Studiengang:	Informatik
Prüfer:	Prof. Dr. Kurt Rothermel
Betreuer:	Dipl. Inf. Timo Heiber Dipl. Inf. Arno Wacker
begonnen am:	15. Mai 2003
beendet am:	14. November 2003
CR-Klassifikation:	C.2.0, C.3, E.3, J.7

Diplomarbeit Nr. 2106

Einsatz kryptographischer Verfahren auf stark ressourcenbeschränkten Geräten

Ulrich Wurst

Institut für Parallele und
Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Kurzfassung:

Diese Arbeit behandelt den Einsatz kryptographischer Verfahren auf ressourcenbeschränkten Plattformen. Dazu wird untersucht, mit welchen kryptographischen Verfahren und Algorithmen eine sichere Kommunikation zwischen zwei Teilnehmern aufgebaut werden kann. Weiterhin werden Metriken und Kriterien entwickelt, um die Anwendbarkeit dieser Verfahren und Algorithmen auf ressourcenschwachen Geräten bewerten. Diese Kriterien werden auf eine Auswahl kryptographischer Algorithmen angewandt um deren Eignung für die vorgegebene Umgebung der Heimautomatisierung zu beurteilen.

Inhaltsverzeichnis

1 Einleitung **11**
1.1 Motivation 12
1.2 Bestehende Arbeiten 14
1.3 Umfang der Arbeit 16
1.4 Gliederung der Arbeit 17

2 Kryptographische Grundlagen **19**
2.1 Sicherheitsziele 19
2.2 Kryptographisch abgesicherte Kommunikation 20
2.3 Terminologie 21
2.4 Stromchiffren 21
2.5 Blockchiffren 22
2.5.1 Electronic-Codebook-Mode (ECB-Mode) 24
2.5.2 Cipherblock-Chaining-Mode (CBC-Mode) 24
2.5.3 Cipher-Feedback-Mode (CFB-Mode) 25
2.5.4 Output-Feedback-Mode (OFB-Mode) 25
2.5.5 Counter-Mode (CTR-Mode) 26
2.6 Hashfunktionen 26
2.7 Message Authentication Code (MAC) 27
2.7.1 MACs auf Basis von Hashverfahren 28
2.7.2 MACs auf CBC-Basis 28
2.8 Kryptographische Systeme 29
2.8.1 AE-Kanäle 30
2.8.2 AEAD-Kanäle 31
2.9 Asymmetrische Verfahren 34
2.9.1 Digitale Signaturen 35
2.9.2 Zertifikate und Public-Key Infrastrukturen 35
2.10 Schlüsselgenerierung 36
2.11 Zufallszahlen 37
2.12 Angriffe auf kryptographische Algorithmen 38

3 Mathematik asymmetrischer Verfahren 43
3.1 Übersicht . . . 43
3.1.1 Gruppen . . . 44
3.1.2 Körper . . . 45
3.1.3 Elliptische Kurven . . . 45
3.2 Wahl des Körpers . . . 48
3.2.1 Primkörper . . . 48
3.2.2 Körper mit Charakteristik 2 . . . 49
3.2.3 Zusammengesetzte Körper . . . 49
3.2.4 Optimal Extension Fields . . . 49
3.3 Koordinaten-Darstellungen . . . 49
3.3.1 Affine Koordinaten . . . 50
3.3.2 Projektive Koordinaten . . . 50
3.4 EC-Diffie-Hellman Key-Agreement . . . 51
3.5 EC-DSA . . . 52
3.6 EC-ElGamal . . . 53
3.7 Zusammenfassung . . . 53

4 Implementierungskriterien 55
4.1 Implementierungsmöglichkeiten . . . 55
4.1.1 Softwareimplementierung auf bestehender Hardware . 55
4.1.2 Softwareimplementierung eines Krypto-Coprozessors . 56
4.1.3 Hardwareimplementierung eines Krypto-Coprozessors 56
4.2 Überblick über einige Hardwareplattformen . . . 58
4.2.1 Atmel AVR/MegaAVR Plattform . . . 59
4.2.2 Microchip PIC Plattform . . . 59
4.2.3 ST Microelectronics ST Plattform . . . 60
4.2.4 Intel 8051 Plattform . . . 60
4.2.5 Klassifikation der Plattformen . . . 61
4.2.6 Mehrkosten . . . 61
4.3 Eigenschaften kryptographischer Algorithmen . . . 62
4.3.1 RAM-Bedarf . . . 62
4.3.2 ROM-Bedarf . . . 63
4.3.3 Durchsatz . . . 63
4.3.4 RAM-Bedarf der Rundenfunktion . . . 63
4.3.5 Schlüsselgröße . . . 64
4.3.6 Key-Setup Aufwand . . . 64
4.3.7 Verwendete Operationen . . . 64
4.4 Eigenschaften sicherer Kanäle . . . 65
4.5 Zusammenfassung . . . 66

5 Kryptographische Algorithmen 67
5.1 RC5 67
5.1.1 Keyschedule 67
5.1.2 Verschlüsselung 68
5.1.3 Sicherheit 68
5.1.4 Implementierung 68
5.1.5 Bewertung 69
5.1.6 RC6 70
5.2 AES - Rijndael 70
5.2.1 Grundlagen des AES Algorithmus 70
5.2.2 Algebraische Grundlage des AES Algorithmus 71
5.2.3 Details des Algorithmus 72
5.2.4 Sicherheit 74
5.2.5 Implementierung 74
5.2.6 Bewertung 76
5.3 IDEA 77
5.3.1 Sicherheit 77
5.3.2 Implementierung 77
5.3.3 Bewertung 78
5.4 Skipjack 79
5.4.1 Sicherheit 80
5.4.2 Implementierung 81
5.4.3 Bewertung 81
5.5 RC4 81
5.5.1 Sicherheit 82
5.5.2 Implementierung 83
5.5.3 Bewertung 83
5.6 RIPEMD-128 84
5.6.1 Sicherheit 86
5.6.2 Implementierung 86
5.6.3 Bewertung 86
5.7 SHA-1 86
5.7.1 Sicherheit 87
5.7.2 Implementierung 88
5.7.3 Bewertung 88
5.8 MD2 88
5.8.1 Sicherheit 89
5.8.2 Implementierung 90
5.8.3 Bewertung 90
5.9 Überblick 91
5.10 Asymmetrische Verfahren 92
5.10.1 Sicherheit 92
5.10.2 Implementierung 93
5.10.3 Bewertung 94

5.11 Zufallszahlengenerator 95
5.12 Zusammenfassung 96

6 Zusammenfassung und Ausblick 99

Abbildungsverzeichnis

2.1 Struktur der i-ten Runde eines Feistel-Netzes 23
2.2 Aufbau eines Hashing Algorithmus 27
2.3 CBC-MAC Verfahren 29
2.4 OMAC Verfahren 30
2.5 EAX Verfahren 34

3.1 Addition zweier Kurvenpunkte P und Q 45
3.2 Verdoppelung des Kurvenpunktes P 46

5.1 Struktur des AES-Chiffriervorgangs 74
5.2 Struktur des IDEA-Chiffriervorgangs 78
5.3 A- und B-Rundenfunktionen des Skipjack Algorithmus 79
5.4 G-Funktion des Skipjack Algorithmus 80
5.5 Struktur des RIPEMD-128 Algorithmus 84
5.6 Hardware-Zufallszahlengenerator 96

Tabellenverzeichnis

2.1 Mögliche Reihenfolgen für Chiffrierung und Authentifikation . 30

3.1 Vergleich unterschiedlicher Koordinatendarstellung 51

4.1 Krypto-Coprozessoren 57
4.2 Parameter von Mikrocontrollern 60
4.3 Mehrkosten durch Verwendung eines MC mit vergrößerten RAM- und ROM-Kapazitäten 62
4.4 Mehrkosten durch Verwendung eines MC der leistungsstärkeren Familie 62

5.1 Leistungsdaten von RC5 69
5.2 Leistungsdaten von AES 75
5.3 Leistungsdaten von IDEA 79
5.4 Leistungsdaten von Skipjack 81
5.5 Leistungsdaten vom RC4 83
5.6 Leistungsdaten der RIPEMD-128 Hashfunktion 86
5.7 Rundenfunktionen von SHA-1 87
5.8 Leistungsdaten der SHA-1 Hashfunktion 88
5.9 Leistungsdaten der MD2 Hashfunktion 90
5.10 Anforderungen der Algorithmen 91
5.11 Ausstattung der Mikrocontroller 92
5.12 Sicherheitsäquivalenzen von symmetrischen und asymmetrischen Kryptoverfahren 93

Kapitel 1

Einleitung

Der Themenbereich des Ubiquitous Computing, also der Allgegenwärtigkeit der Informationsverarbeitung, der auch als Pervasive Computing bezeichnet wird, beschäftigt sich unter anderem mit der Vision, Informationsverarbeitung in Gegenstände des täglichen Lebens zu integrieren. Ziel ist es, Routineaufgaben zu vereinfachen oder allgemein den Bedienkomfort zu erhöhen. Beispiele für Anwendungen des Ubiquitous Computing lassen sich in allen Bereichen finden. Ein oft zitiertes Beispiel ist die „intelligenten Kaffeetasse“, die einem mit entsprechender Elektronik ausgestattetem Kaffeevollautomaten die vom Besitzer der Tasse bevorzugte Kaffeezubereitung übermittelt. Ein weiteres Beispiel sind RFID-Tags, Aufkleber in der Größe herkömmlicher Preisetiketten, die mit einem Mikrocontroller und einem Sender ausgestattet sind. Sie sollen als Nachfolger des Barcode ein berührungsloses maschinelles Auslesen von Produktinformationen an automatischen Kassensystemen oder bei der Inventarisierung ermöglichen. Die dieser Arbeit zugrundeliegende Anwendung soll die Heimautomatisierung sein. Bei der Heimautomatisierung kommunizieren eine Vielzahl von Sensoren wie Helligkeitsmesser oder Glasbruchmelder und Aktoren wie Schalter oder Chipkarten berührungsloser Schliesssysteme mit anderen Geräten wie Alarmanlagen oder allgemeiner Haussteuergeräten. Während dies bislang üblicherweise über elektronische Leitungen wie dem „Europäischen Installationsbus“ (EIB [22]) geschieht ist eine Tendenz wahrzunehmen, den Aufwand des Kabelverlegens, wo immer es möglich ist, durch Funkkommunikation zu umgehen. Beispiele hierfür sind das Funk-Fernwirksystem FS20 der Firma ELV [23] oder das Hybrid-System X10 des gleichnamigen Herstellers [74], das Funkkommunikation sowie Kommunikation über das Stromnetz verwendet. Dies ermöglicht nicht nur eine höhere Flexibilität sondern auch einen niedrigeren Preis und geringeren Aufwand bei der Einführung des Systems in einem Haus.

1.1 Motivation

Die Funkübertragung birgt jedoch gegenüber kabelgebundenen Systemen ein höheres Risiko der Ausspähung oder Manipulation der Kommunikation durch Unbefugte, da kein physikalischer Zugriff auf die Übertragungsleitungen mehr erforderlich ist. Beispielsweise können berührungslose Schliesssysteme anfällig gegenüber Angriffen durch Ausspähen der Funkkommunikation sein. Es ist möglich, in der Nähe eines Empfängers einer berührungslosen Schliessanlage einen zweiten Empfänger zu positionieren, der den Funkverkehr aufzeichnet. Wird in diesem Fall der Schliesscode eines Signalgebers ungeschützt übermittelt, ist es einem Angreifer möglich, durch erneutes Senden des aufgezeichneten Codes das Schloss zu öffnen. Entsprechendes gilt für Alarmanlagen die mit einem Funksignalgeber aktiv und inaktiv geschaltet werden. Auch hier kann der Funkverkehr von einem Angreifer protokolliert werden, der dann vor einem Einbruch auf diese Weise die Alarmanlage inaktiv schalten kann. Sowohl FS20 als auch X10 verfügen über keinerlei Sicherheitsmaßnahmen.

Die Steuerung in den hier betrachteten Systemen wird üblicherweise durch Mikrocontroller ausgeführt. Mikrocontroller sind Allzweck-CPUs die mit integriertem Hauptspeicher und oftmals integrierter Taktgenerierung ausgestattet sind. Sie erlauben es, die Steuerung mit Hilfe eines einzigen Bauteils zu implementieren. Man spricht daher auch von Embedded Systemen. An Hardware, die in Produkten der Heimautomatisierung verwendet wird, werden insbesondere drei Anforderungen gestellt:

- **Physikalische Größe:** Da viele und sehr spezialisierte Sensoren verwendet werden ergeben sich Einschränkungen bezüglich deren Größe. Eine Konsequenz daraus ist die Notwendigkeit, die kryptographische Absicherung der Kommunikation möglichst ohne zusätzliche Bauteile zu implementieren, also ausschließlich unter Verwendung der bereits vorhandenen Komponenten die zur Steuerung des Geräts verwendet werden.

- **Preis:** Bei Produkten der Heimautomatisierung handelt es sich um Massenprodukte. Da viele Sensoren, Aktoren und Steuerungsgeräte eingesetzt werden, müssen die Kosten der einzelnen Produkte niedrig sein. Dies erfordert unter Anderem den Einsatz handelsüblicher Komponenten anstelle von für diesen Einsatzzweck produzierten Spezialkomponenten. Dadurch ergeben sich ebenfalls Konsequenzen für die Leistungsfähigkeit der zur Steuerung verwendeten Mikrocontroller.

- **Energieverbrauch:** Um den Vorteil der durch die Funkkommunikation gewonnenen Flexibilität nicht wieder zu verlieren wird bei der Energieversorgung der Geräte überwiegend auf Batteriebetrieb gesetzt. Dadurch ist der Energieverbrauch der Systeme eine wichtige Größe, da er die maximaler Laufzeit der Geräte festlegt. Da ein Ausfall der Batterie, beispielsweise bei einem Rauchmelder oder einer Komponente einer Alarmanlage, ein hohes Risiko mit sich bringt ist eine lange Laufzeit ein entscheidendes Merkmal eines Systems. Diese Einschränkung gilt nicht für die Steuergeräte oder manche Klassen von Geräten wie ferngesteuerte Steckdosen, da diese über einen Stromanschluss verfügen.

Als Konsequenz aus diesen Anforderungen ergeben sich für Mikrocontroller, die in Produkten der Heimautomatisierung verwendet werden, Einschränkung in folgenden Bereichen:

- **RAM:** Die RAM Kapazität der verwendeten Mikrocontroller beträgt, wie in Kapitel 4 besprochen wird, bei kleineren Geräten zwischen 100 und 512 Byte, bei leistungsfähigeren Modellen zwischen 512 und 2048 Byte. Dadurch ergeben sich starke Einschränkungen in Bezug auf die verwendbaren Algorithmen. Insbesondere viele Optimierungen, die auf vorberechnete und im RAM abgelegte Daten angewiesen sind, können auf diesen Mikrocontrollern nicht durchgeführt werden.

- **ROM:** Die in Betracht gezogenen Mikrocontroller verfügen über bis zu 64 Kilobyte ROM. Da die eigentliche Aufgabe der Mikrocontroller, beispielsweise die Messung der Temperatur und die Übermittlung der Messdaten an ein Steuerungssystem, verhältnismäßig einfach ist, kann ein Großteil der ROM Kapazität für Sicherheitsfunktionen verwendete werden. Damit ist die zur Verfügung stehende ROM-Kapazität eine weniger starke Beschränkung als die geringe RAM-Größe. Da jedoch Mikrocontroller in Familien sehr ähnlicher Geräte erhältlich sind, die sich oft lediglich in den unterschiedlichen RAM-/ROM-Kapazitäten unterscheiden, ist ein sparsamer Umgang mit diesen Ressourcen dennoch erstrebenswert. Dadurch können die Produktionskosten des Gesamtsystems durch den Einsatz eines weniger leistungsfähigen Mikrocontrollers gesenkt werden.

- **Dauerhafter Datenspeicher:** Da bei Batterieausfall der Inhalt des RAM in der Regel gelöscht wird wird eine Möglichkeit benötigt, dauerhaft Daten zu speichern die auch nach einem Stromausfall weiterhin verfügbar sind. Ein Anwendungsbeispiel sind Konfigurationsdaten

oder zuvor zwischen zwei Geräten vereinbare Schlüssel kryptographischer Algorithmen. In Mikrocontrollern wird die dauerhafte Datenspeicherung üblicherweise mittels EEPROM-Speicher gelöst. Da nur wenige Kilobyte EEPROM Speicher in Mikrocontrollern integriert sind, ist es erforderlich mit einem Minimum an Speicher auszukommen.

- **CPU-Leistung:** Die CPU-Leistung der handelsüblichen Mikrocontroller ist üblicherweise mehr als ausreichend um mittels symmetrischer Kryptographie die üblicherweise ohnehin relativ langsame Funkkommunikation abzusichern. Geräte, die für die Heimautomatisierung verwendet werden, können abhängig von der gewählten Modulation Übertragungsraten zwischen 10 [58] und 76 kBit pro Sekunde [35] erreichen. Allerdings kann es wünschenswert sein, die Taktrate des Mikrocontrollers zu verringern um den Energieverbrauch zu reduzieren . Somit ist die Geschwindigkeit der eingesetzten Algorithmen von Bedeutung.

- **Bandbreite der Kommunikation:** Oftmals ist die Verwendung der Funkschnittstelle in Hinblick auf den Energieverbrauch die teuerste Operation des Systems. Dadurch ist es unumgänglich, die durch die kryptographische Absicherung der Kommunikation gegebenenfalls notwendige Vergrößerung des Bandbreitenbedarfs möglichst gering zu halten.

- **Schutz vor Manipulation:** Für Mikrocontroller mit hohen Anforderungen an die Sicherheit sind Modifikationen erforderlich, um das Gerät vor Manipulationen zu schützen. Ansonsten kann es möglich sein, bei physikalischem Zugriff auf das Gerät durch Spannungsmessungen einen geheimen Schlüssel auszulesen oder eine unverschlüsselte Kommunikation zwischen zwei Bauteilen mitzuschneiden. Allerdings sind die Mehrkosten manipulationsgeschützter Komponenten üblicherweise zu hoch, um sie in Massenartikeln einzusetzen. Daher muss ein Gerät, auf das ein Angreifer physikalischen Zugriff hat, als kompromittiert angesehen werden und dieser Fall durch Maßnahmen auf höheren Sicherheitsschichten berücksichtigt werden.

1.2 Bestehende Arbeiten

Für eine sichere Kommunikation zwischen ressourcenbeschränkten Geräten existieren bereits viele spezialisierte Ansätze. Seitens der Hersteller von Mikrocontrollern werden für die sichere Übertragung von Signalen proprietäre

Verfahren angeboten, beispielsweise das Keeloq-Verfahren des Herstellers Microchip [37]. Dieses wird in Funksignalgebern zur Aktivierung von Zentralverriegelungssystemen in PKWs oder bei Funköffnern von Garagentoren verwendet. Speziell beim Keeloq-Verfahren wird der kryptographische Algorithmus geheimgehalten, so dass die Sicherheit des Verfahrens nie öffentlich verifiziert wurde. Das Keeloq-Verfahren erlaubt lediglich die Übermittlung von maximal vier Bit als Nutzdaten in einem 66 Bit grossen Codewort und kann somit nur als sehr einfacher Aktor eingesetzt werden. Die Nutzdaten werden unverschlüsselt übertragen, ihre Integrität und Authentizität ist jedoch sichergestellt. Die Zuordnung von Sendern zu einem Empfänger erfolgt in einer speziellen Lernphase in der der Empfänger alle in dieser Zeit aktivierten Sender akzeptiert und zuvor akzeptierte Empfänger ausschließt. Da das Lernen der zu akzeptierenden Sender bereits über die Funkschnittstelle durchgeführt wird besteht während der Lernphase ein hohes Sicherheitsrisiko.

In leistungsfähigeren Systemen wie PCs wird anstelle einer solchen Lernphase oder vorab übermittelten Geheimnissen die Übermittlung der symmetrischen Sitzungsschlüssel mit Hilfe von asymmetrischer Kryptographie abgesichert. Auch auf ressourcenbeschränkten Plattformen existieren Implementierungen asymmetrischer kryptographischer Algorithmen. Insbesondere asymmetrische Verfahren, die auf der Elliptischen Kurven Kryptographie (ECC) basieren eignen sich für diese Plattformen, da sie deutlich kleinere Schlüssel im Vergleich zu nicht-ECC-Verfahren verwenden. Implementierungen von ECC-Verfahren auf ressourcenbeschränkten Systemen werden unter anderem in [34] und [73] besprochen. Wie in Kapitel 5 besprochen wird ist deren Anwendung jedoch auf leistungsstärkere CPUs beschränkt als in den meisten Geräten der Heimautomatisierung zu finden sind. Daher müssen für diese Geräte andere Verfahren entwickelt werden.

