLC	C	Übertrag löschen
SEC	C	Übertrag setzen
CMP	Z,M,C	mit Register vergleichen
BEQ,BZ	-	Verzweigen wenn Z=1
BMI	-	Verzweigen wenn M=1
JMP	-	unbedingter Sprung

Tabelle A.1: Instruction Set

- *Übertrag:* (Carry, C) wird gesetzt, sobald bei einer Addition ein Übertrag auftritt
- *Minus:* (M) wird gesetzt, wenn das höchstsignifikante Bit eines Ergebnisses 1 ist
- *Null:* (Zero, Z) wird gesetzt, wenn ein Ergebnis 0 ist

A.3 Befehlsreferenz

Dieser Abschnitt erläutert die Formate der Befehle. Dazu werden die Binärcodierungen im Befehlsteil des Maschinenwortes der einzelnen Instruktionen aufgeschlüsselt. Die Angaben *op* und *adr* beziehen sich auf einen 8 Bit breiten Operator bzw. eine 8 Bit breite Speicheradresse in Datenteil des Maschinenwortes. Eine Angabe von − im Operandenteil zeigt an, dass der betreffende Befehl keinen Operanden besitzt. Zur Ausführungszeit wird ein Operand nicht verwendet.

L - Register laden

L	1	1	0	0	0	op
La	1	1	0	0	1	adr

ST - Register schreiben

ST	1	1	1	1	0	adr

ADD - zum Register addieren

ADD	1	1	0	1	0	op
ADDa	1	1	0	1	1	adr

OR - bitweises ODER

OR	0	1	1	0	0	op
ORa	0	1	1	0	1	adr

XOR - bitweises Exklusiv-ODER

XOR	0	0	0	1	0	op
XORa	0	0	0	1	1	adr

AND - bitweises UND

AND	1	1	1	0	0	op
ANDa	1	1	1	0	1	adr

ROR - rechts rotieren durch Übertrag

ROR	0	0	1	1	0	-

ROL - links rotieren durch Übertrag

ROL	1	0	1	1	0	-

CLC - Übertrag löschen

CLC	0	1	0	1	0	-

SEC - Übertrag setzen

SEC	1	0	0	1	0	-

CMP - mit Register vergleichen

CMP	1	0	0	0	0	op
CMPa	1	0	0	0	1	adr

BEQ,BZ - verzweigen wenn Z=1

BEQ	1	0	1	0	0	adr

BMI - verzweigen wenn M=1

BMI	0	1	0	0	0	adr

J - unbedingter Sprung

J	0	0	1	0	0	adr

Anhang B

Beispielprogramme

B.1 Sortierung

Das folgende Beispielprogramm sortiert die Werte ab der Adresse *list* mit dem Bubble Sort Algorithmus. Die Anzahl der Werte wird in Adresse *len* angegeben, die sortierte Liste steht weiterhin an Adresse *list*. Das Programm demonstriert u. a. die indizierte Adressierung mit Hilfe von selbstmodifizierendem Code.

Listing B.1: Bubble Sort

```
1  INITAC   0
2  INITPC   start
3
4  list     L 6
5           L 5
6           L 4
7           L 9
8           L 5
9
10 len      L 5
11
12 idxa     L 0
13 idxb     L 1
14 temp     L 0
15 swap     L 0
16
17 start    La idxa
18          STa loada
19          STa loada2
20          STa storeb
21          La idxb
22          STa cmpb
23          STa loadb
24          STa storea
25          L 0
26          STa swap
27 loada    La 0
28 cmpb     CMPa 0
29          BMI noswap
30 loada2   La 0
31          STa temp
32 loadb    La 0
33 storeb   STa 0
34          La temp
35 storea   STa 0
36          L 1
37          STa swap
38 noswap   La idxa
39          CLC
40          ADD 1
41          STa idxa
42          La idxb
43          CLC
44          ADD 1
45          STa idxb
```

```
46          CMPa len                53          L 0
47          BEQ next                54          STa idxa
48          J start                 55          L 1
49                                  56          STa idxb
50  next    La swap                 57          J start
51          BEQ end                 58
52                                  59  end     J end
```

Erläuterung

Zeilen 1-2 Prolog: Der initiale Akkumulator-Wert ist 0, der Programmzähler startet bei Adresse `start` (s. Zeile 17)

Zeilen 4-8 Die zu sortierenden Elemente stehen im Datenteil eines beliebigen (Dummy-) Befehls