Die Arbeit „The Resurrecting Duckling“ [70] schlägt zur Einbindung von Geräten in bestehendes System vor, diese bei Auslieferung in einen Lernmodus zu versetzen. Das erste System zu dem sich das neue Gerät verbindet wird dann als zukünftiges System akzeptiert. Dieser Vorgang wird als „imprinting“ bezeichnet. Typischerweise dürfte ein solches Gerät eine Resetschaltung beinhalten, mit der es in diesen Urzustand zurückversetzt werden kann. Als Verbindungsmöglichkeit um das imprinting durchzuführen wird der physikalische Kontakt zweier Geräte vorgeschlagen. Damit wird ein sicherer Kanal zwischen beiden Geräten gebildet, über den beispielsweise ein gemeinsamer Schlüssel eines kryptographischen Verfahrens ausgetauscht werden kann.

Das Projekt „Secure Pebblenets“ [4] behandelt die Absicherung der Kommunikation in grossen Netzen sehr kleiner Geräten. Ein mögliches Anwendungsgebiet sind militärische Sensornetze in denen eine große Zahl aus einem Flugzeug abgeworfener Sensoren Feindbewegungen registrieren und melden. Eine Annahme des Projekts ist die Resistenz der Geräte gegenüber Mani-

pulation da die höheren Kosten hierfür in militärischen Anwendungen akzeptiert werden. Die Geräte – Pebbles genannt – werden vor ihrem Einsatz mit einem gemeinsamen Gruppenschlüssel versehen mit dessen Hilfe die Kommunikation mittels symmetrischen Verfahren abgesichert wird. Die Verwendung eines einzelnen gemeinsamen Geheimnisses verbietet sich jedoch in der erwünschten Umgebung der Heimautomatisierung, da dort die Manipulationsresistenz der Geräte nicht vorausgesetzt werden kann. Somit würde die Kompromittierung eines einzelnen Geräts die Kompromittierung des Gesamtsystems bedeuten.

Ein weiteres Projekt, das sich mit der Thematik „Sicherheit in Sensornetzen“ beschäftigt, ist SPINS (Security Protocols for Sensor Networks) [58]. Wie in „Secure Pebblenets“ wird die Kommunikation mittels symmetrischer Kryptographie abgesichert, jedoch ohne die Annahme, dass den einzelnen Sensoren vertraut werden kann, da aus Kostengründen keine manipulationssichere Hardware verwendet wird. SPINS verwendet statt dessen eine vertrauenswürdige Basisstation, die mit den einzelnen Knoten symmetrische Schlüssel vereinbart, die vor Inbetriebname eines Sensors auf sicherem Wege übermittelt werden. In SPINS werden handelsübliche Mikrocontroller eingesetzt, die Verschlüsselung der Daten wird mit Hilfe der Blockchiffre RC5 durchgeführt. Die Zentralisierung des Systems ist jedoch im Szenario der Heimautomatisierung oftmals nicht erwünscht, da der Ausfall des zentralen Steuerungssystems den Ausfall der gesamte Heimautomatisierung bedeuten würde. Zudem könnten Skalierbarkeitsprobleme beim Einsatz eines Heimautomatisierungssystems in grossen Bürogebäuden auftreten.

1.3 Umfang der Arbeit

Diese Arbeit beschäftigt sich mit den Algorithmen und Verfahren die zur Sicherstellung von Vertraulichkeit, Integrität und Authentizität der Kommunikation erforderlich sind. Insbesondere wird die Anwendbarkeit dieser Verfahren in stark ressourcenbeschränkten Systemen untersucht. Die Arbeit ist Teil einer Forschungsarbeit der Universität Stuttgart zur Entwicklung von Sicherheitsverfahren für stark ressourcenbeschränkte Geräte in Ubiquitous Computing integriert. Die Algorithmen und Protokolle, die zur Sicherstellung der Verbindlichkeit und zur Authentifikation erforderlich sind, sind Bestandteil einer anderen Arbeit, ebenso die sichere Schlüsselverteilung [15]. In einer weiteren Arbeit [57] wird eine Zugriffskontrolle für dieses Szenario entwickelt.

Es wird lediglich die Funk-Kommunikation selbst geschützt. Wird ein teilnehmendes Gerät kompromittiert, kann ein Angreifer die Nachrichten an dieses Gerät entschlüsseln und Nachrichten mit der Identität des Geräts erstellen. Dies ist im vorgegebenen Szenario immer möglich da aus Kostengründen nicht auf manipulationsgeschützte Hardware zurückgegrif-

fen werden kann. Da ein geheimer Schlüssel im Gerät vorhanden sein muss, kann dieser mittels physikalischer Methoden wie der Messung der Spannung an EEPROM- oder SRAM-Zellen oder der Datenbusse ausgelesen werden [3, 47]. Die Fähigkeit des Gesamtsystems, einer Kompromittierung einzelner Geräte widerstehen zu können, ist ebenfalls Teil der Arbeit von Holger Cermann [15].

1.4 Gliederung der Arbeit

In Kapitel 2 werden Grundlagen der Kryptographie vorgestellt und die zur Implementierung eines sicheren Kommunikationskanals erforderlichen Verfahren angesprochen. Kapitel 3 beschreibt die Grundlagen, die zum Verständnis der Elliptischen Kurven Kryptographie erforderlich sind. In Kapitel 4 werden Bewertungskriterien und Vergleichsmetriken der kryptographischen Verfahren und der verwendeten Mikrocontroller aufgestellt. Das darauf folgende Kapitel 5 stellt Details ausgesuchter kryptographischer Algorithmen vor und führt eine Bewertung mit Hilfe der zuvor festgelegten Kriterien durch. Abschließend werden in Kapitel 6 die Erkenntnisse der vorangegangen Kapitel zusammengefasst und es wird ein Ausblick auf mögliche zukünftige Weiterentwicklungen gegeben.

Kapitel 2

Kryptographische Grundlagen

Dieses Kapitel beschäftigt sich mit den Zielen eines sicheren Kommunikationssystems und den kryptographischen Verfahren, mit denen sich diese Ziele erreichen lassen. Dazu werden Verfahren wie symmetrische und asymmetrische Verschlüsselungsverfahren, Hashingverfahren, digitale Signaturen und Zertifikate vorgestellt und ihre prinzipielle Funktionsweise erläutert. Die detaillierte Funktionsweise und Bemerkungen zu ihrer Implementierung werden in den folgenden Kapiteln besprochen.

2.1 Sicherheitsziele

Die grundlegende Sicherheit in Kommunikationssystemen lässt sich üblicherweise mit den vier folgenden Anforderungen beschreiben [53]:

- **Vertraulichkeit:** Vertraulichkeit bedeutet, dass niemand außer den beiden Kommunikationspartnern oder einer Gruppe von Kommunikationspartnern in der Lage ist, die übertragenen Daten im Klartext zu lesen. Vertraulichkeit in symmetrischen Kryptosystemen wird durch die Chiffrierung der zu übertragenden Daten mit dem gemeinsamen Schlüssel erreicht, in einem asymmetrischen System durch die Chiffrierung mit dem öffentlichen Schlüssel des Empfängers.

- **Integrität/Authentizität:** Die Integrität einer Nachricht ist dann gewährleistet, wenn eine Manipulation der Nachricht vom Empfänger sicher bemerkt wird. Üblicherweise wird auch vom Kommunikationssystem bereits eine Integritätsicherung mittels Prüfsummen durchgeführt, um Übertragungsfehler zu erkennen. Im Gegensatz dazu ist bei der Integritätsicherung zum Schutz vor böswilligen Manipulation ein Verwenden von Prüfsummenverfahren wie CRC oder Hashing-

Algorithmen nicht ausreichend. Da vorausgesetzt werden kann, dass einem Angreifer die verwendeten Prüfsummenalgorithmen bekannt sind, ist es diesem möglich, zu einer manipulierten Nachricht eine korrekte Prüfsumme neu zu berechnen. Daher ist es erforderlich, neben der Integrität die Authentizität der Daten sicher zu stellen. Beides kann gemeinsam durch den Einsatz von Message Authentication Codes gewährleistet werden.

- **Authentifikation:** Ein sicheres Kommunikationssystem muss in der Lage sein, die Identität eines Kommunikationspartners zweifelsfrei feststellen zu können. Eine Möglichkeit dies zu gewährleisten liegt in der digitalen Signatur der Nachricht. Zur Verringerung des Verwaltungsaufwands werden oftmals zusätzlich Zertifikate und Zertifizierungsstellen verwendet. In symmetrischen System kann die Authentifikation lediglich mit Hilfe eines zuvor auf sicherem Wege übermittelten gemeinsamen Geheimnisses durchgeführt werden.

- **Verbindlichkeit:** Ein sicheres Kommunikationssystem muss verbindlich die Übertragung einer Nachricht protokollierbar machen. Dazu gehört die Nachweisbarkeit des Nachrichteneingangs sowie die Unmöglichkeit, eine abgesandte Nachricht abzuleugnen. Die Sicherstellung der Verbindlichkeit sind Aufgaben höherer Protokollschichten und sollen in dieser Arbeit nicht weiter betrachtet werden.

2.2 Kryptographisch abgesicherte Kommunikation

Abhängig vom Angreifermodell und der erforderlichen Sicherheit können unterschiedliche Arten von kryptographisch abgesicherten Kanälen verwendet werden. Die einfachste Absicherung stellt lediglich die Vertraulichkeit der übertragenen Daten durch Verschlüsselung sicher. Wenn nicht die Vertraulichkeit der übertragenen Nachrichten sondern ausschließlich deren Integrität und Authentizität geschützt werden soll genügt es, die Nachrichten mit einem Message Authentication Code zu versehen. In fast allen Anwendungsbereichen ist es jedoch erforderlich, sowohl die Vertraulichkeit als auch die Integrität und Authentizität der Nachrichten sicherzustellen. In diesem Fall spricht man von einem AE-Schema (Authenticated Encryption). Oftmals ist es erwünscht, zusätzlich zu den Daten die durch AE abgesichert sind, zumindest die Integrität weiterer unverschlüsselter Daten sicherzustellen. Dies sind beispielsweise die Protokollköpfe der Netzwerk- und Transport-Protokolle, die nicht verschlüsselt werden können, da ansonsten der Transport der Daten nicht durchgeführt werden kann. Bei manchen Anwendungen muss deren

Integrität jedoch gewährleistet sein. In diesem Fall spricht man von einem AEAD-Schema (Authenticated Encryption with Associated Data). Im Folgenden werden die kryptographischen Bausteine beschrieben, die für ein AE- oder AEAD-Schema erforderlich sind.

2.3 Terminologie

In der Kryptographie wird eine Verschlüsselungsfunktion $E_{K_e} : M^* \rightarrow M^*$ zusammen mit dem Geheimnis (oder Schlüssel) K_e verwendet, um eine Klartext-Nachricht $x \in M^*$ in das Chiffrat $c \in M^*$ zu überführen. Die Menge M wird als Alphabet bezeichnet. Die zu E_{K_e} inverse Funktion $D_{K_d} = E_{K_e}^{-1}$ überführt eine chiffrierte Nachricht c zurück in die Klartext-Nachricht x. Es gilt $D_{K_d}(E_{K_e}(x)) = x$. Gilt $K_e = K_d$, so spricht man von einem symmetrischen System, andernfalls von einem Asymmetrischen.

Die Chiffrierfunktion hat in einem symmetrischen Kryptosystem dabei die Aufgaben der Konfusion und der Diffussion. Beide Begriffe gehen auf eine Arbeit von Claude Shannon [68] zurück und werden als die Grundbausteine der symmetrischen Kryptographie bezeichnet. Konfusion wird durch eine Substitution der Zeichen des Klartextes verursacht und bewirkt die Unverkettbarkeit von Chiffrat und Schlüssel sowie zwischen Klartext und Chiffrat. Diffusion wird durch Transposition und Permutation des Klartextes erreicht und hat zum Ziel, dass jedes Bit des Chiffrat von möglichst vielen Bits des Klartextes abhängt. Im Idealfall ändert sich bei der Invertierung eines beliebigen Bit des Klartextes jedes Bit des Chiffrats mit einer Wahrscheinlichkeit von $p = 0,5$ so dass auch geringe Änderungen am Klartext sehr grosse Auswirkungen im Chiffrat bewirken. Dies ist erwünscht um Rückschlüsse von ähnlichen Chiffraten auf zueinander ähnliche Klartext-Daten zu verhindern. Für sichere Systeme kann die Verwendung von Konfusion ausreichend sein, für viele Systeme ist die Verwendung von Diffusion jedoch notwendig oder erwünscht.

Da kryptographische Algorithmen in der Regel nur Blöcke fester Länge bearbeiten können, wird die Nachricht x in n Blöcke $x_i, 0 < i \leqq n$ aufgeteilt.

2.4 Stromchiffren

In Stromchiffren wird aus einem, nur den beiden Kommunikationspartnern bekannten, Schlüssel ein Schlüsselstrom beliebiger Länge erzeugt, der mit dem Klartext-Strom verknüpft wird. Üblicherweise wird die XOR-Funktion zur Verknüpfung verwendet, man spricht dann von einer additiven Stromchiffre. Allgemein ist ein Wert des Schlüsselstroms eine Funktion des Schlüssels und des zuvor verschlüsselten Klartextes. Ist der Schlüsselstrom unabhängig vom Klartext spricht man von einer synchronen Stromchiffre. Auf der Empfängerseite wird derselbe Schlüsselstrom erzeugt und durch nochmalige

Verknüpfung mit dem Chiffrat der ursprüngliche Klartext wieder zurückgewonnen. Bei Stromchiffren muss darauf geachtet werden, dass ein Schlüsselstrom niemals wiederverwendet wird. Ansonsten kann folgender Angriff durchgeführt werden [13]:

Die Chiffrierung des Klartextes P_1 durch XOR-Verknüpfung mit dem Schlüsselstrom K ergibt das Chiffrat C_1, die des Klartextes P_2 das Chiffrat C_2. Die XOR-Verknüpfung $C_1 \oplus C_2$ führt dann zu

$$C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = (P_1 \oplus P_2) \oplus (K \oplus K) = P_1 \oplus P_2$$

und somit zur Möglichkeit, den Klartext P_2 durch (gegebenenfalls nur partielle) Kenntnis von P_1 aus dem Chiffrat zu erlangen. Dieses Problem besteht in allen synchronen Stromchiffren, so dass eine Wiederverwendung des Schlüsselstroms unter allen Umständen ausgeschlossen werden muss. Eine Abhilfe bringt hier die Verwendung eines zufällig gewählten und öffentlichen Initialisierungsvektors IV, der mit dem eigentlich Schlüssel verknüpft wird und damit, trotz Beibehaltung des Schlüssels, zu einem anderen Schlüsselstrom führt. Anstelle des Anhängens $K|IV$ oder des Voranstellens $IV|K$ sollte [26] zufolge $H(IV|K)$ verwendet werden um den Schlüssel der Chiffre zu generieren, wobei $H(x)$ eine kryptographische Hashfunktion ist.

Ein Vorteil von Stromchiffren besteht darin, dass synchrone Stromchiffren durch die Unabhängigkeit der einzelnen Bits des Chiffratstroms untereinander fehlerresistent gegenüber fehlerhaft übertragenen Bits sind. Lediglich die fehlerhaften Bits sind im entschlüsselten Klartext ebenfalls fehlerhaft, bei Blockchiffren ist im Allgemeinen immer der gesamte Block unbrauchbar. Allerdings bedeutet bei Stromchiffren der Verlust der Synchronisation zwischen Schlüssel- und Chiffratstrom den vollständigen Verlust der übertragenen Nachricht bis die Synchronisation wiederhergestellt werden kann. Eine synchrone Stromchiffre erzeugt lediglich Konfusion da die einzelnen Zeichen oder Bits des Chiffrats unabhängig voneinander sind. Die Erstellung des Schlüsselstroms ähnelt der Funktionsweise eines Pseudo-Zufallszahlen-Generators.

2.5 Blockchiffren

In Blockchiffren wird der Klartext in Blöcke gleicher Größe unterteilt und diese dann einzeln chiffriert. Blockchiffren verwenden mehrere Runden, in denen eine Verschlüsselungsfunktion zusammen mit dem Rundenschlüssel der entsprechenden Runde auf den Klartext beziehungsweise das Zwischenergebnis der vorangegangenen Runde angewandt wird. Die Rundenschlüssel werden in einem als Key-Schedule oder Key-Setup bezeichneten Prozess aus dem eigentlichen Schlüssel abgeleitet. Alle Blockchiffren verwenden sowohl Konfusion als auch Diffusion. Eine oft verwendete Struktur ist das Feistel-Netz (siehe Abbildung 2.1). In einem Feistel-Netz wird in jeder Runde die

Hälfte des Klartextes oder Zwischenergebnisses mit XOR mit dem Ergebnis einer Rundenfunktion $f(K_r, B)$ der anderen Hälfte verknüpft.
Es gilt: $B(i+1) = A(i)$, $A(i+1) = A(i) \oplus f(K_i, B(i))$.

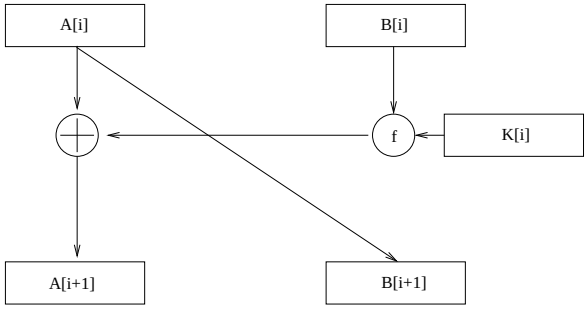

Abbildung 2.1: Struktur der i-ten Runde eines Feistel-Netzes

Im Vergleich mehrerer Blockchiffren wird teilweise der Begriff des Sicherheitsfaktors [24] oder der minimalen sicheren Rundenanzahl [7] verwendet. Beide Begriffe beziehen sich auf die maximale Anzahl von Runden einer Blockchiffre, die gebrochen wurden. Der Sicherheitsfaktor σ nach [24] berechnet sich zu:

$$\sigma = \frac{\text{Anzahl Runden}}{\text{gebrochene Runden}}$$

Ein Wert $\sigma = 1$ steht für eine Chiffre, die in ihrer vollständigen Rundenzahl gebrochen wurde. Eine weitere Eigenschaft von Blockchiffren ist die Existenz schwacher Schlüssel. Ein Schlüssel wird als schwacher Schlüssel bezeichnet, wenn unter seiner Anwendung die Kryptoanalyse vereinfacht wird. In der Regel sind die Klassen schwacher Schlüssel klein genug um sie zu ignorieren aufgrund der geringen Wahrscheinlichkeit bei zufälliger Wahl eines Schlüssels einen schwachen Schlüssel zu wählen. Selbst bei der Blockchiffre IDEA, die Klassen schwacher Schlüssel mit einer maximalen Größe von 2^{64} Mitgliedern aufweist, ist die Wahrscheinlichkeit für die Wahl eines schwachen Schlüssels bei zufälliger Wahl ausreichend gering, um ignoriert zu werden. Viele, aber nicht alle Blockchiffren besitzen Klassen schwacher Schlüssel.

Der Vorteil von Blockchiffren liegt beim Einsatz in ressourcenbeschränkten Umgebungen darin, dass sie zur Generierung eines Message Authentication Codes (siehe Abschnitt 2.7.2) verwendet werden können, wohingegen bei der Verwendung einer Stromchiffre stets ein zusätzlicher kryptographischer Algorithmus erforderlich ist.