Zeile 10 Die Anzahl der zu sortierenden Elemente ist 5

Zeilen 12-13 Die Offsets für die indizierte Adressierung der jeweils zu vergleichenden Elemente wird auf die ersten beiden Elemente gesetzt (Offsets 0 und 1)

Zeile 14 `temp` ist ein Zwischenspeicher für das erste der zu vergleichenden Elemente

Zeile 15 `swap` ist das Abbruchkennzeichen, wenn kein Tausch mehr stattgefunden hat

Zeile 17-24 Die indizierten Lade- und Speicheradressen der zu vergleichenden Elemente werden vorbereitet

Zeile 25-26 Das Abbruchkennzeichen wird initialisiert

Zeile 27 Die Dummy-Ladeadresse 0 wurde zuvor mit dem Index `idxa` überschrieben (s. Zeile 18)

Zeile 28 Die Dummy-Vergleichsadresse wurde zuvor mit dem Index `idxb` überschrieben (s. Zeile 22)

Zeile 29 `BMI` verzweigt bei positivem Minus-Flag, d. h. wenn das Argument von `CMP` größer als der Akkumulatorinhalt ist

Zeilen 30-35 Elementetausch durchführen

Zeile 36 Tauschkennzeichen setzen (d. h. kein Abbruch)

Zeilen 38-45 Offsets für die indizierte Adressierung des Durchlaufs inkrementieren

Zeile 46-48 Wenn noch nicht alle Elemente verglichen wurden, nächsten Durchlauf starten

Zeile 50 Alle Elemente des Durchgangs verglichen: wenn kein Tausch vorgenommen, dann fertig

Zeilen 53-56 Offsets für indizierte Adressierung für nächsten Durchlauf zurücksetzen

Zeile 57 Nächsten Durchlauf starten

B.2 Kartenprüfung

Das Beispiel dieses Abschnittes berechnet die Prüfziffer einer Kreditkartennummer. Die Nummer selbst wird jeweils im Datenteil der Adressen $no1$ - $no16$ abgelegt. Das Ergebnis der Prüfung steht an Adresse res. Zum Prüfen einer Kreditkartennummer bzw. zum Erzeugen der Prüfziffer wird in der Praxis der Luhn-Algorithmus (auch als *Modulo 10*-Algorithmus bekannt) eingesetzt. Die Vorgehensweise ist dabei die folgende:

1. Von der ersten Ziffer der Kreditkartennummer an wird diese nach rechts durchlaufen, wobei der Wert jeder zweiten Ziffer verdoppelt wird (bei einigen Kartenemittenten braucht die erste Ziffer der Kartennummer nicht einbezogen zu werden; dies wird hier nicht beachtet)

2. Die einzelnen Ziffern bzw. Zahlen werden nun summiert. Bei der Verdopplung zweistellig gewordene Ziffern werden als Quersumme addiert

3. Ist das Ergebnis durch 10 ohne Rest teilbar, ist die Prüfziffer korrekt

Listing B.2: Kartenprüfung

```
1   INITAC 0
2   INITPC start
3
4   start   La no1
5           ROR
6           STa no1
7           La no3
8           ROR
9           STa no3
10          La no5
11          ROR
12          STa no5
13          La no7
14          ROR
15          STa no7
16          La no9
17          ROR
18          STa no9
19          La no11
20          ROR
21          STa no11
22          La no13
23          ROR
24          STa no13
25          La no15
26          ROR
27          STa no15
28
29  chk1    La no1
30          CMP 10
31          BMI chk3
32  add1    CLC
33          ADD 247
34          STa no1
35
36  chk3    La no3
37          CMP 10
38          BMI chk5
39  add3    CLC
40          ADD 247
41          STa no3
42
43  chk5    La no5
44          CMP 10
45          BMI chk7
46  add5    CLC
47          ADD 247
48          STa no5
49
50  chk7    La no7
51          CMP 10
52          BMI chk9
53  add7    CLC
54          ADD 247
55          STa no7
56
57  chk9    La no9
58          CMP 10
59          BMI chk11
60  add9    CLC
61          ADD 247
62          STa no9
63
64  chk11   La no11
65          CMP 10
66          BMI chk13
67  add11   CLC
68          ADD 247
69          STa no11
70
71  chk13   La no13
72          CMP 10
73          BMI chk15
74  add13   CLC
75          ADD 247
76          STa no13
77
78  chk15   La no15
79          CMP 10
80          BMI add17
81  add15   CLC
82          ADD 247
83          STa no15
84
85  add17   CLC
86          La no1
87          ADDa no3
88          ADDa no5
89          ADDa no7
90          ADDa no9
91          ADDa no11
92          ADDa no13
93          ADDa no15
94          STa temp
95          CLC
96          La no2
97          ADDa no4
98          ADDa no6
99          ADDa no8
100         ADDa no10
101         ADDa no12
102         ADDa no14
103         STa temp2
```

```
104             ADDa temp
105             STa temp3
106             STa res
107
108   sub       CLC
109             ADD 246
110             CMP 10
111             BMI ready
112             J sub
113
114   ready     STa res
115   end       J end
116
117   temp      L 0
118   temp2     L 0
119   temp3     L 0
120   res       L 0
121
122             L 99
123   no1       L 5
124   no2       L 4
125   no3       L 9
126   no4       L 7
127   no5       L 0
128   no6       L 3
129   no7       L 6
130   no8       L 5
131   no9       L 0
132   no10      L 2
133   no11      L 1
134   no12      L 6
135   no13      L 1
136   no14      L 6
137   no15      L 0
138   no16      L 0
```

Erläuterung

Zeilen 1-2 Prolog: Der initiale Akkumulator-Wert ist 0, der Programmzähler startet bei Adresse `start` (s. Zeile 4)

Zeilen 3-27 Jede zweite Ziffer der Kartennummer wird verdoppelt (`ROR` shiftet nach rechts, ergibt bei Little-Endian Systemen eine Multiplikation)

Zeilen 29-83 Für jede der verdoppelten Ziffern wird die Quersumme berechnet

Zeilen 85-106 Berechne die Summe der Ziffern

Zeilen 108-112 Berechne die abschließende Subtraktion (iterative Subtraktion von 10 bis als Ergebnis 10 bleibt)

Zeilen 117-119 `temp`, `temp2` und `temp3` speichern Additions-Zwischenergebnisse

Zeilen 123-138 `no1` - `no16` speichern die Ziffern der Kartennummer

Literaturverzeichnis

[1] Trusted Computing Group: TCG Specification Architecture Overview. – Forschungsbericht. – Revision 1.4, August 2007

[2] Abadi, Martín ; Wobber, Ted: A logical account of NGSCB. In: *In FORTE '04*, SpringerVerlag, 2004, S. 1–12

[3] Ambrose, Jude A. ; Parameswaran, Sri ; Ignjatovic, Aleksandar: MUTE-AES: a multiprocessor architecture to prevent power analysis based side channel attack of the AES algorithm. In: *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*. Piscataway, NJ, USA : IEEE Press, 2008 (ICCAD '08). – ISBN 978–1–4244–2820–5, 678–684

[4] Bain, Alex ; Mitchell, John ; Sharma, Rahul ; Stefan, Deian ; Zimmerman, Joe: A Domain-Specific Language for Computing on Encrypted Data. In: *FSTTCS 2011*, LIPIcs, December 2011. – Invited paper.

[5] Biondi, P ; Desclaux, F: *Silver needle in the Skype*. Bd. 6. BlackHat Europe, 2006

[6] Blake, Ian ; Kolesnikov, Vladimir: Conditional Encrypted Mapping and Comparing Encrypted Numbers. Version: 2006. `http://dx.doi.org/10.1007/11889663_18`. In: Di Crescenzo, Giovanni (Hrsg.) ; Rubin, Avi (Hrsg.): *Financial Cryptography and Data Security* Bd. 4107. Springer Berlin / Heidelberg, 2006. – ISBN 978–3–540–46255–2, 206-220. – 10.1007/11889663_18

[7] Blake, Ian F. ; Kolesnikov, Vladimir: Strong Conditional Oblivious Transfer and Computing on Intervals. In: *IN ADVANCES IN CRYPTOLOGY - ASIACRYPT 2004*, Springer, 2004, S. 515–529

[8] Brakerski, Zvika ; Vaikuntanathan, Vinod: Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. Version: 2011. http://dx.doi.org/10.1007/978-3-642-22792-9_29. In: *Advances in Cryptology - CRYPTO 2011* Bd. 6841. Springer Berlin / Heidelberg, 2011. – ISBN 978–3–642–22791–2, 505-524. – 10.1007/978-3-642-22792-9_29