Ein Nachteil von Blockchiffren gegenüber Stromchiffren liegt darin, dass in Stromchiffren das Chiffrat stets dieselbe Länge wie der Klartext aufweist. In Blockchiffren beträgt (zumindest ohne zusätzliche Vorkehrungen) die Grö-

ße des Chiffrats immer ein Vielfaches der Blockgröße. Werden überwiegend kleine Nachrichten übertragen, kann dies einen grossen Mehraufwand bedeuten. Ebenfalls problematisch bei Blockchiffren ist die Unabhängigkeit der Blöcke zueinander. Dadurch werden identische Klartext-Blöcke stets auf identische Chiffrat-Blöcke abgebildet, was gegebenenfalls Informationen über den Inhalt der Blöcke enthüllen kann. Um diese Nachteile zu beseitigen ist es möglich, Blockchiffren in unterschiedlichen Modi zu betreiben.

2.5.1 Electronic-Codebook-Mode (ECB-Mode)

Die Ver- und Entschlüsselung einzelner Blöcke unabhängig voneinander wird als Electronic-Codebook-Mode bezeichnet. Die Ver- und Entschlüsselung im ECB-Modus werden wie folgt durchgeführt:

$$c_i = E_K(x_i)$$
$$x_i = D_K(c_i)$$

Vorteil des ECB-Modus ist, dass sich Fehler in der Übertragung eines Blocks nicht weiter fortpflanzen können, da alle Blöcke unabhängig voneinander sind. Die größten Nachteile des ECB-Modus sind wie bereits erwähnt die Abbildung identischer Klartextblöcke einer Nachricht auf jeweils dieselben Chiffratblöcke sowie die Verlängerung des Chiffrats auf das nächste ganzzahlige Vielfache der Blockgröße.

2.5.2 Cipherblock-Chaining-Mode (CBC-Mode)

Um die Nachteile des ECB-Modus zu beheben kann der Cipherblock- Chaining-Modus verwendet werden. Im CBC-Modus wird ein Block x_i des Klartextes durch die Funktion $c_i = E_K(c_{i-1} \oplus x_i)$ chiffriert. Für die Verschlüsselung des ersten Klartextblocks x_1 wird ein Initialisierungsvektor $IV = c_0$ verwendet. Dieser muss nicht geheim sein, sollte aber für jede zu chiffrierende Nachricht unterschiedlich sein, da ansonsten zwei identische Klartext-Nachrichten auf dieselbe verschlüsselte Nachricht abgebildet werden. Außerdem kann es in manchen Situationen sinnvoll sein, unvorhersagbare Initialisierungsvektoren zu verwenden [53]. Die Entschlüsselung wird mit demselben Initialisierungsvektor $c_0 = IV$ wie bei der Verschlüsselung vorgenommen.

$$c_i = E_K(c_{i-1} \oplus x_i)$$
$$x_i = c_{i-1} \oplus D_K(c_i)$$

Durch die Abhängigkeit eines Chiffratblocks von allen vorherigen Blöcken besteht im CBC-Modus die Gefahr durch die Abbildung identischer Klartextblöcke auf identische Chiffratblöcke nicht mehr. Lediglich zwei Nachrichten die mit demselben Schlüssel verschlüsselt wurden und vollständig identisch sind werden auf identische Chiffratblöcke abgebildet. Aber auch

dies lässt sich umgehen, indem für jede Nachricht ein nur einmal verwendeter Initialisierungsvektor verwendet wird.

2.5.3 Cipher-Feedback-Mode (CFB-Mode)

Der CFB-Modus verschlüsselt einen Datenblock durch $c_i = x_i \oplus E_K(c_{i-1})$ wobei wieder ein Initialisierungsvektor $IV = c_0$ benötigt wird. Die Entschlüsselung wird durch die Funktion $x_i = c_i \oplus E_K(c_{i-1})$ ausgeführt.

$$c_i = x_i \oplus E_K(c_{i-1})$$
$$x_i = c_i \oplus E_K(c_{i-1})$$

Im CFB-Modus wird sowohl zur Ver-, als auch zur Entschlüsselung lediglich die Verschlüsselungsfunktion der Blockchiffre verwendet. Dies macht den CFB-Modus für eine Implementierung auf einer ressourcenbeschränkten Plattform attraktiv. Der CFB-Modus wandelt die Blockchiffre in eine asynchrone Stromchiffre um. Damit ist die Größe des Chiffrats unabhängig von der Blockgröße der Blockchiffre. Das Chiffrat eines mit Hilfe einer Blockchiffre im ECB- oder CBC-Modus chiffrierten Klartextes weist immer eine Länge des nächstgrößeren ganzzahligen Vielfachen der Blockgröße auf. Dahingegen ist die Länge eines durch eine Stromchiffre oder einer Blockchiffre im CFB-Modus erzeugten Chiffrats immer mit der des Klartextes identisch. Auch dies stellt bei der Implementierung auf ressourcenbeschränkten Systemen einen Vorteil dar, da sich durch die Verschlüsselung die Menge der zu übertragenden Daten nicht vergrößert. Da in ressourcenbeschränkten Umgebungen die Kommunikationskosten durch den großen Energiebedarf der Funkkommunikation und deren geringe Bandbreite sehr hoch sind, sollte eine Vergrößerung des Bandbreitenbedarfs vermieden werden.

2.5.4 Output-Feedback-Mode (OFB-Mode)

Der Output-Feedback-Mode ähnelt dem Cipher-Feedback-Mode, auch hier wird die Blockchiffre als Schlüsselstromgenerator einer Stromchiffre verwendet. Allerdings wird im OFB-Modus nicht das Chiffrat als Eingabe für die nächste Runde bei der Berechnung des Schlüsselstroms verwendet, sondern der zuvor erzeugte Block des Schlüsselstroms. Somit handelt es sich um eine synchrone Stromchiffre. Wiederum muss der Initialisierungsvektor $IV = S_0$ bekannt sein.

$$S_i = E_K(S_{i-1}); x_i = c_i \oplus S_i$$
$$S_i = E_K(S_{i-1}); c_i = x_i \oplus S_i$$

Vorteil des OFB-Modus gegenüber dem CFB-Modus ist die Unabhängigkeit des Schlüsselstroms vom Klartext und damit die Möglichkeit, den Schlüsselstrom unabhängig von der Verarbeitung des Klartextes zu berechnen. In

einer Implementierung auf einem ressourcenschwachen Mikrocontroller ist dieser Vorteil jedoch nicht von Bedeutung.

2.5.5 Counter-Mode (CTR-Mode)

Eine weitere Variation, die ebenfalls eine synchrone Stromchiffre aus einer Blockchiffre erzeugt, besteht im Counter-Modus (CTR). Der Counter-Modus ähnelt dem OFB-Modus, allerdings wird auf die Rückkopplung des Schlüsselstroms gänzlich verzichtet. Stattdessen wird ein Zähler verwendet, der als Eingabe der Blockchiffre dient.

$$S_i = E_K(n); n = n + 1; x_i = c_i \oplus S_i$$
$$S_i = E_K(n); n = n + 1; c_i = x_i \oplus S_i$$

Der Vorteil des Counter-Modus liegt in der Unabhängigkeit der einzelnen Blöcke voneinander. Zum Einen können damit im Counter-Modus mehrere Blöcke des Schlüsselstroms parallel berechnet werden, womit moderne superskalare CPUs besser ausgelastet sind. Zum Anderen ist es zur Entschlüsselung eines Blocks nicht erforderlich, alle vorhergehenden Blöcke zu entschlüsseln. Bei der Verwendung eines mit dem Wert 1 beginnenden Zählers, der nach jedem Block um den Wert 1 erhöht wird ist der Schlüsselstrom des n-ten Blocks $E_K(n)$. Dies macht den CTR-Modus für Systeme mit wahlfreiem Zugriff, also beispielsweise die Verschlüsselung von Festplatten, sehr geeignet. Die Ver- und Entschlüsselung erfolgt ansonsten wie im OFB-Modus. Da der Schlüsselstrom lediglich vom geheimen Schlüssel und dem Zählerwert abhängig ist muss ausgeschlossen werden, dass derselbe Zählerwert bei einem identischen Schlüssel mehrfach verwendet wird. Dies würde, wie auch die mehrmalige Verwendung des Schlüsselstroms in einer herkömmlichen Stromchiffre, dazu führen, dass aus den mit XOR verknüpften Chiffraten ohne Kenntnis des Schlüssels das Ergebnis der mit XOR verknüpften Klartexte berechnet werden könnte.

2.6 Hashfunktionen

Eine Hashfunktion $h(x)$ überführt eine Nachricht m mit beliebiger Länge auf einen Hashwert $h = h(m)$ mit konstanter Länge. Kryptographische Hashverfahren unterscheiden sich von herkömmlichen Hashverfahren durch höhere Anforderungen an Kollisionsresistenz und Unumkehrbarkeit der Hashfunktion. Erstrebenswert für eine hohe Kollisionsresistenz ist eine Gleichverteilung der Hashwerte bei der Eingabe aller möglichen Nachrichten und somit eine Wahrscheinlichkeit dass eine Kollision der Hashwerte zweier Nachrichten stattfindet von $\frac{1}{2^m}$ bei einer Länge des Hashwertes von m Bit. Zwei weitere wichtige Eigenschaften sind preimage-resistance und secondary-preimage-resistance [20]. Die Eigenschaft preimage-resistance besagt, dass

es nicht möglich (oder unverhältnismäßig schwer) ist, eine Nachricht zu finden, deren Hashwert einem Vorgegebenen entspricht. Secondary-preimage-resistance besagt, dass es nicht möglich ist, eine Nachricht zu erzeugen, die denselben Hashwert hat wie eine vorgegebene Nachricht.

Hashfunktionen teilen üblicherweise die zu bearbeitende Nachricht in Blöcke gleicher Größe auf und wenden iteriert eine Kompressionsfunktion auf den aktuellen Block und den Hashwert des zuvor verarbeiteten Blocks an, um den Hashwert des aktuellen Blocks zu berechnen (Abbildung 2.2). Der initiale Wert zur Berechnung des ersten Blocks ist üblicherweise fest vorgegeben. Die Variablen in denen die Zwischenergebnisse gespeichert werden werden als Verkettungsvariablen bezeichnet. Vor Anwendung der Kompressionsfunktion findet üblicherweise ein Padding der Eingabedaten statt, um die Nachricht auf ein Vielfaches der Blockgröße zu verlängern. Oftmals sind Hashverfahren nur bei Nachrichten einer fest vorgegebenen Länge kollisionsresistent. Daher wird in der Regel die Länge der Nachricht an den letzten Block der Nachricht angehängt, um den Bruch des Verfahrens durch Konstruktion einer Nachricht mit einer beliebigen Länge zu verhindern [53].

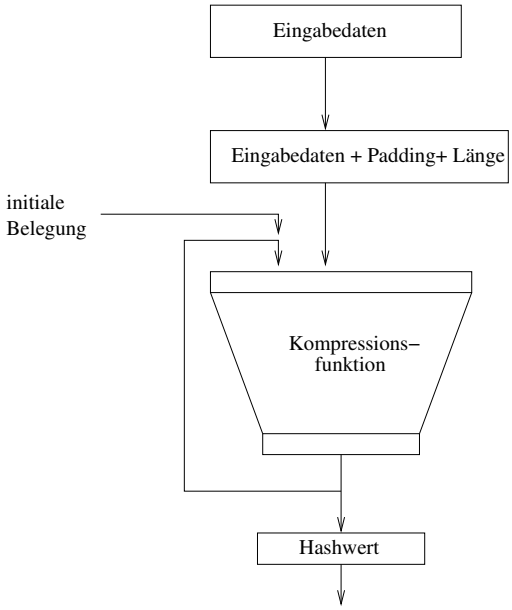

Abbildung 2.2: Aufbau eines Hashing Algorithmus

2.7 Message Authentication Code (MAC)

Die Verwendung eines Hashverfahrens schützt nur vor Veränderungen aufgrund von Übertragungsfehlern, nicht jedoch vor böswilligen Veränderungen

der übertragenen Daten durch einen Angreifer. Da dieser den Hashwert einer von ihm manipulierten Nachricht neu berechnen und übermitteln kann, trifft ein korrekter Hashwert lediglich eine Aussage über die Integrität der Nachricht, nicht aber über die Authentizität. Dieses Problem kann durch die Verwendung von Message Authentication Codes behoben werden. Bei MACs handelt es sich um integritätssichernde Verfahren, die zusätzlich einen geheimen, nur den beiden Kommunikationspartnern bekannten, Schlüssel verwenden um damit neben der Integrität auch die Authentizität sicherzustellen. Der überwiegende Teil der MAC-Algorithmen lässt sich in die Klassen der auf Hashverfahren basierenden und der auf CBC-Blockchiffren basierenden Verfahren aufteilen.

2.7.1 MACs auf Basis von Hashverfahren

Die erste Möglichkeit, MACs zu erzeugen, verwendet ein beliebiges kryptographisches Hashverfahren und verknüpft den geheimen Schlüssel mit der Nachricht beispielsweise durch Voranstellen. Auch kompliziertere Verknüpfungen von Schlüssel und Nachricht sind denkbar. Ein Angreifer kann nicht den korrekten MAC zu einer von ihm gefälschten Nachricht berechnen, da er nicht über den Schlüssel verfügt. Ein Beispiel eines solchen Verfahrens ist das HMAC Verfahren, das in RFC-2104 [48] standardisiert ist. Es berechnet den MAC einer Nachricht M als $h(K \oplus \text{opad}|h(K \oplus \text{ipad}|M)$ berechnet. Die Konstanten opad und ipad werden verwendet, um den Schlüssel auf die für das Hashverfahren benötigte Blockgröße zu verlängern.

2.7.2 MACs auf CBC-Basis

Eine Blockchiffre im CBC-Modus kann zur Erstellung eines MAC herangezogen werden. Dazu wird die Nachricht mit der Blockchiffre und dem gemeinsamen Schlüssel chiffriert. Der chiffrierte letzte Block der Nachricht wird als MAC zusätzlich zur Nachricht übertragen (siehe Abbildung 2.3). Ein Angreifer kann keinen gültigen MAC einer von ihm veränderten Nachricht erzeugen, da ihm der dazu notwendige Schlüssel nicht bekannt ist. Aufgrund der verwendeten Blockchiffre ist es erforderlich, ein Padding-Verfahren zu verwenden, um Nachrichten mit einer Länge die kein Vielfaches der Blockgröße ist verarbeiten zu können. Das CBC-MAC-Verfahren ist lediglich bei Nachrichten fester und vorgegebener Größe sicher [6]. Soll ein CBC-MAC bei Nachrichten beliebiger Länge verwendet werden, ist es erforderlich das Verfahren zu modifizieren. Eine solche Modifikation ist das OMAC-Verfahren [38], das durch die Verknüpfung des letzten Klartextblocks mit einer Konstanten sicher stellt, dass es einem Angreifer nicht möglich ist, durch Verlängerung oder Verkürzung der Nachricht eine gefälschte Nachricht mit einem korrekten MAC zu erzeugen (siehe Abbildung 2.4). Falls die Länge der Nachricht kein ganzzahliges Vielfaches der Blockgröße der verwendeten Chiffre

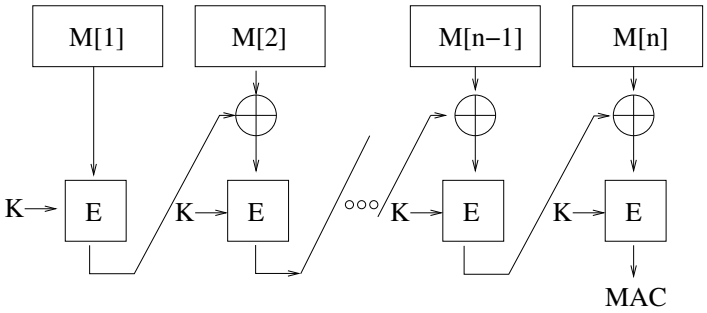

Abbildung 2.3: CBC-MAC Verfahren

darstellt, muss im letzten Block ebenfalls ein Padding auf die Blockgröße durchgeführt werden.

OMAC verwendet bei Nachrichten, deren Länge ein ganzzahliges Vielfaches der Blockgröße ist die Konstante $L * u$ für die XOR Operation des letzten Blocks. Dabei ist $L = E_K(0^b)$ für die Blockgröße b (in Bit) und u eine beliebige öffentliche Konstante $u \in GF[2^b]$. Bei Nachrichten deren Länge kein ganzzahliges Vielfaches der Blockgröße darstellt, wird statt dessen die Konstante $L * u^{-1}$ (OMAC2) oder $L * u^2$ (OMAC1) verwendet.

Vorteil von auf CBC-MAC basierenden Verfahrens gegenüber solchen Verfahren, die auf Hashverfahren basieren, ist die Möglichkeit, die Blockchiffre, die zur Verschlüsselung der Daten dient auch zur Generierung des MAC heranzuziehen. Dadurch kann ein Algorithmus eingespart werden, was das OMAC-Verfahren insbesondere für ressourcenbeschränkte Systeme attraktiv macht.

2.8 Kryptographische Systeme

Die Sicherheit kryptographischer Systeme wird maßgeblich von der korrekten Verknüpfung der kryptographischen Grundbausteine wie symmetrischen Blockchiffren oder MAC-Algorithmen zu sicheren Systemen beeinflusst. Während als kryptographische Grundbausteine üblicherweise Standardverfahren verwendet werden, deren korrekte Implementierung sich mit Hilfe von Testvektoren verifizieren lässt, ist die Verknüpfung der Primitiven deutlich fehleranfälliger, da Abhängigkeiten zwischen den Bausteinen leicht zu Sicherheitslücken führen können. Wie bei den kryptographischen Primitiven selbst ist es so auch bei den sicheren Systemen und Protokollen erstrebenswert, auf bewährte Verfahren zurückzugreifen, deren Sicherheit validiert wurde.

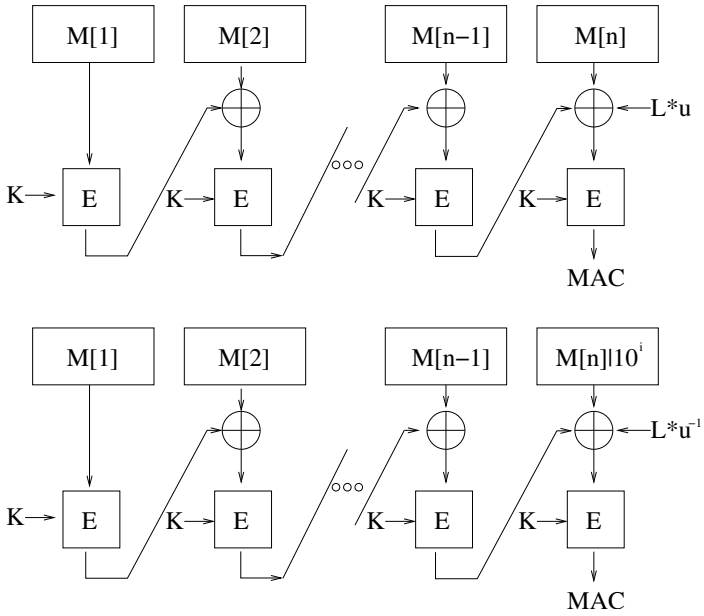

Abbildung 2.4: OMAC Verfahren

Methode	Operationen
authenticate-then-encrypt	$a = Auth(x)$, $C = Enc(x\|a)$, $transmit(C)$
encrypt-and-authenticate	$C = Enc(x)$, $a = Auth(x)$, $transmit(C\|a)$
encrypt-then-authenticate	$C = Enc(x)$, $a = Auth(C)$, $transmit(C\|a)$

Tabelle 2.1: Mögliche Reihenfolgen für Chiffrierung und Authentifikation

2.8.1 AE-Kanäle

Die wichtigste Anwendung kryptographischer Verfahren liegt in der Schaffung eines sicheren Kommunikationskanals. Ein sicherer Kanal der die im Abschnitt 2.1 genannten Ziele erfüllt, erfordert neben der Verschlüsselung zur Wahrung der Geheimhaltung zumindest eine Sicherstellung der Nachrichtenintegrität und -Authentizität, um eine Manipulation der übertragenen Nachricht durch Angreifer zu verhindern. Die Reihenfolge der Operationen Verschlüsselung und Integritätsicherung ist dabei von Bedeutung. Die drei möglichen Reihenfolgen, Verschlüsselung und Authentizitätssicherung durchzuführen, sind in Tabelle 2.1 dargestellt. In [49] beschreibt Krawczyk die Möglichkeit, in einem System, das zuerst mit einem MAC die Nachrichtenintegrität und -authentizität sicherstellt und danach die Nachricht und den MAC der Nachricht verschlüsselt (authenticate-then-encrypt), Kenntnisse über den Klartext zu erlangen. Dies kann auch bei kryptographisch sicheren und korrekt implementierten Chiffrier- und MAC-Algorithmen möglich sein. Gleiches gilt für die Verwendung eines MAC, der aus dem Klartext

gewonnen wird und der unverschlüsselt übertragen wird. Krawczyk zufolge ist lediglich die Reihenfolge encrypt-then-authenticate bei jedem sicheren MAC-Verfahren und jedem sicheren Chiffrieralgorithmus sicher. Die sichere Übertragung einer Nachricht muss also wie folgt durchgeführt werden:

1. Wenn erforderlich: Generierung eines Initialisierungsvektors IV
2. Verschlüsselung der Nachricht M zu $C = E_K(M, IV)$ unter Verwendung des symmetrischen Algorithmus E und des Sitzungsschlüssels K.
3. Berechnung des Message Authentication Code $T = MAC(IV|C)$.
4. Übermittlung des Tupel (IV,C,T)

Die Sicherheit eines authenticate-then-encrypt Systems ist Krawczyk zufolge jedoch gewährleistet, wenn die Blockchiffre E im CBC-Modus betrieben wird oder wenn E eine Stromchiffre ist.