[9] Brenner, Michael ; Paya Vaya, Guillermo ; Blume, Holger ; Smith, Matthew: hChip - Geheime FPGA-basierte Programmausführung mit homomorpher Kryptografie. In: *BMBF Call: Validierung des Innovationspotenzials wissenschaftlicher Forschung (VIP)* (2012)

[10] Brenner, Michael ; Perl, Henning ; Smith, Matthew: How Practical is Homomorphically Encrypted Program Execution? An Implementation and Performance Evaluation. In: *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012

[11] Brenner, Michael ; Perl, Henning ; Smith, Matthew: Practical Applications of Homomorphic Encryption. In: *Proceedings of the 7th International Conference on Security and Cryptography (SECRYPT 2012)*, SciTePress, 2012

[12] Brenner, Michael ; Smith, Matthew: *Verfahren und Vorrichtung zur Berechnung homomorph verschlüsselter Programme, angemeldetes Patent, amtl. Az. 10 2011 012 328.8*

[13] Brenner, Michael ; Smith, Matthew: *Encrypted Computing, Patent pending, PCT/EP2012/053165*

[14] Brenner, Michael ; Wiebelitz, Jan ; Voigt, Gabriele von ; Smith, Matthew: Secret program execution in the cloud applying homomorphic encryption. In: *Digital Ecosystems and Technologies Conference (DEST), 2011 Proceedings of the 5th IEEE International Conference on*, 2011, S. 114 –119

[15] Brenner, Michael ; Wiebelitz, Jan ; Voigt, Gabriele von ; Smith, Matthew: A Smart-gentry based Software System for Secret Program

Execution. In: *Proceedings of the 6th International Conference on Security and Cryptography (SECRYPT 2011)*, SciTePress, 2011. – ISBN 978–989–8425–71–3, S. 238–244

[16] Chazelle, Bernard ; Kilian, Joe ; Rubinfeld, Ronitt ; Tal, Ayellet: The Bloomier filter: an efficient data structure for static support lookup tables. In: *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA : Society for Industrial and Applied Mathematics, 2004 (SODA '04). – ISBN 0–89871–558–X, 30–39

[17] Checkoway, Stephen ; Davi, Lucas ; Dmitrienko, Alexandra ; Sadeghi, Ahmad-Reza ; Shacham, Hovav ; Winandy, Marcel: Return-oriented programming without returns. In: *Proceedings of the 17th ACM conference on Computer and communications security*. New York, NY, USA : ACM, 2010 (CCS '10). – ISBN 978–1–4503–0245–6, 559–572

[18] Coron, Jean-Sébastien ; Mandal, Avradip ; Naccache, David ; Tibouchi, Mehdi: Fully Homomorphic Encryption over the Integers with Shorter Public Keys. Version: 2011. `http://dx.doi.org/10.1007/978-3-642-22792-9_28`. In: *Advances in Cryptology - CRYPTO 2011* Bd. 6841. Springer Berlin / Heidelberg, 2011. – ISBN 978–3–642–22791–2, 487-504. – 10.1007/978-3-642-22792-9_28

[19] Damgard, Ivan ; Geisler, Martin ; Kroigaard, Mikkel: Efficient and Secure Comparison for On-Line Auctions. Version: 2007. `http://dx.doi.org/10.1007/978-3-540-73458-1_30`. In: Pieprzyk, Josef (Hrsg.) ; Ghodosi, Hossein (Hrsg.) ; Dawson, Ed (Hrsg.): *Information Security and Privacy* Bd. 4586. Springer Berlin / Heidelberg, 2007. – ISBN 978–3–540–73457–4, 416-430. – 10.1007/978-3-540-73458-1_30

[20] Damgard, Ivan ; Meldgaard, Sigurd ; Nielsen, Jesper: Perfectly Secure Oblivious RAM without Random Oracles. Version: 2011. `http://dx.doi.org/10.1007/978-3-642-19571-6_10`. In: Ishai, Yuval (Hrsg.): *Theory of Cryptography* Bd. 6597. Springer Berlin / Heidelberg, 2011. – ISBN 978–3–642–19570–9, 144-163. – 10.1007/978-3-642-19571-6_10

[21] Diffie, W. ; Hellman, M.: New directions in cryptography. In: *Information Theory, IEEE Transactions on* 22 (1976), nov, Nr. 6, S.

644 – 654. http://dx.doi.org/10.1109/TIT.1976.1055638. – DOI 10.1109/TIT.1976.1055638. – ISSN 0018–9448

[22] Dijk, Marten van ; Gentry, Craig ; Halevi, Shai ; Vaikuntanathan, Vinod: Fully Homomorphic Encryption over the Integers. Version: 2010. http://dx.doi.org/10.1007/978-3-642-13190-5_2. In: *Advances in Cryptology - EUROCRYPT 2010* Bd. 6110. Springer Berlin / Heidelberg, 2010. – ISBN 978–3–642–13189–9, 24-43. – 10.1007/978-3-642-13190-5_2

[23] Even, Shimon ; Goldreich, Oded ; Lempel, Abraham: A randomized protocol for signing contracts. In: *Commun. ACM* 28 (1985), Juni, Nr. 6, 637–647. http://dx.doi.org/10.1145/3812.3818. – DOI 10.1145/3812.3818. – ISSN 0001–0782

[24] Feistel, Horst: In: *Scientific American* 228 (1973), Nr. 5, 15–23 S.

[25] Foster, Ian (Hrsg.) ; Kesselman, Carl (Hrsg.): *The grid: blueprint for a new computing infrastructure*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1999. – ISBN 1–55860–475–8

[26] Garay, Juan ; Schoenmakers, Berry ; Villegas, Jose: Practical and Secure Solutions for Integer Comparison. Version: 2007. http://dx.doi.org/10.1007/978-3-540-71677-8_22. In: Okamoto, Tatsuaki (Hrsg.) ; Wang, Xiaoyun (Hrsg.): *Public Key Cryptography - PKC 2007* Bd. 4450. Springer Berlin / Heidelberg, 2007. – ISBN 978–3–540–71676–1, 330-342. – 10.1007/978-3-540-71677-8_22

[27] Gentry, Craig: Fully homomorphic encryption using ideal lattices. In: *Proceedings of the 41st annual ACM symposium on Theory of computing*. New York, NY, USA : ACM, 2009 (STOC '09). – ISBN 978–1–60558–506–2, 169–178

[28] Gentry, Craig ; Halevi, Shai: Implementing Gentry's Fully-Homomorphic Encryption Scheme. Version: 2011. http://dx.doi.org/10.1007/978-3-642-20465-4_9. In: Paterson, Kenneth (Hrsg.): *Advances in Cryptology – EUROCRYPT 2011* Bd. 6632. Springer Berlin / Heidelberg, 2011. – ISBN 978–3–642–20464–7, 129-148. – 10.1007/978-3-642-20465-4_9

[29] Gentry, Craig ; Halevi, Shai ; Vaikuntanathan, Vinod: i-Hop Homomorphic Encryption and Rerandomizable Yao Circuits. Version: 2010. http://dx.doi.org/10.1007/978-3-642-14623-7_9. In: Rabin, Tal (Hrsg.): *Advances in Cryptology - CRYPTO 2010* Bd. 6223. Springer Berlin / Heidelberg, 2010. – ISBN 978–3–642–14622–0, 155-172. – 10.1007/978-3-642-14623-7_9

[30] Gentry, Craig ; Halevi, Shai ; Vaikuntanathan, Vinod: i-Hop Homomorphic Encryption and Rerandomizable Yao Circuits. Version: 2010. http://dx.doi.org/10.1007/978-3-642-14623-7_9. In: Rabin, Tal (Hrsg.): *Advances in Cryptology - CRYPTO 2010* Bd. 6223. Springer Berlin / Heidelberg, 2010. – ISBN 978–3–642–14622–0, 155-172. – 10.1007/978-3-642-14623-7_9

[31] Goldreich, Oded: Towards a theory of software protection and simulation by oblivious RAMs. In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. New York, NY, USA : ACM, 1987 (STOC '87). – ISBN 0–89791–221–7, 182–194

[32] Goldreich, Oded ; Ostrovsky, Rafail: Software protection and simulation on oblivious RAMs. In: *J. ACM* 43 (1996), May, 431–473. http://dx.doi.org/http://doi.acm.org/10.1145/233551.233553. – DOI http://doi.acm.org/10.1145/233551.233553. – ISSN 0004–5411

[33] Goodrich, Michael ; Mitzenmacher, Michael: Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation. Version: 2011. http://dx.doi.org/10.1007/978-3-642-22012-8_46. In: Aceto, Luca (Hrsg.) ; Henzinger, Monika (Hrsg.) ; Sgall, Jirí (Hrsg.): *Automata, Languages and Programming* Bd. 6756. Springer Berlin / Heidelberg, 2011. – ISBN 978–3–642–22011–1, 576-587. – 10.1007/978-3-642-22012-8_46