Neben der generischen Komposition eines AE-Systems sind auch im Detail analysierte Implementierungen bekannt. Der Vorteil solcher Konstruktionen gegenüber generischen Verfahren liegt darin, dass das Risiko, durch schlechte Wahl der kryptographischen Algorithmen die Sicherheit des Systems zu vermindern, nicht besteht [46]. Da die Abhängigkeiten zwischen den kryptographischen Algorithmen analysiert werden können, ist es zudem für viele dieser Verfahren zulässig, für Chiffrierung und Authentifikation identische Schlüssel zu verwenden. Ein Beispiel eines solchen Schemas ist der OCB-Modus für Blockchiffre der zur Sicherstellung der Vertraulichkeit und Authentizität dieselbe Blockchiffre verwendet [61]. OCB benötigt lediglich einen Schlüssel und erfordert zudem für jede Nachricht eine NOnce, die nur einmal verwendet werden darf, aber weder zufällig noch unvorhersehbar sein muss.

2.8.2 AEAD-Kanäle

Die bereits in Abschnitt 2.2 angesprochenen AEAD-Systeme stellen neben der Vertraulichkeit, Integrität und Authentizität der Daten zusätzlich die Integrität und Authentizität weiterer unverschlüsselter Daten sicher. Vertreter von AEAD-Systeme sind EAX [5] und CCM [18]. EAX verwendet als Verschlüsselungsalgorithmus eine beliebige Blockchiffre im Counter-Mode. Die Integrität und Authentizität wird mit Hilfe des OMAC-Algorithmus unter Verwendung derselben Blockchiffre sichergestellt. CCM verwendet ebenfalls eine Blockchiffre (üblicherweise AES) im Counter-Mode, um die Nachricht zu verschlüsseln und dieselbe Blockchiffre im CBC-Mode, um die Integrität und Authentizität sicher zu stellen.

Die grundlegenden Blöcke, die in EAX verwendet werden, sind auf Seite 33 aufgezählt. Dabei ist $M = M_1M_2...M_m$ die in m Blöcke der Länge n (in Bit) aufgeteilte Nachricht und $n' = |M| \bmod n$ die Länge (ebenfalls in

Bit) des letzten Blocks. Abbildung 2.5 zeigt die Konstruktion des AEAD-Systems aus diesen Blöcken. Wieder ist M die Nachricht, die verschlüsselt und mit gesicherter Authentizität übermittelt wird. H ist der Protokollkopf, der nicht verschlüsselt übertragen wird, dessen Authentizität jedoch sichergestellt sein soll. N ist eine Zufallszahl (NOnce) die während der gesamten Nutzungsdauer des Schlüssels K nur einmal verwendet werden darf und als Initialisierungswerk der Blockchiffre im CTR-Modus dient. Die Größen $\tau, |H|$ und $|N|$ müssen beim Aufbau der Kommunikation ausgehandelt werden. Das Verfahren erzeugt das Chiffrat C der Nachricht M, sowie den MAC T, der über Nachricht M, nonce N und Protokollkopf H gebildet wird. Der Algorithmus, der die vorab definierten Funktionen verwendet, kann somit wie folgt angegeben werden:

$T_N \leftarrow OMAC_K^0(N)$
$T_H \leftarrow OMAC_K^1(H)$
$C \leftarrow CTR_K^{T_N}(M)$
$T_C \leftarrow OMAC_K^2(C)$
$T \leftarrow$ die ersten τ Bit von $T_N \oplus T_C \oplus T_H$
sende$(P = H||N||C||T)$

Die Entschlüsselung und Überprüfung der Authentizität der übermittelten Nachricht wird auf Empfängerseite wie folgt durchgeführt:

if $|P| < \tau + |H| + |N|$ **then**
  return INVALID
end if
$(H||N||C||T) \leftarrow P$
$T_N \leftarrow OMAC_K^0(N)$
$T_H \leftarrow OMAC_K^1(H)$
$T_C \leftarrow OMAC_K^2(C)$
$T' \leftarrow$ die ersten τ Bit von $T_N \oplus T_C \oplus T_H$
if $T' \neq T$ **then**
  return INVALID
else
  $M \leftarrow CTR_K^{T_N}(C)$
  return M
end if

Für EAX wird lediglich ein Schlüssel benötigt, der sowohl für die Blockchiffre im CTR-Modus zur Chiffrierung der Nachricht, als auch für den OMAC Algorithmus verwendet wird. EAX verwendet $2\lceil\frac{|M|}{n}\rceil + \lceil\frac{|H|}{n}\rceil + \lceil\frac{|N|}{n}\rceil$ Aufrufe der Blockchiffre. Zur Initialisierung werden drei weitere Aufrufe der Blockchiffre benötigt, um die Verschlüsselung $L = E_K(0^n)$ (vergleiche Abschnitt 2.7.2) und T_N aus der NOnce N zu berechnen. Der Bandbreiten-Mehrbedarf beschränkt sich auf $\tau + |N|$ Bits zur Übermittlung des MAC und der nonce.

$CBC_K(M)$:

$C_0 \leftarrow 0^n$
for $i = 1$ to m **do**
  $C_i = E_K(M_i \oplus C_{i-1})$
end for
return C_m

$CTR_K^N(M)$

$S \leftarrow E_K(N)||E_K(N+1)||...||E_K(N+m-1)$
$C \leftarrow M \oplus S$
return C

$pad(M, B, P)$

if $n' = 0$ **then**
  return $M_1||M_2||...||M_{m-1}||M_m \oplus B$
else
  return $(M_1||M_2||...||M_{m-1}||(M_m||10^{n-1-n'}) \oplus P)$
end if

$OMAC_K(M)$

$L \leftarrow E_K(0^n); B \leftarrow 2L; P \leftarrow 4L$
return $CBC_K(pad(M, B, P))$

$OMAC_K^t(M)$

return $OMAC_K([t]_n||M)$

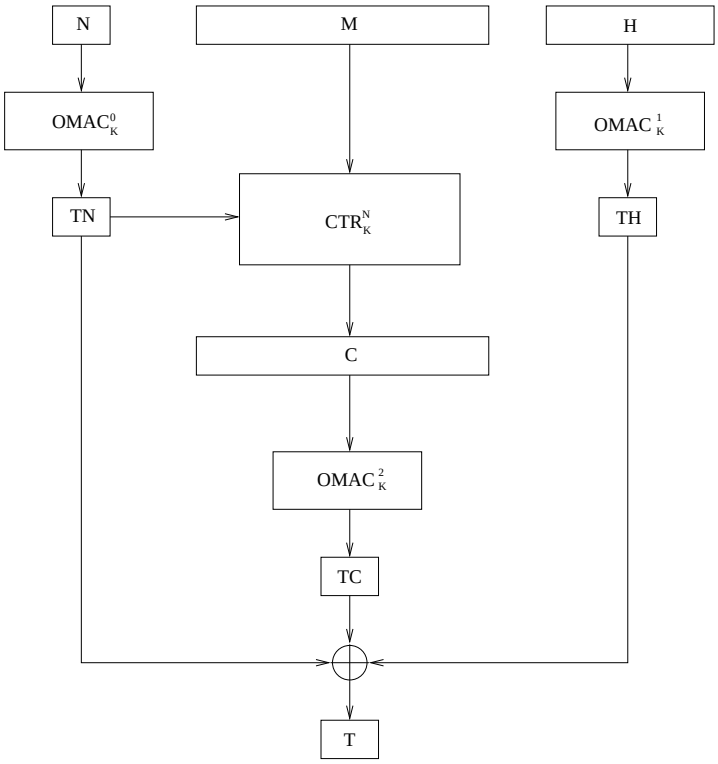

Abbildung 2.5: EAX Verfahren [5]

Die bisher vorgestellten Verfahren ermöglichen eine vertrauliche Kommunikation deren Integrität und Authentizität sicher gestellt wird. Zusätzlich werden Verfahren benötigt, um auf sichere Weise Schlüssel zwischen den Kommunikationspartnern zu vereinbaren. Dies kann entweder über einen sicheren Kanal wie eine direkte elektronische Verbindung die nicht abgehört werden kann geschehen oder aber mit Hilfe asymmetrischer Kryptographie.

2.9 Asymmetrische Verfahren

Von asymmetrischen Kryptoverfahren (auch: Public Key Verfahren) spricht man, wenn für Ver- und Entschlüsselung unterschiedliche Schlüssel zum Einsatz kommen, die als öffentlicher sowie privater oder geheimer Schlüssel bezeichnet werden. Für die einander entgegengesetzten Operationen (beispielsweise Ver- und Entschüsselung) wird für eine Operation der öffentliche Schlüssel, für die Umkehrung der private Schlüssel benötigt. Es besteht keine Notwendigkeit, den öffentlichen Schlüssel geheim zu halten, er kann in öffentlichen Verzeichnissen bereitgestellt werden. Eine Nachricht entschlüsseln kann nur, wer den privaten Schlüssel des Schlüsselpaars aus öffentli-

chem und privatem Schlüssel kennt. Asymmetrische Verfahren sind sehr viel aufwändiger und langsamer als symmetrische Verfahren. Daher werden sie üblicherweise nur dazu verwendet, einen zufällig generierten symmetrischen Sitzungsschlüssel zwischen zwei Kommunikationspartnern sicher auszutauschen. Die eigentliche Kommunikation wird dann mit einem symmetrischen Verfahren verschlüsselt. Man spricht in diesem Fall von Hybridsystemen.

Asymmetrische Verfahren lösen das Problem der Schlüsselverteilung, das bei symmetrischen Verfahren zu Schwierigkeiten führt, falls viele Teilnehmer miteinander kommunizieren wollen, die Kommunikation aber vor Allen außer dem aktuellen Kommunikationspartner geheim gehalten werden soll. Um dies zu bewerkstelligen sind in einem symmetrischen System $O(n^2)$ Schlüssel bei n Teilnehmern notwendig. Asymmetrische Verfahren kommen dagegen mit n Schlüsseln aus, da alle Teilnehmer lediglich den öffentlichen Schlüssel des Kommunikationspartners benötigen, um vertraulich mit ihm kommunizieren zu können.

Bei der Verwendung einer hierarchischen Public-Key-Infrastruktur mit Zertifikaten und Zertifizierungsstellen (siehe Abschnitt 2.9.2) kann die Anzahl der bekannten Schlüssel weiter verringert werden.

2.9.1 Digitale Signaturen

Digitale Signaturen sind das Äquivalent zu einer eigenhändigen Unterschrift im traditionellen Schriftverkehr. Sie basieren auf der Public-Key Kryptographie und können nur vom Inhaber des geheimen Schlüssels eines Schlüsselpaars geleistet werden. Üblicherweise wird aus dem zu signierenden Dokument eine Prüfsumme mittels eines Hashverfahrens (siehe Abschnitt 2.6) errechnet und diese durch Verschlüsselung mit dem geheimen Schlüssel als Signatur des Dokuments verwendet. Dadurch kann die Gültigkeit einer Signatur durch jeden überprüft werden, der den öffentlichen Schlüssel des Unterzeichnenden kennt. Ein häufig verwendetes Signaturschema ist der Digital Signature Standard (DSS) [56] der den Digital Signature Algorithmus (DSA) [56] oder den RSA Algorithmus [2] verwendet.

2.9.2 Zertifikate und Public-Key Infrastrukturen

Digitale Signaturen stellen sicher, dass eine Nachricht vom signierenden Partner erstellt wurde und dass die Nachricht nicht verändert wurde. Allerdings wird zur Überprüfung der Signatur der öffentliche Schlüssel des Kommunikationspartners benötigt. Dieser muss über einen sicheren Weg ausgetauscht werden, damit nicht ein Angreifer seinen öffentlichen Schlüssel als den eines Anderen ausgeben kann. Gleichzeitig ist es aber – zumindest in größeren Public-Key Infrastrukturen – keine gangbare Lösung, allen Teilnehmern alle Schlüssel auf sicherem Wege zu übermitteln. Dies würde vor allem die Integration neuer Teilnehmer sehr aufwändig machen. Die übli-

che Methode, dieses Problem zu lösen, besteht in der Errichtung von Zertifizierungsstellen (CA, engl. Certification Authority) die die Identität der Teilnehmer garantieren. Dazu werden alle verwendeten Schlüsselpaare durch die CA generiert. Der öffentliche Schlüssel wird zusammen mit weiteren Angaben wie dem Namen des Schlüsselinhabers und dem Gültigkeitszeitraum des Schlüssels in einem Zertifikat zusammengefasst. Dieses wird mit dem geheimen Schlüssel der CA digital signiert. Somit ist es jedem Teilnehmer, der den öffentlichen Schlüssel der CA kennt, möglich, ein von einem anderen Teilnehmer übermitteltes Zertifikat auf seine Gültigkeit hin zu überprüfen. Auf diese Weise kann ein Teilnehmer immer sicher sein, dass ein öffentlicher Schlüssel zu dem Teilnehmer gehört, der im Zertifikat identifiziert wird. In einem System mit CA reduziert sich die Vorbedingung für eine zuverlässige Identifikation eines Teilnehmers auf eine sichere Übermittlung des öffentlichen Schlüssels der CA an alle Teilnehmer. Dieser kann aber bereits bei Auslieferung des Systems fest hinterlegt werden. Zusätzlich wird eine Liste zurückgezogener Zertifikate (CRL, engl. Certificate Revocation List) geführt und verteilt. In diese werden Zertifikate eingetragen, die vor ihrem eigentlichen Verfallsdatum als ungültig gekennzeichnet werden sollen. Dies ist beispielsweise dann erforderlich, wenn der zugehörige private Schlüssel kompromittiert wurde oder wenn der Inhaber des Zertifikats aus der Firma ausgeschieden ist. Vor Allem im ersten Fall, also der Kompromittierung des privaten Schlüssels des zum Zertifikat gehörenden Schlüsselpaars, ist eine zuverlässige Verteilung der CRL an alle Teilnehmer von hoher Bedeutung für die Sicherheit des Systems.

2.10 Schlüsselgenerierung

Oftmals ist es erstrebenswert, bei der Verwendung eines einzelnen gemeinsamen Geheimnisses mehrere symmetrische Schlüssel zu erzeugen, beispielsweise um bei einem System mit Verschlüsselung und MAC separate Schlüssel für diese beiden Funktionen zu verwenden. Ansonsten kann die Gefahr bestehen, dass ein Angriff auf den schwächeren der beiden Algorithmen auch den Schüssel des eigentlich stärkeren Verfahrens offenlegt. Zudem ist die Auswirkung, die eine Verwendung desselben Schlüssels in zwei unterschiedlichen Algorithmen auf die Sicherheit des Gesamtsystems hat, kaum abzuschätzen und sollte daher vermieden werden. Ein weiterer Grund könnte auch die Erzeugung von Sitzungsschlüsseln sein, um in jeder Sitzung die Daten mit einem anderen Schlüssel zu chiffrieren. Da viele Algorithmen anfällig gegenüber zueinander ähnlichen Schlüsseln (related keys) sind [41], sollen die aus einem gemeinsamen Geheimnis gewonnenen Schlüssel möglichst keine Rückschlüsse aufeinander zulassen und in keinem einfachen Zusammenhang zueinander stehen. Eine Möglichkeit, dies zu erreichen liegt in der Verwendung eines symmetrischen Chiffrierverfahrens oder eines kryptogra-

phischen Hashverfahrens in der Schlüsselgenerierung. Beispielsweise kann bei der Verwendung eines gemeinsamen Geheimnisses K und der Blockchiffre E der Schlüssel des Chiffrieralgorithmus zu $K_C = E_K(0)$ und der des MAC-Algorithmus zu $K_M = E_K(1)$ bestimmt werden. Dies ist möglich, da jede gute Blockchiffre resistent gegen Known-Plaintext Angriffe ist. Die Erzeugung von Sitzungsschlüsseln ist möglich, indem eine Zufallszahl z erzeugt wird und unverschlüsselt zum Kommunikationspartner übertragen wird. Beide Kommunikationspartner erzeugen den Sitzungsschlüssel dann als $K_C = E_K(z)$. Dieses Verfahren wird unter anderem in der Kryptographieschicht des Mobilkommunikationssystems GSM angewandt.

Eine weitere erstrebenswerte Eigenschaft kryptographischer Systeme ist die forward security (auch: forward secrecy). Ein System besitzt forward security, wenn bei Kompromittierung eines Sitzungsschlüssels keine älteren Sitzungen kompromittiert werden. Ein System, das für alle Sitzungen unterschiedliche Schlüssel verwendet, die auf sichere Weise von einem Hauptschlüssel abgeleitet werden, besitzt forward security. Ein System besitzt weiter die Eigenschaft perfect forward security wenn auch bei Offenlegung des Hauptschlüssels die in der Vergangenheit verwendeten Sitzungsschlüssel nicht kompromittiert sind. Perfect forward secrecy kann beispielsweise mit Hilfe eines Diffie Hellman Schlüsseltauschs erreicht werden [53].

Während in leistungsstärkeren Umgebungen die Schlüsselverteilung mit Hilfe asymmetrischer Kryptographie erfolgen kann, ist das wie in Abschnitt 5.10 ausgeführt wird in ressourcenbeschränkten Umgebungen nur schwer möglich. Daher müssen alternative Möglichkeiten zur Übermittlung eines gemeinsamen Geheimnisses verwendet werden. Eine Möglichkeit, die in [70] genannt wird, ist der physikalische Kontakt zweier Geräte. Da in Umgebungen wie der Heimautomatisierung oftmals ohnehin eine Zuordnung zwischen Sensoren oder Aktoren und deren Steuergeräten getroffen werden muss, beispielsweise um festzulegen welche Fernbedienung welche Tür öffnet, liegt es nahe, diese Zuordnung durch eine einmalige elektronische Verbindung zweier Geräte festzulegen. Da die direkte Verbindung zweier Geräte als sicherer Kanal verwendet werden kann, kann in diesem Zusammenhang ein Paar-Schlüssel zwischen den beiden Geräten ausgetauscht werden, aus dem in späteren Kommunikationssitzungen Sitzungsschlüssel gewonnen werden können.

2.11 Zufallszahlen

Die Bedeutung von Zufallszahlen bei der Implementierung kryptographischer Systeme darf nicht unterschätzt werden. Zufallszahlen werden vor allem bei der Generierung symmetrischer Sitzungsschlüssel oder im Digital Signature Algorithm, sowie als Initialisierungsvektor benötigt. Gute Zufallszahlen zeichnen sich dadurch aus, dass sie statistisch zufällig erscheinen und

nicht deterministisch voneinander hergeleitet werden können. Die Verwendung von Pseudozufallszahlen, die, ausgehend von einem zufälligen Wert, eine Folge von Zahlen erzeugen, die nicht offensichtlich deterministisch erscheint, ist nicht für alle Anwendungen zulässig. Während für den Digital Signature Algorithm (DSA) eine Zahl erforderlich ist, die zufällig sein muss, so ist im DSA-Algorithmus auf elliptischen Kurven (EC-DSA) eine Zahl erforderlich, die, unter Verwendung desselben Schlüssels, nie wiederverwendet werden darf und die unvorhersagbar sein muss [39]. Diese Anforderungen können auf Embedded Systemen zu Schwierigkeiten führen, da dies entweder Zufallszahlen mit einem sehr grossen Wertebereich, so dass die Wahrscheinlichkeit der Wiederverwendung einer Zufallszahl ausreichend gering ist oder aber eine Speicherung bereits verwendeter Zufallszahlen erfordert. Beide Bedingungen lassen sich mit beschränkten Ressourcen nur schwer realisieren. Eine Pseudozufallsfolge wie sie ein Linearer Kongruenzgenerator $r_i = (a * r_{i-1} + c) \bmod M$ liefert, hat innerhalb dessen Periode stets unterschiedliche Pseudozufallszahlen. Wenn ein Wert r_i sowie die Parameter a und c bekannt sind kann ein Angreifer jedoch die folgende Zufallszahl r_{i+1} berechnen. Diese Methode, Zufallszahlen zu berechnen, ist daher in kryptographischen Anwendungen meist nicht zulässig. Eine bessere Möglichkeit, (Pseudo-)Zufallszahlen zu erzeugen, liegt in den kryptographischen Algorithmen selbst. Die Ausgabe „guter“ kryptographischer Algorithmen ist von Zufallszahlen nicht zu unterscheiden. Ausgehend von einem zufälligen Startwert können Zahlen erzeugt werden, die statistischen Tests auf Zufälligkeit genügen und aufgrund des grossen Wertebereichs der kryptographischen Funktion auch statistisch einzigartig sind. Die zuvor verwendete Pseudo-Zufallszahl kann als Klartext zusammen mit einem geheimen Schlüssel mit einer Blockchiffre verschlüsselt und das Chiffrat als neue Pseudo-Zufallszahl verwendet werden. In einem Algorithmus wie dem One-Time-Pad, der echte Zufallszahlen erfordert, ist aber auch diese Art, Zufallszahlen zu generieren, nicht zulässig.

2.12 Angriffe auf kryptographische Algorithmen

Gegen die in Abschnitt 2.1 definierten Sicherheitsziele können verschiedene Angriffe unternommen werden. Um diese klassifizieren zu können ist es zunächst erforderlich, ein Angreifermodell festzulegen:

- **Passiver Mitschnitt/Known Chiphertext:** Das schwächste Angreifermodell verfügt lediglich über die Möglichkeit, die verschlüsselte Kommunikation zu belauschen.

- **Known Plaintext:** Von Known-Plaintext Angriffen auf kryptographische Systeme spricht man, wenn dem Angreifer das Chiffrat und der dazugehörige Klartext (oder Teile davon) bekannt sind. Ein gutes

Kryptosystem muss gewährleisten, dass es dem Angreifer auch unter diesen Umständen nicht möglich ist, den Schlüssel des Systems zu bestimmen oder weitere Chiffrat-Teile deren Klartext er nicht hat, zu entschlüsseln.

- **Chosen Plaintext:** Um einen Chosen-Plaintext Angriff durchführen zu können, muss ein Angreifer über die Möglichkeit verfügen, von ihm gewählte Klartexte verschlüsseln zu lassen. Auch in diesem Fall darf es nicht möglich sein, aus den Paaren von Klartext und Chiffrat den Schlüssel zu errechnen oder andere chiffrierte Daten zu entschlüsseln.

- **Adaptive Chosen Plaintext:** Bei einem Adaptive-Chosen-Plaintext Angriff ist es einem Angreifer möglich, beliebig viele Chosen-Plaintext Angriffe durchzuführen. Dies wird verwendet, um die Auswirkungen von Änderungen im Klartext auf das Chiffrat zu untersuchen.

Abhängig vom Angreifermodell können unterschiedliche Angriffe auf die kryptographischen Algorithmen unternommen werden, um entweder den Schlüssel oder den Klartext aus dem Chiffrat zu errechnen.

- **Brute Force:** Von Brute-Force Angriffen spricht man, wenn ein Verfahren gebrochen werden soll, indem alle möglichen Schlüssel geprüft werden. Ein Brute-Force Angriff auf einen Algorithmus mit einem 2^{128} Bit langen Schlüssel erfordert im Durchschnitt 2^{127} Tests. In guten kryptographischen Algorithmen ist der Brute-Force Angriff der einzig Mögliche, andernfalls wird der Algorithmus als gebrochen bezeichnet. Ein Brute-Force Angriff kann bei allen Angreifermodellen unternommen werden. Allerdings ist ein reiner brute-force Angriff im Allgemeinen nicht möglich. Damit festgestellt werden kann, ob ein gewählter Schlüssel korrekt ist, muss verifiziert werden können, ob der bei der Entschlüsselung mit diesem Schlüssel gewonnene Klartext korrekt ist. Dazu ist eine Kenntnis von Teilen des Klartextes erforderlich oder zumindest ein probabilistisches Verfahren, das entscheiden kann, ob der Klartext gültig sein kann.

- **lineare Kryptoanalyse:** Lineare Kryptoanalyse ist ein Know-Plaintext Angriff, bei dem versucht wird, aus mehreren Paaren aus Klartext und Chiffrat Beziehungen zwischen Bits des Klartextes und des Chiffrats zu finden, so dass die Chiffrat-Bits bei Änderungen im Klartext nicht mit der erwünschten Wahrscheinlichkeit von $p = 0,5$ den Wert 0 oder 1 annehmen, sondern eine Tendenz zu einem dieser Werte aufweisen. Es wird versucht, über diese Tendenz Informationen über weitere Bits des Klartextes oder des Schlüssels zu gewinnen. Lineare Kryptoanalyse wurde erstmals von Matsui und Yamagishi eingesetzt, um die FEAL-Chiffre zu analysieren [51].

- **differentielle Kryptoanalyse:** Differentielle Kryptoanalyse versucht aufgrund der Änderungen im Chiffrat zweier bekannter und zueinander ähnlicher Klartexte, Kenntnisse über den Schlüssel der Chiffre zu erlangen. Somit handelt es sich in der Regel um einen Adaptive-Chosen-Plaintext Angriff, bei dem eine oder mehrere Nachrichten solange verändert und chiffriert werden bis ausreichend Informationen über den Schlüssel hergeleitet werden konnten. Erstmal wurde differentielle Kryptoanalyse von Murphy [55] zur Analyse von FEAL-4 angewandt.

- **related Key Angriffe:** Falls in einem Kryptosystem mehrere Schlüssel verwendet werden, die in einem (einfachen) Zusammenhang zueinander stehen, kann es dadurch zu einer Herabsetzung der Sicherheit kommen [9]. Daher sollte stets darauf geachtet werden, dass Schlüssel in keinem oder zumindest in keinem einfachen Zusammenhang zueinander stehen (siehe Abschnitt 2.10).

Bei Angriffen auf Hashfunktionen wird entweder versucht, zu einer gegebenen Nachricht eine neue Nachricht zu finden, die denselben Hashwert besitzt oder es wird versucht, eine Nachricht zu erzeugen, die einen bestimmten vorgegebenen Hashwert hat. Die obere Grenze für den dafür erforderlichen Aufwand ist der Wertebereich des Hashwertes. Ist es möglich, mit einem geringeren Aufwand Kollisionen zu errechnen, sollte die Hashfunktion nicht mehr in Umgebungen eingesetzt werden, die eine hohe Kollisionsresistenz verlangen. Der sogenannte Geburtstags-Angriff (Birthday Attack) nutzt die Tatsache aus, dass die Wahrscheinlichkeit einer Kollision einer Hashfunktion quadratisch in der Anzahl der Nachrichten steigt. Während die Wahrscheinlichkeit einer Kollision eines Hashwertes mit einem fest gewählten anderen Hashwert im Idealfall $\frac{1}{2^b}$ bei Verwendung eines b Bit grossen Hashwertes beträgt, beträgt die Wahrscheinlichkeit, dass die Hashwerte zweier beliebiger Nachrichten kollidieren lediglich $\frac{1}{2^{0.5*b}}$. Falls also eine Kollision zwischen zwei beliebigen Nachrichten zu einem Sicherheitsproblem führt, muss der Wertebereicht des Hashwertes vergrössert werden.

Insbesondere in hardwarenahen Umgebungen muss zusätzlich die Gefahr durch Seitenkanal-Angriffe betrachtet werden. Seitenkanal-Angriffe versuchen ein Kryptosystem durch Beobachtungen des Energieverbrauchs, elektromagnetischer Abstrahlungen oder der Laufzeit des Verfahrens zu brechen. Falls die Laufzeit des Algorithmus abhängig vom Schlüssel oder den Klartextdaten ist, kann dieses Wissen dazu genutzt werden, Kenntnisse über ansonsten geheime Daten zu erlangen. Außerdem fallen unter den Begriff des Seitenkanal-Angriffs jegliche Art physikalischer Angriffe, wie beispielsweise der Mitschnitt der Kommunikation der CPU-Datenbusse oder das Auslesen von EEPROM-Datenzellen durch Messung ihrer Spannung. Die Gefahr durch Seitenkanal-Angriffe lässt sich durch eine Kapselung der CPU erschweren, sowie durch Maßnahmen, die eine Zerstörung der CPU oder speziell

geschützter Speicherbereichen beim Bruch eines Siegels bewirken. Angriffe, die auf der Messung elektromagnetischer Abstrahlungen beruhen, können mittels Störsignalen wie zufälligen Schwankungen in der CPU-Taktrate erschwert werden [47]. Seitenkanal-Angriffe lassen sich in nicht-invasive Angriffe, wie die Messung des Energieverbrauchs und invasive Angriffe, die das Gerät beschädigen, aufteilen.

In der Kryptographie gilt ein Verfahren als gebrochen, wenn ein beliebiger Angriff mit geringerem Aufwand als die vollständige Abarbeitung des Schlüsselraums durchgeführt werden kann um den Schlüssel zu errechnen oder den Klartext aus dem Chiffrat ohne Kenntnis des Schlüssels zu gewinnen. Die Auswirkung dieses Bruchs des Verfahrens auf die Sicherheit müssen gesondert betrachtet werden. Ein Angriff, der den Aufwand eines Algorithmus mit 128-Bit Schlüssel von 2^{128} auf 2^{120} herabsetzt und zudem 2^{64} Klartext-Chiffrat Paare benötigt, wird nur geringe Auswirkungen auf Systeme mit niedrigen Sicherheitsanforderungen haben.

Kapitel 3

Mathematik asymmetrischer Verfahren

Dieses Kapitel beschäftigt sich mit asymmetrischen kryptographischen Verfahren, insbesondere mit Elliptischer Kurven Kryptographie. Dazu wird ein Einblick in die mathematischen Grundlagen der asymmetrischen Verfahren gegeben und Implementierungsaspekte werden angesprochen. Zusätzlich wird der Ablauf einiger asymmetrischen kryptographischen Verfahren vorgestellt.

3.1 Übersicht

Wie bereits in Abschnitt 2.9 angesprochen, wird in asymmetrischen kryptographischen Verfahren ein Schlüsselpaar aus einem öffentlichem und einem privatem Schlüssel verwendet. Beide stehen in einem Zusammenhang, der private (geheime) Schlüssel kann oftmals aus dem Öffentlichen errechnet werden. Der Aufwand hierzu ist jedoch so hoch dass das Verfahren sicher ist. Dazu wird eine Falltür-Funktion (Trapdoor-Function) verwendet, die in eine Richtung einfach zu lösen ist, deren Umkehrung aber sehr „teuer“ im Sinne von Rechenzeit oder Speicherplatz ist. Um ein asymmetrisches kryptographisches Verfahren zu brechen, also beispielsweise ein Chiffrat zu entschlüsseln ohne im Besitz des privaten Schlüssels zu sein, muss diese Umkehrung berechnet werden. Die üblicherweise verwendeten Falltürfunktionen sind die Integer-Faktorisierung (IFP = Integer Faktorisierungs Problem), der diskrete Logarithmus (DLP = Diskreter Logarithmus Problem) und der diskrete Logarithmus auf elliptischen Kurven (ECDLP). Das Integer Faktorisierung Problem nutzt aus, dass es einfach ist, zwei ganze Zahlen zu multiplizieren, aber sehr viel aufwändiger, aus einer Zahl $z = p * q \bmod n$ deren ganzzahligen Faktoren p und q zu bestimmen. Der schnellste derzeit bekannte Algorithmus zur Faktorisierung ist das Zahlkörpersieb (NFS = number field sieve) mit einer Laufzeit von $O((exp(1,923 + o(1))(log\ n)^{\frac{1}{3}}(log\ log\ n)^{\frac{2}{3}})$ [52].

Diese Laufzeit ist subexponentiell, so dass eine Verlängerung des Schlüssels um ein Bit keine Verdoppelung des Aufwandes und damit der Sicherheit mit sich bringt, wie dies bei symmetrischen Verfahren üblicherweise der Fall wäre.

Entsprechendes gilt für das zweite verwendete Problem, den diskreten Logarithmus. Dabei wird ausgenutzt, dass zwar die Exponentiation $b = a^x \bmod n$ einfach zu berechnen ist, nicht aber der diskrete Logarithmus, also die Berechnung von x bei gegebenem a, b, n. Der schnellste zur Zeit bekannte Algorithmus zur Berechnung des diskreten Logarithmus ist ebenfalls das Zahlkörpersieb mit derselben Laufzeit wie bei der Faktorisierung. Somit besitzen beide Verfahren dieselbe Verschlüsselungsstärke und sind subexponentiell.

Ein drittes Verfahren, die Berechnung des diskreten Logarithmus auf elliptischen Kurven, erfordert zur Umkehrung dagegen einen exponentiellen Algorithmus (Pollard-ρ) mit der Laufzeit $O(\sqrt{\pi q}\frac{1}{2r})$ bei der Verwendung von r parallelen Prozessoren [39]. Da dies die Verwendung kleinerer Schlüssel unter Beibehaltung des Sicherheitsgrades ermöglicht, ist dieses Verfahren zur Verwendung auf ressourcenbeschränkten Systemen besser geeignet als die anderen Verfahren und wird daher im Folgenden betrachtet.

3.1.1 Gruppen

Gruppen $G = (M, +)$ im mathematischen Sinne sind algebraische Strukturen mit einer Menge M und einer binären Operation +. Weiterhin müssen vier Forderungen erfüllt sein, damit G eine Gruppe bildet: [50]

1. Abgeschlossenheit: $a, b \in M \rightarrow a + b \in M$
2. Assoziativität: $\forall a, b, c \in M : (a + b) + c = a + (b + c)$
3. Identität: $\exists 0_G \in M : \forall a \in M : a + 0_G = 0_G + a = a$
4. Inversion: $\forall a \in M \exists b = a^{-1} \in M : a + b = b + a = 0_G$

   Falls G keine multiplikative sondern eine additive Gruppe ist wird $b = -a$ geschrieben.

Gilt zusätzlich $\forall a, b \in M : a + b = b + a$, spricht man von einer abelschen oder kommutativen Gruppe. Gibt es ein Element $g \in M$, so dass $M = \{g^0, g^1, g^2 ... g^n\}$, so nennt man die Gruppe zyklisch und g einen Generator der Gruppe. Falls M endlich ist, ist auch n endlich. Die Anzahl der Elemente in M wird als Ordnung der Gruppe bezeichnet. Die Ordnung eines Elements x der Gruppe ist dagegen der kleinste Wert m, so dass $x^m = 1$.

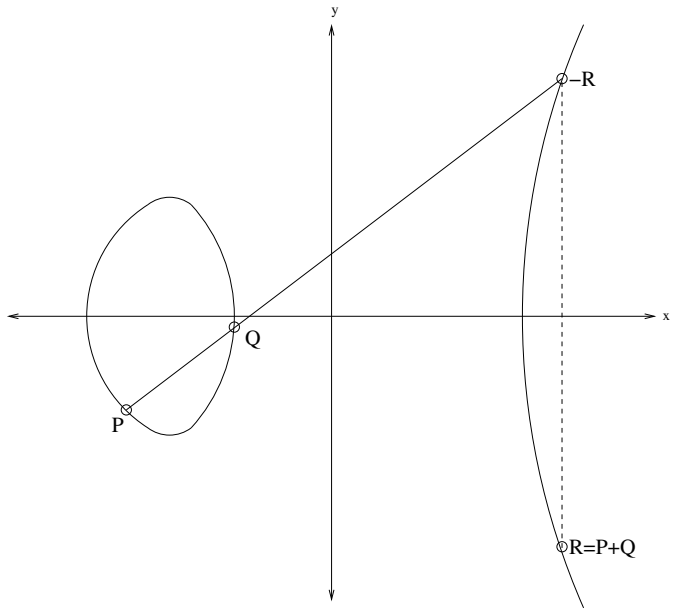

Abbildung 3.1: Addition zweier Kurvenpunkte P und Q

3.1.2 Körper

Ein Körper $K = (M, +, \bullet)$ ist eine algebraische Struktur, ebenfalls mit einer Menge M, jedoch zwei binären Operationen + und $\bullet$. Folgende Forderungen müssen erfüllt sein:

1. $(M, +)$ und $(M \setminus \{0\}, \bullet)$ sind abelsche Gruppen
2. Links- und Rechtsdistributivität: $\forall a, b, c \in M : a \bullet (b + c) = a \bullet b + a \bullet c$ und $\forall a, b, c \in M : (a + b) \bullet c = a \bullet c + b \bullet c$

Ein Körper mit einer endlichen Menge M wird als endlicher Körper oder Galois Körper (GF engl.: Galois Field) bezeichnet.

3.1.3 Elliptische Kurven

Bei der Verwendung des diskreten Logarithmus in kryptographischen Verfahren wird ausgenutzt, dass die Berechnung der Exponentiation $b = a^n \bmod p$ mit dem square-and-multiply-Algorithmus [42] mit linearem Aufwand in der Länge der binären Darstellung von n möglich ist. Die Berechnung des diskreten Logarithmus, also des kleinsten n, so dass $a^n = b \bmod p$ bei gegebenem a, b und p, erfordert allerdings – anders als bei der Berechnung des Logarithmus in $\mathbb{R}$ – wie bereits erwähnt einen sehr viel höheren Aufwand. Der diskrete Logarithmus wird in Verfahren wie dem Signaturalgorithmus DSA

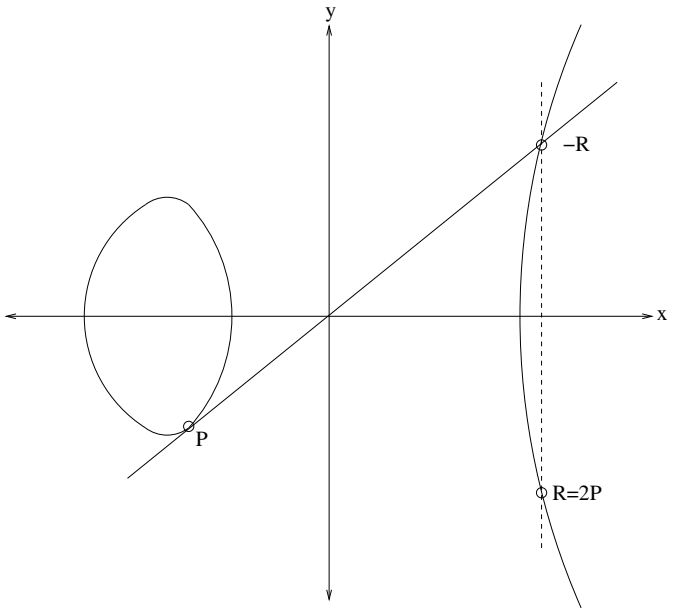

Abbildung 3.2: Verdoppelung des Kurvenpunktes P

oder dem Chiffriersystem ElGamal verwendet. Auch das Diffie-Hellman Key-Agreement-Protocol (DH) verwendet das DLP als mathematische Grundlage.