[34] Greene, James ; Datta, Sham: White Paper - Intel Trusted Execution Environment / Intel Corp. 2007. – Forschungsbericht. – 0110/JG/EMC/PDF 323586-001US

[35] Groth, Jens: Cryptography in Subgroups of $\mathbb{Z}_n^*$. Version: 2005. http://dx.doi.org/10.1007/978-3-540-30576-7_4. In: Kilian, Joe (Hrsg.):

Theory of Cryptography Bd. 3378. Springer Berlin / Heidelberg, 2005. – ISBN 978–3–540–24573–5, 50-65. – 10.1007/978-3-540-30576-7_4

[36] Hennessy, John L. ; Patterson, David A.: *Computer Architecture. A Quantitative Approach, 4th rev. ed.* Academic Press, 2006. – ISBN 978–0123704900

[37] Henry, Kevin: The Theory and Applications of Homomorphic Cryptography (Master Thesis). University of Waterloo, Ontario, Canada, 2008

[38] Howe, DC: TrackMeNot: Resisting surveillance in web search. In: *Lessons from the Identity Trail: Anonymity* (2009)

[39] Kak, Avinash: *Lecture Notes on Computer and Network Security*. Purdue University, 2012

[40] Kocher, Paul ; Jaffe, Joshua ; Jun, Benjamin: Differential Power Analysis. Version: 1999. http://dx.doi.org/10.1007/3-540-48405-1_25. In: Wiener, Michael (Hrsg.): *Advances in Cryptology — CRYPTO' 99* Bd. 1666. Springer Berlin / Heidelberg, 1999. – ISBN 978–3–540–66347–8, 789-789. – 10.1007/3-540-48405-1_25

[41] Kolesnikov, Vladimir ; Sadeghi, Ahmad-Reza ; Schneider, Thomas: *How to Combine Homomorphic Encryption and Garbled Circuits Improved Circuits and Computing the Minimum Distance Efficiently*

[42] Kolesnikov, Vladimir ; Sadeghi, Ahmad-Reza ; Schneider, Thomas: Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. Version: 2009. http://dx.doi.org/10.1007/978-3-642-10433-6_1. In: *Cryptology and Network Security* Bd. 5888. Springer Berlin / Heidelberg, 2009. – ISBN 978–3–642–10432–9, 1-20. – 10.1007/978-3-642-10433-6_1

[43] Kolesnikov, Vladimir ; Schneider, Thomas: Improved Garbled Circuit: Free XOR Gates and Applications. Version: 2008. http://dx.doi.org/10.1007/978-3-540-70583-3_40. In: Aceto, Luca (Hrsg.) ; Damgård, Ivan (Hrsg.) ; Goldberg, Leslie (Hrsg.) ; Halldórsson, Magnús (Hrsg.) ; Ingólfsdóttir, Anna (Hrsg.) ; Walukiewicz, Igor (Hrsg.):

Automata, Languages and Programming Bd. 5126. Springer Berlin / Heidelberg, 2008. – ISBN 978–3–540–70582–6, 486-498. – 10.1007/978-3-540-70583-3_40

[44] Malkhi, Dahlia ; Nisan, Noam ; Pinkas, Benny ; Sella, Yaron: Fairplay - a secure two-party computation system. In: *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*. Berkeley, CA, USA : USENIX Association, 2004 (SSYM'04), 20–20

[45] Naehrig, Michael ; Lauter, Kristin ; Vaikuntanathan, Vinod: Can homomorphic encryption be practical? In: *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. New York, NY, USA : ACM, 2011 (CCSW '11). – ISBN 978–1–4503–1004–8, 113–124

[46] Nakamoto, Satoshi: Bitcoin: A Peer-to-Peer Electronic Cash System / bitcoin.org. 2008. – Forschungsbericht

[47] Nicolas Falliere, Eric C. Liam O Murchu M. Liam O Murchu: W32.Stuxnet Dossier. In: *Symantec Security Response*. Symantec Corp., 2011

[48] Patterson, D.A. ; Hennessy, J.L.: *Computer Organization and Design: The Hardware/software Interface*. Morgan Kaufmann, 2005

[49] Perl, Henning ; Brenner, Michael ; Smith, Matthew: POSTER: An Implementation of the Fully Homomorphic Smart-Vercauteren Crypto-System. In: *Computer and Communications Security, CD Proceedings of the 18th ACM Conference on*, 2011