Die dem DLP zugrundeliegende Idee ist nicht auf eine Anwendung in der Gruppe Z_p^* beschränkt, auch in anderen endlichen Gruppen kann sie (in abgewandelter Form) verwendet werden. Für die Wahl einer anderen Gruppe könnte es zwei Anreize geben [39]. Entweder sind Rechenoperationen in einer Gruppe einfacher durchzuführen als in einer anderen. Damit kann eine Verlängerung des Schlüssels und damit eine Erhöhung der Sicherheit ohne zusätzliche Kosten an Rechenzeit erreicht werden. Ein zweiter Grund kann sein, dass das DLP in einer anderen Gruppe schwieriger zu lösen ist. Im diesem Fall wäre es möglich, die Schlüsselgröße zu reduzieren und dennoch denselben Grad an Sicherheit zu erreichen wie zuvor auf der Gruppe Z_p^*. Eine Gruppe, die diese Eigenschaft hat, ist die Gruppe der diskreten Lösungen einer elliptischen Kurve über einem endlichen Körper F_p. Eine elliptische Kurve über einem Körper K ist die Menge der Lösungen einer Gleichung der Form

$$y^2 = x^3 + ax + b \; ; \; a, b, x, y \in K$$

Weiterhin wird $4a^3 + 27b^2 \neq 0$ vorausgesetzt um Singularitäten, die die Sicherheit des Verfahrens verringern, auszuschließen. Die Lösungen einer elliptischen Kurve ist die Menge der Paare (x,y) aus $F_p \times F_p$, die die Gleichung der elliptischen Kurve erfüllen. Zusätzlich zu diesen Paaren wird der Punkt

$\mathbb{O}$ im Unendlichen definiert und der Menge der Lösungen hinzugefügt. Die Menge der Lösungen einer elliptischen Kurve bilden zusammen mit der Addition eine Gruppe. Dazu ist die Addition wie folgt definiert:

Das Ergebnis der Addition der Punkte P und Q auf der Menge der Lösungen einer elliptische Kurve ist (geometrisch betrachtet) der an der x-Achse gespiegelte dritte Schnittpunkt der Verbindungsgerade von P und Q mit der elliptischen Kurve (Abbildung 3.1, aus Gründen der Anschaulichkeit wird eine elliptische Kurve über $\mathbb{R}$ anstelle über einem endlichen Körper gezeigt). Zusätzlich wird das Ergebnis einer Addition eines Punktes P zu sich selbst definiert als der an der x-Achse gespiegelten Schnittpunkt der Tangente an der elliptischen Kurve in P mit der elliptischen Kurve (Abbildung 3.2). Das Ergebnis der Addition $R = P + Q$ und der Punktverdoppelung $R = 2P = P + P$ ist somit wieder ein Punkt auf der elliptischen Kurve. Arithmetisch ist die Addition der Punkte $P = (x_1, y_1)$ und $Q = (x_2, y_2)$ in affiner Darstellung zu $R = (x_3, y_3) = P + Q$ im Körper F_p definiert als

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ \lambda &= \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{falls } P \neq Q \\ \\ \frac{3x_1^2 + a}{2y_1} & \text{falls } P = Q \end{cases} \end{aligned}$$

Für den Fall $P = (x_1, y_1) = -Q = (x_1, -y_1)$ schneidet die Verbindungsgerade die elliptische Kurve kein weiteres Mal und als Summe von P und Q wird der Punkt $\mathbb{O} = (x_1, \infty)$ festgelegt<. Das Inverse des Punktes $P = (x, y)$ ergibt sich zu $-P = (x, -y)$.

Das Äquivalent zur Exponentiation des DLP ist in ECC-Systemen die Skalarmultiplikation von Kurvenpunkten. Die Skalarmultiplikation eines Punktes $R = sP = \underbrace{P + P + ... + P}_{s}; s \in \mathbb{I}^+$ kann mit einer Abwandlung des als „square and multiply“ bekannten Algorithmus effizient durchgeführt werden. Die Funktionsweise des „double and add“ Algorithmus ist wie folgt: Betrachte s als Binärdarstellung $s = \sum_{i=0}^{t-1} s_i 2^i$. Die Berechnung von $R = sP$ erfolgt dann mit

$R \leftarrow P$
for $i = t - 1$ downto 0 **do**
  if $s_i = 1$ **then**
    $R \leftarrow P + 2R$
  else
    $R \leftarrow 2R$
  end if
end for

Der Algorithmus benötigt $log_2(s)$ Punktverdoppelungen und durchschnittlich $\frac{1}{2} log_2(s)$ Punktadditionen im Vergleich zu s Punktadditionen bei der

naiven Methode der s-fachen Addition.

Analog zum DLP ist das ECDLP auf elliptischen Kurven definiert als die Berechnung von k aus $Q = kP$ bei gegebenen Kurvenpunkten Q und P. Ein ECC-Kryptosystem verwendet drei Parametersätze. Die Systemparameter bestehen aus den Faktoren a und b, die die elliptische Kurve identifizieren und der Zahl p, die den endlichen Körper F_p bestimmt. Weiterhin wird der Kurvenpunkt P als Startpunkt den Systemparametern zugeordnet. Der geheime Schlüssel ist die Zahl k, der Öffentliche der Kurvenpunkt $Q = kP$. Damit ist die Generierung von Schlüsselpaaren während der Laufzeit sehr einfach: Es muss lediglich zufällig ein Wert k gewählt werden und dann der Kurvenpunkt kP als zugehöriger öffentlicher Schlüssel berechnet werden. Im Gegensatz zu Verfahren, die auf dem IFP basieren, muss keine Primzahl bestimmt werden, beziehungsweise die Primeigenschaft einer zufällig gewählten Zahl nachgewiesen werden. Die Bestimmung von Systemparametern zur Laufzeit ist normalerweise nicht erforderlich. Üblicherweise werden eine elliptische Kurve und ein endlicher Körper gewählt die aus der Literatur als sicher bekannt sind [27]. Der Startpunkt kann ebenfalls vorab festgelegt werden.

Die Speicherung der Punkte kann in komprimierter Darstellung erfolgen. Dazu wird lediglich die Abszisse eines Punktes gespeichert und in einem zusätzlichen Bit vermerkt, ob die Ordinate größer oder kleiner als 0 ist. Dies spart nahezu 50% an Speicherplatz oder Bandbreite für die Speicherung oder Übermittlung eines öffentlichen Schlüssels. Die Rückgewinnung des eigentlichen Punktes erfolgt durch Einsetzen der Abszisse in die Kurvengleichung der elliptischen Kurve.

3.2 Wahl des Körpers

Die Wahl des Körpers F_p ist für eine effiziente Verarbeitung – insbesondere in ressourcenbeschränkten Systemen – von entscheidender Bedeutung. Allgemein wird ein Körper der Form $GF(p^m)$ gewählt. Dabei ist p eine Primzahl und wird Charakteristik des Körpers genannt und m ein positiver Wert $\in \mathbb{I}$. Die Wahl von m richtet sich nach der angestrebten Schlüssellänge. In [73] werden folgende Kandidaten genannt:

3.2.1 Primkörper

Die Wahl eines Primkörpers $GF(p)$ ohne zusätzliche Einschränkungen für p ist in Systemen ohne Hardware-Coprozessor für modulare Arithmetik kaum geeignet. Allerdings ermöglichen verallgemeinerte Mersenne-Primzahlen der Form $p = f(2^m)$ mit einem Polynom f niedrigen Grades wie $p = 2^{192} - 2^{64} - 1$ eine sehr einfache Berechnung der modularen Reduktion und werden beispielsweise in der Implementierung von [30] eingesetzt.

3.2.2 Körper mit Charakteristik 2

Ein Körper mit Charakteristik 2 (auch Binär-Körper genannt) hat die Form $GF(2^m)$. Üblicherweise werden Elemente des Körpers in der Polynomialdarstellung betrachtet. Insbesondere bei einer Hardwareimplementierungen der Arithmetik kann so eine hohe Verarbeitungsgeschwindigkeit erreicht werden, da einige Operationen der Körperarithmetik sehr effizient durchzuführen sind. Die Addition entspricht einem bitweisen XOR der Binärdarstellung der beiden Summanden und kommt daher ohne Übertrag zwischen den einzelnen Worten der Binärdarstellung aus. Zudem ist die Quadrierung in $GF(2^m)$ eine lineare Operation [33].

3.2.3 Zusammengesetzte Körper (Composite Fields)

Arithmetik auf Körpern der Form $GF((2^s)^r)$ kann ebenfalls effizient eingesetzt werden, wenn s klein genug gewählt wird, so dass die Inversion und Multiplikationen in $GF(2^s)$ vorab berechnet und zur Laufzeit durch Zugriffe auf eine Tabelle ausgeführt werden kann. Die Leistung übersteigt die von Binärkörpern, allerdings verbietet sich die Benutzung zusammengesetzer Körper aufgrund einer Angriffsmöglichkeit [28] auf ein ECC-System, das über einem zusammengesetzten Körper konstruiert wurde.

3.2.4 Optimal Extension Fields (OEF)

Eine stark von der zu verwendenden Hardwareplattform abhängige Implementierungsmethode besteht in der Verwendung von OEF. Als OEF werden Körper $GF(p^m)$ bezeichnet, die die Pseudo-Mersenne Primzahl $p = (2^n \pm c); n, c \in \mathbb{I}^+, log_2(c) \leq \frac{n}{2}$ wählen, so dass p möglichst nahe an der Maschinenwort-Größe liegt und als Faktor der Hardwaremultiplikation der CPU gewählt werden kann.

3.3 Koordinaten-Darstellungen

Zur Berechnung einer Punkte-Addition oder Punkte-Verdoppelung über einem Primkörper wird, nach der in Abschnitt 3.1.3 gezeigten Formel, der Term $\frac{y_2-y_1}{x_2-x_1}$ bzw $\frac{3x_1^2+a}{2y_1}$ benötigt. Die dazu notwendige Inversion ist in einem endlichen Körper eine sehr teure Operation. Die Inversion kann jedoch zum Preis einer höheren Anzahl von Additionen und Multiplikationen eingespart werden, wenn anstelle der in Abschnitt 3.1.3 verwendeten affinen Koordinatendarstellung eine alternative Darstellung gewählt wird. In [14] werden dazu mehrere Möglichkeiten genannt die im Folgenden besprochen werden.

3.3.1 Affine Koordinaten

Wie bereits in Abschnitt 3.1.3 beschrieben, berechnet sich die Punkte-Addition und Punkt-Verdoppelung auf elliptischen Kurven in affiner Koordinatendarstellung über einem Primkörper als

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ \lambda &= \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{falls } P \neq Q \\ \\ \frac{3x_1^2 + a}{2y_1} & \text{falls } P = Q \end{cases} \end{aligned}$$

Zur Berechnung einer Punkte-Addition werden somit eine Inversion, zwei Multiplikationen und eine Quadrierung benötigt, für die Berechnung einer Punkte-Verdoppelung eine Inversion, zwei Multiplikationen und zwei Quadrierungen. Die ebenfalls benötigten Additionen und Subtraktionen werden nicht betrachtet da sie insbesondere im Vergleich zur Inversion kaum ins Gewicht fallen. In [14] werden bei einer Implementierung auf einem 32 Bit Prozessor 0,2 Mikrosekunden für eine Addition, 1,3 Mikrosekunden für eine Multiplikation sowie 146,2 Mikrosekunden für eine Inversion genannt.

3.3.2 Projektive Koordinaten

In projektiver Darstellung wird ein Punkt durch das Tupel (X, Y, Z) repräsentiert, das sich aus der affinen Abbildung (x, y) als $X = xZ, Y = yZ, Z =$ beliebig $\neq 0$ berechnen lässt. In der Regel wird $Z = 1$ gewählt. Die Rückgewinnung der affinen Koordinaten eines Punktes (X, Y, Z) erfolgt mit $(\frac{X}{Z^2}, \frac{Y}{Z^3})$. Diese Art der projektiven Darstellung wird als Jacobi-Koordinaten Darstellung bezeichnet. Die Gleichung der elliptische Kurve lautet dann

$$Y^2 = X^3 + aXZ^4 + bZ^6$$

Die Punkt-Verdoppelung in Jacobi Koordinaten $2(X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$ ergibt [30]

$$\begin{aligned} X_2 &= (3X_1^2 + aZ_1^2)^2 - 8X_1Y_1^2 \\ Y_2 &= (3X_1^2 + aZ_1^2)(4X_1Y_1^2 - X_2) - 8Y_1^4 \\ Z_2 &= 2Y_1Z_1 \end{aligned}$$

Die Punkte-Addition $(X_1, Y_1, Z_1) + (X_2, Y_2, Z_2) = (X_3, Y_3, Z_3)$ ergibt

$$\begin{aligned} X_3 &= (Y_1Z_2^3 - Y_2Z_1^3)^2 - (X_1Z_2^2 + X_2Z_1^2)(X_1Z_2^2 - X_2Z_1^2) \\ Y_3 &= \frac{1}{2}[V(Y_1Z_2^3 - Y_2Z_1^3) - (Y_1Z_2^3 + Y_2Z_1^3)(X_1Z_2^2 - X_2Z_1^2)^3] \\ Z_3 &= Z_1Z_2(X_1Z_2^2 - X_2Z_1^2) \\ V &= (X_1Z_2^2 + X_2Z_1^2)(X_1Z_2^2 - X_2Z_1^2)^2 - 2X_3 \end{aligned}$$

Punktverdoppelung		Punktaddition		gemischte Darstellung	
$A \leftarrow 2A$	1I,2M,2S	$A \leftarrow A + A$	1I,2M,1S	$J \leftarrow J + A$	8M, 3S
$P \leftarrow 2P$	7M,3S	$P \leftarrow P + P$	12M,2S	$J \leftarrow J + C$	11M,3S
$J \leftarrow 2J$	4M,4S	$J \leftarrow J + J$	12M,4S	$C \leftarrow C + A$	8M, 3S
$C \leftarrow 2C$	5M,4S	$C \leftarrow C + C$	11M,3S		

Tabelle 3.1: Vergleich unterschiedlicher Koordinatendarstellung [14]

Neben der Darstellung in Jacobi-Koordinaten sind auch noch projektive Standard-Koordinaten, Chudnovsky-Koordinaten sowie Mischformen möglich. Bei Darstellungen in Mischform werden bei einer Punkt-Addition die beiden Summanden in unterschiedlichen Darstellungen betrachtet. Die Wahl der Koordinatendarstellung ist dabei immer vom Verhältnis der Geschwindigkeiten der Rechenoperationen, wie der Multiplikation und der Inversion, in der jeweiligen Implementierung abhängig. Tabelle 3.1 zeigt die Anzahl erforderlicher Inversionen (I), Multiplikationen (M) und Quadrierungen (S) für die Addition und Punktverdoppelung bei affiner (A), projektiver (P), Jacobi (J) und Chudnovsky (C) Darstellung.

3.4 EC-Diffie-Hellman Key-Agreement

Der Ablauf des Diffie-Hellman Key-Agreement zur Vereinbarung eines geheimen Schlüssel zwischen den Parteien A und B ist auf der Gruppe der Kurvenpunkte einer elliptischen Kurve wie folgt definiert.

Der Teilnehmer A wählt eine beliebige Zahl $k \in GF(p^m)$ und übermittelt $Q = kP$ an B (der Basispunkt P ist beiden als Systemparameter bekannt). B wählt seinerseits die Zahl $l \in GF(p^m)$ und übermittelt $R = lP$ zurück an A. A berechnet nun $T_1 = kR$ und B berechnet $T_2 = lQ$ und da $T_1 = kR = k(lP) = l(kP) = l(Q) = T_2$ verfügen beide Teilnehmer über den Punkt $T = T_1 = T_2$, dessen binäre Repräsentation als Schlüssel eines symmetrischen Verfahrens verwendet werden kann. Ein Angreifer kennt die Punkte P, Q und R, kann aber den Punkt T nicht berechnen, da er dazu den geheimen Schlüssel k oder l benötigen würde. Um diesen aus Q beziehungsweise R berechnen zu können, müsste er den diskreten Logarithmus auf der elliptischen Kurve lösen. Dies erfordert, wie in Abschnitt 3.1 beschrieben, einen in der Schlüsselgröße exponentielle Aufwand.

Das Diffie-Hellman Key-Agreement-Protocol erfordert von beiden Teilnehmern die Generierung einer Zufallszahl aus dem Körper $GF(p^m)$, sowie zwei Skalarmultiplikationen auf der elliptischen Kurve.

Das Diffie-Hellman Key-Agreement-Protocol ist allerdings anfällig gegen man-in-the-middle Angriffe. Wenn es einem Angreifer gelingt, alle Nachrichten zwischen A und B abzufangen und durch Eigene zu ersetzen, ist er in der Lage mit A (der ihn für B hält) einen gemeinsamen Schlüssel $K' = Pk'$

und mit B (der ihn für A hält) einen Schlüssel $K'' = Pl'$ zu vereinbaren. Somit ist er in der Lage, alle Nachrichten zwischen A und B zu entschlüsseln oder verändern und an den jeweiligen Gegenüber weiter zu leiten.

3.5 EC-DSA

Bei EC-DSA handelt sich es um eine Adaption des Digital Signature Algorithmus (DSA) auf das DLP über elliptischen Kurven. Der Signaturvorgang läuft bei Verwendung des Startpunktes P, des geheimen Schlüssels k und des öffentlichen Schlüssels $Q = kP$ wie folgt ab [39]:

1. Wähle eine zufällige Zahl m aus F_p. Die Zahl muss unvorhersagbar und statistisch einzigartig sein. Eine Wiederverwendung mit demselben öffentlichen Schlüssel muss ausgeschlossen werden. Diese Anforderung ist strenger als im ursprünglichen DSA-Algorithmus in dem lediglich eine zufällige Zahl gefordert wurde.

2. $R = (x, y) = mP; r = x \bmod n$ Falls $r = 0$ muss mit einer neuen Zahl m wieder begonnen werden

3. Berechne $m^{-1} \bmod n$

4. Berechne $s = m^{-1}(h + kr) \bmod n$ wobei h der SHA-1-Hashwert der zu signierenden Nachricht ist.
   Falls $s = 0$ muss mit einer neuen Zahl m wieder begonnen werden

5. Das Paar (r,s) ist die Signatur der Nachricht

Die Signaturverifikation läuft wie folgt ab:

1. Überprüfe dass r und s aus $[1, n-1]$ sind

2. Berechne $w = s^{-1} \bmod n$

3. Berechne $u_1 = hw \bmod n$ und $u_2 = rw \bmod n$ (h ist wieder der SHA-1 Hash der Nachricht)

4. Berechne $u_1P + u_2Q = R = (x_0, y_0)$ sowie $v = x_0 \bmod n$

5. Die Signatur ist gültig wenn $v = r$

Die Signaturgenerierung erfordert eine Skalarmultiplikation, die Generierung einer Zufallszahl, zwei Multiplikationen, eine Inversion und eine Addition in F_p. Die Verifikation zwei Skalarmultiplikationen, eine Punkteaddition sowie zwei Multiplikationen in F_p.

3.6 EC-ElGamal

Neben der in ElGamal ebenfalls definierten Signaturgenerierung und -verifikation ist insbesondere die Möglichkeit der asymmetrischen Verschlüsselung von Interesse. Wieder wird der geheime Schlüssel k und der öffentliche Schlüssel $Q = kP$ verwendet. Zur Verschlüsselung einer Nachricht $m \in M^*$ ist es erforderlich, diese zuerst mit einem umkehrbaren Verfahren mit dem Kurvenpunkt $K = (x, y)$ zu identifizieren. Danach erfolgt die Verschlüsselung mit dem öffentlichen Schlüssel Q wie folgt:

1. Wähle eine zufällige Zahl $r \in [1, n-1]$
2. Berechne $A = rP$ und $B = K + rQ$
3. Übermittle (A,B)

Die Entschlüsselung findet in diesem Fall wie folgt statt:

1. Berechne $B - kA = (K + rQ) - k(rP) = (K + r(kP)) - k(rP) = K$
   Die Subtraktion $B - kA$ wird als Addition $(B + (-(kA))$ des zu kA inversen Kurvenpunktes durchgeführt

Zuletzt muss der Kurvenpunkt $K = (x, y)$ wieder in die Nachricht M überführt werden. Die Sicherheit des Verfahrens liegt darin, dass ein Angreifer nicht in der Lage ist, die Berechnung $B - kA$ durchzuführen, da ihm zwar die Werte A und B bekannt sind, nicht aber der geheime Schlüssel k. Das Verfahren erfordert zur Verschlüsselung zwei Skalarmultiplikationen auf der elliptischen Kurve sowie eine Punktaddition. Zur Entschlüsselung werden eine Skalarmultiplikation und ebenfalls eine Punktaddition benötigt.

Das ElGamal Verfahren schreibt keine bestimmte Vorgehensweise zur Identifikation der Nachricht $m \in M^*$ mit dem Kurvenpunkt $K = (x, y)$ explizit vor. Da nicht zu jedem $m \in M^*$ ein Kurvenpunkt $(m, f(m))$ existiert, ist diese naheliegende Abbildung nicht verwendbar. Anstatt dessen müssen probabilistische Verfahren verwendet werden, die den Punkt wählen, dessen x-Koordinate m am nächsten kommt. Verfahren, die dies ermöglichen, sind beispielsweise in [43] und [44] beschrieben.

3.7 Zusammenfassung

Bei der Implementierung eines ECC-Kryptosystems stehen viele Optionen zur Verfügung. Neben der elliptischen Kurve selbst und der Wahl des Körpers über dem die Kurve betrachtet wird, stehen auch Details wie die Koordinatendarstellung und unterschiedliche Implementierungen der modularen Arithmetik zur Wahl. Diese können die Geschwindigkeit des Verfahrens entscheidend beeinflussen. Die Wahl der Implementierung hängt dabei immer

von den technischen Gegebenheiten ab. Da in handelsüblichen CPUs polynomielle Arithmetik nicht direkt unterstützt wird, sind Implementierungen über dem Körper $GF(2^m)$ eher für Hardwareimplementierungen geeignet. Implementierungen über OEF sind dagegen für ressourcenbeschränkte Mikrocontroller geeignet, da durch die Wahl der Charakteristik die Bitbreite der CPU berücksichtigt werden kann.

Kapitel 4

Implementierungskriterien kryptographischer Komponenten

Dieses Kapitel beschäftigt sich mit einer Klassifikation verschiedener Hardware-Plattformen und den Auswirkungen der Ressourcenbeschränkung auf die Wahl kryptographischer Algorithmen. Es wird ein Kriterienkatalog aufgestellt, der die Anwendbarkeit verschiedener kryptographischer Verfahren auf den unterschiedlichen Hardwareplattformen prüft.

4.1 Implementierungsmöglichkeiten

Zur Implementierung einer kryptographischen Komponente in einem Embedded System können drei Ansätze verfolgt werden:

- Softwareimplementierung auf dem bestehenden Mikrocontroller der Gerätesteuerung

- Softwareimplementierung auf einem zweiten Mikrocontroller (kryptographischer Coprozessor)

- Implementierung mit Hilfe dedizierter Kryptographie-Hardware (kryptographischer Coprozessor in ASIC-Technologie)

4.1.1 Softwareimplementierung auf bestehender Hardware

Die kostengünstigste Möglichkeit, ein Embedded System um eine kryptographische Absicherung der Kommunikation zu erweitern, besteht in der

Integration kryptographischer Algorithmen und Protokolle in die Software des Steuerungs-Mikrocontrollers. Falls auf dem verwendeten Mikrocontroller genügend Ressourcen wie Programmspeicher oder Rechenkapazität zur Verfügung stehen, kann die kryptographische Absicherung ohne zusätzliche Hardwarekosten erreicht werden. Andernfalls ist es oftmals möglich, durch Verwendung eines leistungsstärkeren Mitglieds der verwendeten Mikrocontrollerfamilie, die notwendigen Ressourcen zur Verfügung zu stellen. Aufgrund der starken Ähnlichkeit der einzelnen Mitglieder der Mikrocontroller einer Familie untereinander, können die bestehenden Entwicklungsergebnisse oftmals ohne Änderungen übernommen werden. Die Verwendung der bestehenden Hardware kommt mit einem minimalen Mehrbedarf an Ressourcen aus und ist daher die erstrebenswerte Implementierungsmethode.

4.1.2 Softwareimplementierung eines Krypto-Coprozessors

Teilweise ist die Implementierung einer Kryptographie-Komponente auf dem bestehenden Mikrocontroller nicht möglich, beispielsweise weil nicht ausreichend freie Ressourcen zur Verfügung stehen und kein Mikrocontroller derselben Familie mit zusätzlichen Ressourcen existiert. In diesem Fall ist es möglich, einen weiteren Mikrocontroller als kryptographischen Coprozessor zu verwenden. Die Kommunikation zwischen dem eigentlichen Steuerungs-Controller und dem kryptographischen Coprozessor kann entweder über vorgegebene Schnittstellen des Steuerungs-Controllers stattfinden oder der Coprozessor kann in den Datenpfad des Systems eingefügt werden und so für eine transparente Absicherung der Kommunikation sorgen. Diese Lösung kann selbst dann verwendet werden, wenn es sich bei dem Steuerungs-Controller um ein geschlossenes System handelt, wie ein fertiges zugekauftes Bauteil, das der Entwickler nicht modifizieren kann. Der Nachteil des Verfahrens im Vergleich zu der reinen Softwareimplementierung des vorherigen Abschnitts besteht im höheren Energie- und Platzbedarf und den höheren Kosten durch den zusätzlichen Mikrocontroller sowie die erforderlichen diskreten Bauteile.

4.1.3 Hardwareimplementierung eines Krypto-Coprozessors

Die Verwendung eines Krypto-Coprozessors in ASIC-Ausführung ist bezüglich Durchsatz und Ressourcenbedarf einem Mikrocontroller üblicherweise weit überlegen. Durch den Einsatz eines speziellen Krypto-Coprozessors als Erweiterung in einem Mikrocontroller erhöht sich der Preis des Controllers um schätzungsweise 20 bis 30 Prozent [16]. Für den Fall, dass durch den Einsatz eines in den Mikrocontroller integrierten Krypto-Coprozessors ein ansonsten notwendiger zweiter Mikrocontroller eingespart werden kann, der lediglich als Krypto-Coprozessor verwendet wird, sind die Mehrkosten geringer als die zu erwartenden Kosten durch die Verwendung von zwei CPUs. Allerdings beschleunigen die Krypto-Coprozessoren üblicherweise le-

Hersteller, Name	RSA 512	RSA 1024	RSA 2048	DSA 512	DSA 1024
Hitachi CoProcessor	68/-	210/-	480/-	-	-
Siemens CCP	60/20	450/-	-	95/175	-
Siemens ACE	37/10,3	230/24	1475/268	50/90	143/271
NEC SuperMAP	16/2	100/7	750/45	31/70	-
SGS MAP	20/2	110/5	780/100	25/80	100/160
Philips FameX	37/10	160/25	1100/54	58/82	150/225
Atmel MEU	34/5	165/16	-	-	-

Tabelle 4.1: Zeitaufwand von Krypto-Coprozessoren für Signaturgenerierung/-Verifikation in Millisekunden [32]

diglich Rechenoperationen mit modularer Arithmetik und beinhalten keine vollständigen Kryptoeinheiten mit Signaturgenerierung und -Verifikation. Diese Funktionen müssen meist weiterhin vom vorhandenen Mikrocontroller ausgeführt werden. Daher ist auch bei der Verwendung eines Krypto-Coprozessors oftmals ein dedizierter zweiter Mikrocontroller erforderlich, der mit Hilfe des Coprozessors die kryptographischen Algorithmen implementiert oder aber der vorhandene Mikrocontroller muss mit zusätzlichen Ressourcen ausgestattet werden. Der Einsatz von Krypto-Coprozessoren im Bereich der asymmetrischen Kryptographie ist oft unumgänglich für Systeme mit hohen Anforderungen an den Durchsatz, insbesondere für Algorithmen, die nicht auf Elliptischen Kurven Kryptographie, sondern dem DLP auf Z_p^* oder dem Integer-Faktorisierungsproblem beruhen. Tabelle 4.1 enthält Leistungsdaten einiger Hardware Krypto-Coprozessoren. Es ist die benötigte Zeit zur Signaturgenerierung/-Verifikation mit verschiedenen Schlüsselgrössen zwischen 512 und 2048 Bit Länge angegeben.

Der Krypto-Coprozessor kann entweder als vom eigentlichen Mikrocontroller getrennte Komponente implementiert sein oder direkt auf dem Mikrocontroller integriert werden. Zu einigen handelsüblichen Mikrocontrollern, wie der Atmel AVR-Familie, werden vom Hersteller Varianten mit Kryptographie-Erweiterungen angeboten. Diese sind entweder in Form von Recheneinheiten für modulare Arithmetik oder vollständigen symmetrischen Algorithmen wie DES oder SHA-1 sowie Hardware-Zufallszahlengeneratoren ausgeführt. Im Falle der Atmel AVR Mikrocontroller wird der SC16 Coprozessor verwendet. Diese Erweiterungen sind überwiegend für den Einsatz in Smartcards gedacht. Einige dieser Chips werden allerdings auch für die Verwendung in anderen Systemen produziert wie der Atmel A97SC3201, der für den Einsatz in Personalcomputern im Rahmen des Trusted Computing gedacht ist [29]. Dennoch sind kryptographische Coprozessoren als Option handelsüblicher Mikrocontroller bislang die Ausnahme.

4.2 Überblick über einige Hardwareplattformen

Als Hardwareplattformen werden ausschließlich 8-Bit Mikrocontroller in Betracht gezogen. Trotz der grossen Fortschritte im Bereich der Hochleistungs-CPUs wie sie in üblichen PCs zu finden sind, ist die weitaus größere Zahl der produzierten CPUs relativ leistungsschwache Mikrocontroller. Zahlen aus dem Jahr 2002 zeigen, dass mehr als 50 Prozent der produzierten CPUs 8-Bit Datenbreite hatten [71]. Dies ist nicht verwunderlich, da die Aufgaben, die Mikrocontroller in Gegenständen des täglichen Lebens wie Radios, elektronischen Zahnbürsten oder Geschirrspülmaschinen erfüllen müssen, oftmals relativ wenig Rechenleistung erfordern und deutlich mehr Geräte mit Mikrocontrollern als vollwertige PCs eingesetzt werden. Auch im hier betrachteten Szenario der Heimautomatisierung werden an die Mikrocontroller relativ geringe Anforderungen gestellt, um ihre Aufgaben, wie die Übermittlung der gemessenen Temperatur an ein anderes System oder das Senden eines Schliesscodes, zu erfüllen. Da die implementierten kryptographischen Verfahren lediglich eine Ergänzung der Systeme um sichere Kommunikation darstellen sollen, ist es unumgänglich, dies mit möglichst wenigen zusätzlichen Mittel zu bewerkstelligen.

In heute verwendeten Prozessoren werden überwiegend zwei verschiedene Architekturen eingesetzt. Die Akkumulator-Architektur verwendet für alle Befehle mit zwei Operanden ein spezielles Akkumulatorregister als einen der Operanden. Dieses wird üblicherweise auch als Ziel der Operation verwendet. Dies ermöglicht sehr kompakte Befehle, da ein Operand und das Ziel der Operation implizit festgelegt sind. Allerdings ist, wenn nicht zusätzliche Register vorhanden sind, jeder Befehl der mehrere Operanden verwendet mit einem (oftmals langsamen) Hauptspeicherzugriff verbunden.

Die alternativ verwendete Architektur ist die General-Purpose-Register-Architektur, die eine größere Anzahl Register verwendet, die als Operand und Ziel einer Operation eingesetzt werden können. Die General-Purpose-Register-Architektur kann weiter klassifiziert werden, ausschlaggebend ist hierbei, wie viele Operanden im Hauptspeicher liegen dürfen. Werden nur Register als Operanden und Ergebnis der Operation zugelassen, spricht man von einer Register/Register oder Load/Store-Architektur. Die meisten RISC-Architekturen verfolgen diesen Ansatz. Von Register/Speicher-Architekturen spricht man, wenn ein Operand oder das Ziel im Hauptspeicher liegen darf und von Speicher/Speicher-Architekturen, wenn mehrere Operanden und das Ziel der Operation im Hauptspeicher liegen dürfen.

Alle hier vorgestellten Mikrocontroller verwenden die Harvard Speicherarchitektur, verfügen also über getrennte Programm- und Datenspeicher. Der Programmspeicher ist in allen vorgestellten Systemen als ROM oder Flash-ROM Speicher ausgelegt, kann also lediglich für konstante Werte verwendet werden.

Zusätzlich verfügen die Mikrocontroller oftmals über EEPROM-Speicher

zur Speicherung von Daten, die auch ohne Stromzufuhr erhalten bleiben sollen. Da EEPROM in der Regel deutlich langsamer im Zugriff ist als der RAM-Speicher, ist eine Verwendung des EEPROM als Zwischenspeicher der Algorithmen meist ausgeschlossen. Bei Atmel AVR Mikrocontrollern wird ein Schreibzugriff auf das EEPROM mit konstant 8,5 ms angegeben, während auf das RAM innerhalb zweier Takte zugegriffen werden kann. Bei einem CPU-Takt von 8 MHz sind dies 0,25 μs. Zudem werden für das EEPROM lediglich 100000 Schreibvorgänge garantiert.

4.2.1 Atmel AVR/MegaAVR Plattform

Die Mikrocontroller der Atmel AVR Familie sind 8-Bit RISC Mikrocontroller. Sie lassen sich in zwei Klassen einteilen, die sich vor allem dadurch unterscheiden, dass die Mitglieder der MegaAVR Familie im Gegensatz zu denen der AVR Familie über 8-Bit Hardwaremultiplikation verfügen. Die Mikrocontroller der AVR Familie verfügen über 1 bis 8 Kilobyte ROM und über 128 bis 512 Byte RAM. Außerdem sind zwischen 64 und 512 Byte EEPROM auf dem Chip integriert. Die AVR-Mikrocontroller können mit maximal 12 MHz getakted werden. Da der überwiegende Teil der Maschinenbefehle innerhalb eines Taktes ausgeführt wird, entspricht das einer Leistung von bis zu 12 MIPS (Million Instructions per Second). Die MegaAVR Mikrocontroller verfügen über zwischen 8 und 128 Kilobyte ROM und 512 bis 4096 Byte RAM. Auch die Größe des EEPROM liegt zwischen 512 und 4096 Byte. Der maximale Takt beträgt bis zu 16 MHz, was einer Leistung von bis zu 16 MIPS entspricht. AVR CPUs sind Register/Register Maschinen mit einem relativ umfangreichen Befehlssatz, verfolgen aber keinen reinen General-Purpose-Register Ansatz, da für viele Befehle bestimmte Gruppen von Registern als Operanden vorgeschrieben sind.

4.2.2 Microchip PIC Plattform

Die ebenfalls häufig verwendeten PIC Mikrocontroller sind in einer Vielzahl von Ausstattungen verfügbar. Es handelt sich ebenfalls um RISC-Architekturen, allerdings in Akkumulator-Architektur. Die leistungsschwächere PIC16 Familie wird mit RAM-Größen zwischen 25 und 368 Byte RAM angeboten sowie zwischen 512 und 8192 Byte ROM. Die leistungsstärkeren PIC18 Mikrocontroller verfügen über Hardwaremultiplikation und einen deutlich vergrößerten Befehlssatz und sind mit 512 bis 4096 Byte RAM sowie 2 bis 64 Kilobyte ROM ausgestattet. Sowohl PIC16 als auch PIC18 sind mit bis zu 40 MHz taktbar. Da ein Befehlszyklus aus vier Taktzyklen besteht entspricht das einer Leistung von maximal 10 MIPS.

	RAM in Byte	ROM in Byte	EEPROM in Byte	HW-Mult.	MIPS	Klasse
AVR	128-512	1k-8k	64-512	nein	12	I
MegaAVR	512-4k	8k-128k	512-4k	ja	16	II
PIC16	25-368	512-8k	64-256	nein	10	I
PIC18	512-4k	2k-64k	256-1k	ja	10	II
I8051	64-(256+2k)	4k-64k	bis 128k	ja	5	I-II
ST6	64-320	1k-8k	–	nein	8	I
ST7	512-2k	4k-60k	–	ja	8	II

Tabelle 4.2: Parameter von Mikrocontrollern

4.2.3 ST Microelectronics ST Plattform

Die 8-Bit RISC Mikrocontroller der Firma ST Microelectronics sind ebenfalls in zwei Familien erhältlich. Die leistungsschwächere ST6 Familie verfügt über 64 bis 256 Byte RAM sowie 1 bis 8 Kilobyte ROM. Die stärkere ST7 Familie verfügt über bis zu 2048 Byte RAM, maximal 64 Kilobyte ROM und Hardwaremultiplikation. Beide Familien können mit bis zu 16 MHz getaktet werden. Der kürzeste Befehlszyklus beträgt 2 Takte, so dass eine Leistung von maximal 8 MIPS erreicht werden kann. Die ST Mikrocontroller verwenden ebenfalls die Akkumulator-Architektur.

4.2.4 Intel 8051 Plattform

Die CISC-Mikrocontroller der Intel 8051 Architektur werden von vielen Herstellern in Lizenz mit eigenen Modifikationen gefertigt, weshalb eine Angabe der Leistungsdaten schwer fällt. Alle 8051 verfügen über eine 8-Bit Hardwaremultiplikation. Die ROM-Größe einiger Modelle beträgt bis zu 64 Kilobyte, die maximale RAM-Größe 256 Byte. Zusätzlich verfügen 8051 Mikrocontroller häufig über erweitertes RAM (XRAM), das jedoch nicht direkt adressiert werden kann. Hierbei sind Größen bis zu 128 Kilobyte erhältlich, typisch sind allerdings Werte bis maximal 2 Kilobyte. Einige Modelle der 8051 Familie können mit bis zu 40 MHz getaktet werden. Aufgrund einer internen Taktteilung (üblicherweise um den Faktor 12), die in fast allen Modellen angewandt wird, beträgt die Leistung jedoch meist weniger als 5 MIPS. Einige Ausnahmen erreichen jedoch pro Megahertz Takt eine Leistung von einem MIPS. Die Firma Cygnal bietet CPUs auf Basis der 8051 Plattform an, die bei einem CPU-Takt von 50 MHz eine Leistung von 100 MIPS erreichen. Die Architektur der 8051 Familie ist ebenfalls eine Akkumulator-Architektur, die Mikrocontroller verfügen jedoch auch über Universalregister.

4.2.5 Klassifikation der Plattformen

Bei der Betrachtung der einzelnen Mikrocontroller sind zwei Klassen zu erkennen. Die Klasse der leistungsschwächeren Geräte verfügt über keine Hardwaremultiplikation und über maximal 256 Byte RAM. Die leistungsfähigeren Geräte verfügen über 8-Bit Hardwaremultiplikation und in der Regel zwischen 512 und 4096 Byte RAM. Während Geräte der leistungsschwachen Kategorie über maximal 8 Kilobyte ROM verfügen, sind die leistungsstärkeren Geräte mit 8 bis 64 Kilobyte erhältlich. Die Leistung in MIPS unterscheidet sich zwischen Geräten der unterschiedliche Klassen dagegen nicht signifikant. Tabelle 4.2 fasst die Parameter der einzelnen Mikrocontroller zusammen.

4.2.6 Mehrkosten bei Verwendung eines leistungsstärkeren Mikrocontrollers

Soll ein bestehendes Embedded System um die kryptographische Absicherung der Kommunikation erweitert werden, so wird es oftmals erforderlich sein, den Mikrocontroller, dessen Ressourcen aus Kostengründen auf die bisherige Aufgabe möglichst genau abgestimmt waren, durch ein leistungsstärkeres Modell zu ersetzen. Im Folgenden werden beispielhaft für Mikrocontroller der PIC16 und PIC18 Familien die prozentualen Mehrkosten für die Vergrößerung der Ressourcen RAM und ROM und die Mehrkosten beim Einsatz der PIC18 anstelle der PIC16 Familie angegeben. Die Kosten sind laut Herstellerangaben in der jeweils günstigsten Packungsform und Lieferart. Soweit möglich wurde bei der Auswahl der verglichenen Controller auf identische Leistungsmerkmale außerhalb des jeweils Betrachteten geachtet.

Tabelle 4.3 zeigt die Mehrkosten durch die Verwendung eines Mikrocontrollers derselben Familie, der aber über die doppelte Menge RAM oder ROM verfügt. Wie ersichtlich ist, bewegen sich die zu erwartenden Mehrkosten in der Regel deutlich unterhalb von 10%. In Tabelle 4.4 sind die Mehrkosten bei Verwendung der leistungsstärkeren Prozessorfamilie aufgezeigt. Während bei leistungsschwachen Mitgliedern der PIC16 Familie der Aufpreis auf ein vergleichbares Modell der PIC18 Familie mit 75% sehr hoch ist, fällt es bei stärkeren PIC16 Prozessoren mit 15% weniger stark ins Gewicht. Beide Tabellen zeigen, dass es deutlich günstiger ist, einen leistungsstärkeren Mikrocontroller einzusetzen anstelle eines zweiten Mikrocontroller, der lediglich als kryptographischer Coprozessor eingesetzt wird. In diesem Fall lägen die Mehrkosten oberhalb von 100%, da zusätzlich zum Mikrocontroller weitere Hardware-Komponenten zur Taktgenerierung und Stromversorgung benötigt würden.