[50] Pinkas, Benny ; Reinman, Tzachy: Oblivious RAM Revisited. Version: 2010. http://dx.doi.org/10.1007/978-3-642-14623-7_27. In: Rabin, Tal (Hrsg.): *Advances in Cryptology - CRYPTO 2010* Bd. 6223. Springer Berlin / Heidelberg, 2010. – ISBN 978–3–642–14622–0, 502-519. – 10.1007/978-3-642-14623-7_27

[51] Rabin, Michael O.: How To exchange Secrets with Oblivious Transfer / Aiken Computation Lab, Harvard University. 1981. – Forschungsbericht

[52] Reid, F. ; Harrigan, M.: An Analysis of Anonymity in the Bitcoin System. In: *Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom)*, 2011, S. 1318 –1326

[53] Rüter, Horst: Zahlung und Kundenbindung per Karte. In: *EHI-Research: Anteile der Zahlungsarten am Umsatz des Einzelhandels* (2010)

[54] Schnorr, C. P.: *Gittertheorie und algorithmische Geometrie, Reduktion von Gitterbasen und Polynomidealen.* Johann Wolfgang Goethe-Universität Frankfurt/Main, 1994

[55] Smart, N. ; Vercauteren, F.: Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. Version: 2010. `http://dx.doi.org/10.1007/978-3-642-13013-7_25`. In: *Public Key Cryptography, PKC 2010* Bd. 6056. Springer Berlin / Heidelberg, 2010, 420-443

[56] Stehlé, Damien ; Steinfeld, Ron: Faster Fully Homomorphic Encryption. Version: 2010. `http://dx.doi.org/10.1007/978-3-642-17373-8_22`. In: *Advances in Cryptology - ASIACRYPT 2010* Bd. 6477. Springer Berlin / Heidelberg, 2010. – ISBN 978–3–642–17372–1, 377-394. – 10.1007/978-3-642-17373-8_22

[57] Trapnell, C. ; Salzberg, S. L.: How to map billions of short reads onto genomes. In: *Nature Biotechnology* 27 (2009), Nr. 5, S. 455

[58] Vaikuntanathan, V.: Computing Blindfolded: New Developments in Fully Homomorphic Encryption. In: *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, 2011. – ISSN 0272–5428, S. 5 –16

[59] Vollmer, Heribert: *Introduction to Circuit Complexity: A Uniform Approach.* Secaucus, NJ, USA : Springer-Verlag New York, Inc., 1999. – ISBN 3540643109

[60] Yao, Andrew C.: Protocols for secure computations. In: *SFCS '82: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science.* Washington, DC, USA : IEEE Computer Society, 1982. – ISSN 0272–5428

[61] Yao, Andrew Chi-Chih: How to generate and exchange secrets. In: *Foundations of Computer Science, Annual IEEE Symposium on* 0 (1986), S. 162–167. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/SFCS.1986.25. – DOI http://doi.ieeecomputersociety.org/10.1109/SFCS.1986.25. – ISSN 0272–5428

Lebenslauf

Persönliche Daten

Name:	Michael Brenner
Geburtsdaten:	7. August 1974 in Neustadt a. Rbge.
Familienstand:	verheiratet

Schulausbildung

07/1994	Abitur, Gymnasium Neustadt a. Rbge.

Wehrdienst

08/1994 - 07/1995	2./Trsp. Btl. 11, Dörverden Dienstgrad bei Dienstzeitende: Hauptgefreiter d. R.

Berufsausbildung

08/1995 - 07/1998	Ausbildung zum Datenverarbeitungskaufmann dvg Hannover mbH, BBS Alfeld

Berufliche Erfahrungen

08/1998 - 04/2008	Softwareentwickler und Softwarearchitekt dvg Hannover (zwischenzeitlich Finanz IT, heute Finanz Informatik)
- 2000:	Datenbankentwicklung (IBM Mainframe)
- 2002:	Netzwerkkomponentenentwicklung (IBM, Sun, JEE)
- 2008:	Middlewarekomponentenentwicklung (IBM, Sun, JEE)
seit 05/2008	Wissenschaftlicher Mitarbeiter Forschungszentrum L3S, Gottfried Wilhelm Leibniz Universität Hannover, Institut für Verteile Systeme, Fachgebiet Distributed Virtual Reality, Lehrgebiet

	Rechnernetze; BMBF-gefördertes Projekt DGI-2, BMBF-gefördertes Projekt ASEV

Wissenschaftliche Ausbildung

09/2000 - 12/2003	Studium der angewandten Informatik Fachhochschule Hannover, Fachbereich Informatik Abschluss: Diplom-Informatiker (FH)
09/2004 - 08/2006	Studium der angewandten Informatik Fachhochschule Hannover, Fachbereich Informatik Spezialisierung: Sichere Informationssysteme Abschluss: Master of Science
seit 05/2008	Doktorand Forschungszentrum L3S, Gottfried Wilhelm Leibniz Universität Hannover

Workshop Chair

04/2013	17th Intl. Conf. on Financial Cryptography and Data Security, WAHC'13: Workshop on Applied Homomorphic Cryptography; Veröffentlicht in Springer *Lecture Notes On Computer Science (LNCS)*

Publikationen mit wiss. Qualitätssicherung

M. Brenner, H. Perl, M. Smith (2012): *How Practical is Homomorphically Encrypted Program Execution? An Implementation and Performance Evaluation*, Proceedings of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-2012), ISBN 978-0-7695-4745-9/12, pp. 375-382, Liverpool, UK

M. Brenner, H. Perl, M. Smith (2012): *Practical Applications of Homomorphic Encryption*, Proceedings of the 7th International Conference on Security and Cryptography (SECRYPT 2012), ISBN 978-989-8565-24-2, pp. 5-14, Rom, Italien

H. Perl, Y. Mohammed, M. Brenner, M. Smith (2012): *Fast Confidential Search for Bio-Medical Data Using Bloom Filters and Homomorphic Cryptography*, Proceedings of the 8th IEEE International Conference on eScience (eScience 2012), *erscheint am 8. Oktober 2012 (IEEE)*, Chicago, USA

M. Harbach, S. Fahl, M. Brenner, T. Muders and M. Smith (2012): *Towards Privacy-Preserving Access Control with Hidden Policies, Hidden Credentials and Hidden Decisions*, Proceedings of the 10th Annual Conference on Privacy, Security and Trust (PST 2012), IEEE press, ISBN 978-1-4673-2323-9, pp. 17-24, Paris, Frankreich

H. Perl, M. Brenner, M. Smith (2011): *An Implementation of the Fully Homomorphic Smart-Vercauteren Cryptosystem (Poster)*, Proceedings of the 18th ACM Conference on Computer and Communications Security (ACM CCS 2011), ISBN 978-1-4503-0948-6, pp. 837-840, Chicago, USA

M. Brenner, J. Wiebelitz, G. v. Voigt, M. Smith (2011): *A Smart-Gentry based Software System for Secret Program Execution*, Proceedings of the 6th International Conference on Security and Cryptography (SECRYPT 2011), ISBN 978-989-8425-71-3, pp. 238-244, Sevilla, Spanien

M. Brenner, J. Wiebelitz, G. v. Voigt, M. Smith (2011): *Secret Program Execution in the Cloud applying Homomorphic Encryption*, Proceedings of the 5th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2011), ISBN 978-1-4577-0871-8, pp. 114-119, Daejeon, Südkorea

M. Brenner, J. Wiebelitz, M. Smith (2010): *A JEE-based Architecture for Distributed Multi-Domain Resource Accounting*, Proceedings of the 6th IEEE International Conference on eScience Workshops (eScienceW 2010), ISBN 978-0-7695-4295-9/10, pp. 31-38, Brisbane, Australien

J. Wiebelitz, M. Brenner, C. Kunz, M. Smith (2010): *Early Defense - Enabling Attribute-Based Authorization in Grid Firewalls*, Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (ACM HPDC 2010), ISBN 978-1-60558-942-8, pp. 336-339, Chicago, USA

J. Wiebelitz, W. Müller, M. Brenner, G. v. Voigt (2008): *Towards a Comprehensive Accounting Solution in the Multi-Middleware Environment of the D-Grid Initiative*, Proceedings of the Cracow Grid Workshop 2008, ISBN 978-83-61433-00-2, pp. 229-236, Krakau, Polen

Patente

M. Brenner, M. Smith(2011): *Verfahren und Vorrichtung zur Berechnung homomorph verschlüsselter Programme*, Deutsches Patent- und Markenamt (angemeldetes Patent, amtl. Az. 10 2011 012 328.8)

M. Brenner, M. Smith(2012): *Encrypted Computing*, International Patent Application (WIPO Appl. No. WO/2012/113908, Intl. Appl. No. PCT/EP2012/053165, Intl. Filing Date 24.02.2012, IPC G06F 7/00)