Familie	alter MC			neuer MC			Aufpreis	Aufpreis
	Produkt	RAM	ROM	Produkt	RAM	ROM	in $	in %
PIC18	F2220	512	4k	F2320	512	8k	0,37	8,3
PIC18	F4220	512	4k	F4320	512	8k	0,38	7,7
PIC18	F242	768	8k	F252	1536	16k	0,42	8,8
PIC18	F448	768	16k	F458	1536	32k	0,45	7,7
PIC16	HV540	25	768	C56A	25	1536	0,09	6,4
PIC16	F873	192	7k	F876	368	14k	0,57	13,2
PIC16	F873A	192	7k	F876A	368	15k	0,31	7,7

Tabelle 4.3: Mehrkosten durch Verwendung eines MC mit vergrößerten RAM- und ROM-Kapazitäten

alter MC			neuer MC			Aufpreis
Produkt	ROM	Preisspanne	Produkt	ROM	Preisspanne	in %
PIC16	3584	$1,61-$2,81	PIC18	4k	$2,78-$4,91	ca. 75
PIC16	7168	$2,41-$4,97	PIC18	8k	$3,15-$5,29	15-30
PIC16	14336	$4,10-$5,11	PIC18	16k	$4,73-$5,81	ca. 15

Tabelle 4.4: Mehrkosten durch Verwendung eines MC der leistungsstärkeren Familie

4.3 Eigenschaften kryptographischer Algorithmen

Die in Embedded Systemen verwendeten Mikrocontroller weisen vor Allem in den Bereichen RAM- und ROM-Kapazität starke Einschränkungen im Vergleich mit leistungsstärkeren Systemen auf. Eine weitere Einschränkung liegt in der zur Verfügung stehenden Energie der Systeme, die üblicherweise mit Batterien betrieben werden. Daher sind RAM- und ROM-Bedarf sowie die Geschwindigkeit die wichtigsten Eigenschaften kryptographischer Algorithmen für den Einsatz in Embedded Systemen.

4.3.1 RAM-Bedarf

Da die RAM-Größe vieler Mikrocontroller stark limitiert ist, ist der RAM-Bedarf der kryptographischen Verfahren eine entscheidende Größe bei der Auswahl eines geeigneten Algorithmus. Hierbei sind zwei Parameter von Bedeutung. Der RAM-Bedarf während des Keysetup oder der Ver- beziehungsweise Entschlüsselung der Daten geben den Maximalbedarf an RAM an. Der globale Speicherbedarf gibt die Größe des benötigten RAM an, um die im Keysetup berechneten Rundenschlüssel, beziehungsweise im Fall von Stromchifffren den inneren Zustand des Algorithmus, zu speichern. Der globale Speicherbedarf wird nicht nur während der eigentlichen Ver- oder Entschlüsselung sondern während des gesamten Nutzungszeitraums eines Schlüssels benötigt, wenn nicht das Keysetup mehrfach durchgeführt werden soll.

4.3.2 ROM-Bedarf

Der ROM-Bedarf ist – außer bei den asymmetrischen Kryptoalgorithmen – von weniger grosser Bedeutung, da die meisten Algorithmen deutlich weniger als 2000 Byte benötigen, aber selbst Mikrocontroller der leistungsschwächeren Klasse im Allgemeinen mit großer ROM-Kapazität erhältlich sind. Da Mikrocontrollern in Harvard-Architektur getrennten Speicher für Programm und Daten verwenden muss bei der Ablage konstanter Werte im ROM beachtet werden, dass der Lesezugriff auf das ROM gegebenenfalls langsamer ist als ein Lesezugriff auf das RAM. So erfordert beispielsweise der Lesezugriff auf den Programmspeicher auf einem Atmel AVR Mikrocontroller drei Takte, während der Lesezugriff auf das RAM nach zwei Takten abgeschlossen ist. Zudem ist denkbar, dass aufgrund der Trennung von Programm und Datenspeicher der effiziente Zugriff auf den Programmspeicher zur Laufzeit überhaupt nicht möglich ist. Auf Architekturen mit dieser Einschränkung ist die Implementierung von Algorithmen, die eine SBox oder andere Tabellen verwenden, kaum möglich.

4.3.3 Durchsatz

Der Durchsatz eines kryptographischen Verfahrens ist aus zwei Gründen von Bedeutung. Erstens kann der Durchsatz des kryptographischen Verfahrens in Systemen mit hohem Datenaufkommen den Durchsatz des Gesamtsystems begrenzen. Viel wichtiger bei einer Vielzahl von Anwendungen ist jedoch, dass der Energieverbrauch der meisten Geräte proportional zur Betriebsdauer ist. Üblicherweise verfügen Mikrocontroller über Energiesparmodi, in die während Untätigkeit geschaltet werden kann. Somit benötigen langsame Verfahren mehr Energie.

4.3.4 RAM-Bedarf der Rundenfunktion

Der RAM-Bedarf der Rundenfunktion kann bei Register/Register-Architekturen einen grossen Einfluss auf die Performance eines Algorithmus haben. Der Zugriff auf den Hauptspeicher benötigt üblicherweise mehrere Takte, während ein Zugriff auf ein CPU-Register in einem Takt möglich ist. Falls es auf einer Register/Register-Architektur möglich ist, alle während einer Runde verwendeten Daten im Registersatz zu halten, kann eine deutliche Leistungssteigerung erzielt werden. Aufgrund der Anzahl der Prozessorregister und des RAM-Bedarfs der Rundenfunktion kann jedoch noch keine verlässliche Aussage getroffen werden, ob es möglich sein wird, den Algorithmus so zu implementieren, dass die veränderlichen Daten vollständig in den Prozessorregistern abgelegt sind. Details in der Implementierung der Register – beispielsweise können in Atmel Mikrocontrollern in vielen Operationen nur bestimmte Register als Operanden eingesetzt werden – können

eine Implementierung vereiteln. Es kann lediglich die Negativaussage getroffen werden, dass, wenn der RAM-Bedarf der Rundenfunktion die Anzahl der Prozessorregister übersteigt, keine effiziente Implementierung möglich sein wird. CPUs mit Akkumulator-Architektur sind hiervon jedoch nicht betroffen, da diese ohnehin für alle Operationen einen Wert des Hauptspeichers als zweiten Operanden verwenden. Somit ist die Performance auf Akkumulatormaschinen unabhängig vom RAM-Bedarf der Rundenfunktion. Eine Ausnahme von dieser Regel bilden Akkumulator-Architekturen mit zusätzlichen Allzweckregistern wie die Intel 8051 Architektur.

4.3.5 Schlüsselgröße

Im Gegensatz zur Blockgröße und der damit verbundenen unteren Grenze für den RAM-Bedarf der Rundenfunktion hat die Schlüsselgröße üblicherweise nur geringe Auswirkungen auf die Performance einer Implementierung. Da der Schlüssel in fast allen Algorithmen ohnehin zu einem deutlich größeren Rundenschlüssel erweitert wird, ist es nicht möglich, den Schlüssel in den CPU-Registern zu speichern. Eine wichtige Eigenschaft für Implementierungen mit geringem RAM-Bedarf ist dagegen die iterative Berechenbarkeit des Rundenschlüssels aus dem Rundenschlüssel der vorangegangenen Runde. Ist eine solche iterative Berechnung möglich, so kann auf Kosten des Durchsatzes der RAM-Bedarf deutlich reduziert werden, indem der Rundenschlüssel während der jeweiligen Runden berechnet wird.

4.3.6 Key-Setup Aufwand

Die Bedeutung des Key-Setup Aufwands kann in den meisten Fällen ignoriert werden, da ein einmal erzeugter Rundenschlüssel üblicherweise zur Verarbeitung einer Vielzahl von Datenblöcken verwendet wird. Das Verhältnis des Zeitaufwands von Keysetup und Verarbeitung eines Blocks kann jedoch als Anhaltspunkt zur Bestimmung des Aufwands bei der Verarbeitung kleiner Nachrichten mit jeweils anderem Schlüssel dienen.

4.3.7 Verwendete Operationen

Die in der Rundenfunktion verwendeten Operationen sind insbesondere in drei Fällen von Bedeutung:

(1) Verwendet ein Algorithmus Multiplikationen, so ist eine Anwendung auf CPUs ohne Unterstützung einer Hardwaremultiplikation praktisch unmöglich. Abgesehen von IDEA wird die Hardwaremultiplikation jedoch nur für asymmetrische Kryptoverfahren benötigt, die aufgrund ihres RAM-Bedarfs ohnehin nur auf der Klasse der leistungsfähigeren CPUs lauffähig sind. Diese CPUs verfügen alle über eine 8-Bit Hardwaremultiplikation.

(2) Neben der Multiplikation wird häufig die bitweise logische Rotation in Kryptoalgorithmen verwendet. Im Gegensatz zu leistungsfähigeren 32 Bit

CPUs, die oftmals innerhalb eines Taktes eine Rotation eines 32-Bit Registers um beliebig viele Stellen zulassen, ist in Mikrocontrollern üblicherweise nur eine Rotation eines 8-Bit Registers um eine Bit-Stelle möglich. Dadurch ist es erforderlich, Rotationen um mehrere Stellen durch mehrfache Anwendung einstelliger Rotationen durchzuführen. Dabei kann es zu Angriffen durch Messungen des Zeitbedarfs der Rotationen kommen [31]. Dies kann verhindert werden, indem mit Hilfe von NOP-Befehlen eine konstante Laufzeit des Algorithmus erzwungen wird.

(3) Zusätzlich ist auch die im Algorithmus verwendete Datenbreite von Bedeutung für eine Implementierung auf 8-Bit Mikrocontroller. Auf einer 8-Bit CPU können 32-Bit Logik-Operationen mit üblicherweise vier Maschinencode Instruktionen ausgeführt werden, da kein Übertrag erforderlich ist. Dahingegen ist der Aufwand einer 32-Bit Addition auf einer 8-Bit CPU davon abhängig, ob ein ADD-with-Carry Befehl implementiert ist. Ist dies der Fall, so kann eine 32-Bit Addition im Optimalfall mit 4 Befehlen durchgeführt werden. Eine Architektur ohne ADD-with-Carry, wie beispielsweise eine PIC16 CPU, benötigt dagegen deutlich mehr Befehle, da der Übertrag der vorangegangenen Addition mit mindestens drei Befehlen inklusive einem vom Zustand des Carry-Flag abhängigen Sprung erfolgen muss. Noch deutlicher wird die Auswirkung der Datenbreite bei der Verwendung von Multiplikationen, die die Datenbreite des Prozessors übersteigen. Eine 16-Bit Multiplikation unter Verwendung der 8-Bit Hardwaremultiplikation des PIC18 benötigt mindestens 24 Befehle.

Grundsätzlich sollte bei einer Implementierung eines kryptographischen Algorithmus auf eine konstante, nicht vom Klartext oder dem Schlüssel abhängige, Laufzeit geachtet werden. Da Messungen der Laufzeit sonst Rückschlüsse auf innere Zustände des Algorithmus zulassen sollte beispielsweise bei datenabhängigen Sprüngen stets mit Hilfe von NOP-Befehlen für beide Zweige eine konstante Laufzeit erzwungen werden [31, 45].

4.4 Eigenschaften sicherer Kanäle

Bei der Implementierung kryptographischer Systeme zur Bereitstellung eines sicheren Kommunikationskanals werden vor Allem drei Komponenten benötigt. Neben einer Block- oder Stromchiffre wird ein MAC-Algorithmus und üblicherweise ein Zufallszahlengenerator benötigt. Letzterer wird unter anderem verwendet, um die Initialisierungsvektoren zu generieren oder einen zufälligen Sitzungsschlüssel zu berechnen. Bei der Verwendung einer Blockchiffre als Chiffrieralgorithmus besteht die Möglichkeit, diese zur Berechnung des MAC als CBC-MAC zu verwenden. Wird eine Stromchiffre zur Verschlüsselung verwendet, so ist eine zusätzliche Blockchiffre oder ein kryptographisches Hashverfahren erforderlich, da es keine Möglichkeit gibt, eine beliebige Stromchiffre zur Berechnung eines MAC zu verwenden.

Aufgrund von Anfälligkeiten einiger Chiffren gegenüber Related-Key-Angriffen (siehe Abschnitt 2.8.1), ist in generischen Kompositionen von Chiffrierung und Authentizitätssicherung davon abzuraten, für Verschlüsselung und CBC-MAC denselben Schlüssel zu verwenden. Daher beträgt der RAM-Bedarf des kombinierten Chiffrierungs- und MAC-Algorithmus eines generischen AE-Systems stets mindestens das Zweifache des globalen RAM-Bedarfs der verwendeten Blockchiffre. In Verfahren wie EAX oder CWC ist die Verwendung nur eines Schlüssels jedoch oftmals zulässig. Da in den meisten Mikrocontrollern RAM deutlich stärker beschränkt ist als ROM, lohnt es sich in der Regel eines dieser Verfahren einzusetzen. Diese erfordern zwar üblicherweise mehr Programmcode, aufgrund des einzelnen Schlüssels jedoch deutlich weniger RAM.

4.5 Zusammenfassung

Wie bereits erläutert ist es aus Kostengründen stets vorzuziehen, den für eine kryptographische Absicherung der Kommunikation notwendige Ressourcenmehrbedarf durch Verwendung eines leistungsfähigeren Mitglieds der Mikrocontroller-Familie zu decken. Die Wahl der kryptographischen Algorithmen sollte dabei stets auf die zu verwendende Hardwareplattform zugeschnitten sein. Neben den verwendeten Befehlen wie der Multiplikation oder einer Addition mit einer größeren Datenbreite wie der Prozessor verwendet ist insbesondere die geringe RAM Kapazität von Mikrocontrollern zu beachten. Bei Systemen die ein sehr schmalbandige Kommunikation verwenden ist es in der Regel vorzuziehen, eine Berechnung des Rundenschlüssel zur Laufzeit durchzuführen. Die Verringerung des Durchsatzes ist bei einer Kommunikationsbandbreite unterhalb von 50 Kilobit/Sekunde von geringerer Bedeutung als die erzielte Ersparnis an RAM-Kapazität.

Kapitel 5

Funktionsweise kryptographischer Algorithmen

Diese Kapitel beschäftigt sich mit der Struktur und Funktionsweise kryptographischer Funktionen, sowie deren Bewertung in Hinsicht auf ihre Sicherheit und Tauglichkeit für ressourcenbeschränkte Systeme. Bei symmetrischen Blockchiffren wird dabei lediglich die Hinrichtung (Verschlüsselung) besprochen, da bei Verwendung des CFB oder OFB-Mode der Blockchiffren nur die Verschlüsselungsfunktion erforderlich ist. Auch für eine Anwendung in einem von CBC-MAC abgeleiteten MAC Verfahren wird lediglich die Verschlüsselungsfunktion einer Blockchiffre benötigt.

5.1 RC5

Der RC5 Algorithmus [59] ist eine symmetrische Blockchiffre, die 1994 von Ronald Rivest von der amerikanischen Firma RSA Security entwickelt wurde. RC5 zeichnet sich durch eine sehr einfache Beschreibung und die Verwendung von datenabhängigen Bit-Rotationen als wesentliche Verschlüsselungsfunktion aus. RC5 kann in Blockgröße w, Schlüsselgröße b und Rundenzahl r parametrisiert werden. Der üblicherweise verwendete Algorithmus mit Wortbreite $w = 32$ Bit, Schlüssellänge $b = 16$ Byte und Rundenzahl $r = 12$ wird als RC5-32/12/16 bezeichnet. Die Blockgröße des Algorithmus ist das Doppelte der Wortbreite.

5.1.1 Keyschedule

RC5-32/12/16 verwendet $2*12+2$ Rundenschüssel mit jeweils 32 Bit Breite. Da jeder Rundenschlüssel von jedem Anderen abhängt, ist es nicht möglich, die Rundenschlüssel während der Ver-, beziehungsweise Entschlüsselung zu

berechnen. Die Rundenschlüssel müssen vorab berechnet und abgespeichert werden. Bei der Verwendung des RC5-32/12/16 Algorithmus sind somit 104 Byte RAM zur Speicherung der Rundenschlüssel erforderlich.

5.1.2 Verschlüsselung

Der RC5 Algorithmus bearbeitet bei der Verschlüsselung zwei Worte A und B mit einer Breite von w Bit. Der Verschlüsselungsalgorithmus verwendet lediglich die Funktionen Addition modulo 2^w, XOR und ROL(a,x), also eine zyklische Links-Rotation eines Wortes a mit w Bit Breite um x Bit. Die Rundenschlüssel K_0 bis K_{2r+1} wurden vorab berechnet. Dem im Folgenden vorgestellten Algorithmus wird der zu verschlüsselnde Klartext in den beiden w Bit breiten Variablen A und B übergeben.

$A \leftarrow A + K_0 \bmod 2^w$
$B \leftarrow B + K_1 \bmod 2^w$
for $i = 1$ to r **do**
  $A := (\mathrm{ROL}(A \oplus B), B) + K_{2i} \bmod 2^w$
  $B := (\mathrm{ROL}(B \oplus A), A) + K_{2i+1} \bmod 2^w$
end for

Am Ende der Verschlüsselungsfunktion enthalten A und B die verschlüsselten Daten.

5.1.3 Sicherheit

RC5 wird als sicher angesehen, es existieren keine erfolgreichen Angriffe auf RC5-32/12/16. Für in der Rundenzahl reduzierte RC5-Variationen sind jedoch Angriffe bekannt [40]. Eine Implementierung von RC5 auf Systemen ohne Unterstützung für Registerrotationen um eine beliebige Anzahl von Bit kann jedoch zu einem Sicherheitsrisiko durch Timing-Angriffe führen [31]. Dies trifft für viele Mikrocontroller zu, da diese üblicherweise nur eine Rotation eines 8-Bit Registers um ein Bit erlauben, so dass die Anzahl der benötigten Takte einer Runde von RC5 direkte Rückschlüsse auf die Anzahl der Ein-Bit-Rotationen zulässt. Gegen diese Sicherheitslücke ist lediglich eine Verwendung von NOP-Operationen, zum Erreichen einer konstanten Laufzeit, denkbar. Da jedoch für einen Angriff nach dem Verfahren von [31] 2^{20} Chiffratblöcke (ca. 8,4 Megabyte) benötigt werden, ist eine Anwendung des Algorithmus ohne konstante Laufzeit in Geräten, die wenige Daten übermitteln oder häufig einen Schlüsselwechsel vornehmen, möglicherweise akzeptierbar.

5.1.4 Implementierung

Tabelle 5.1 zeigt die Leistungsdaten und den Ressourcenbedarf dreier RC5-32/12/16-Implementierungen auf Mikrocontrollern. Der ROM-Bedarf setzt

CPU	MIPS	ROM in Byte	RAM in Byte	Durchsatz in Byte/Sek	Setup in ms	Quelle
Atmel	4	402 + 598	80 (?)	9467	3.92	a)
Atmel	4	518 + 598	80 (?)	14545	3.92	a)
ATMega32	8	624 + 546	104	29090	6,36	b)

Tabelle 5.1: Leistungsdaten dreier RC5-Chiffren auf Mikrocontrollern
Quellen: a) [58], b) eigene Implementierung

sich aus der Codegröße für die Chiffrierfunktion und der Keyschedule-Funktion zusammen. Der RAM-Bedarf ergibt sich aus den Rundenschlüsseln und gegebenenfalls erforderlichen temporären Variablen und beinhaltet weder den Schlüssel noch den Puffer zur Übergabe der zu verschlüsselnden Daten. Damit ergibt sich ein zusätzlicher RAM-Bedarf von 16 Byte für den Schlüssel (sofern dieser nach Durchführung des keysetup noch benötigt wird) und 8 Byte für den Ein-/Ausgabepuffer. Die Chiffrier-Funktion der eigenen Implementierung ist hinsichtlich Geschwindigkeit optimiert, die Keysetup-Routine ist dagegen als nicht optimierter C-Code ausgeführt. Zur Implementierung von [58] ist nicht bekannt, in wieweit die erweiterten Rundenschlüssel, die 104 Bytes RAM erfordern, bereits in den angegebenen 80 Byte RAM enthalten sind.

5.1.5 Bewertung

RC5 ist durch seine einfache Runden-Funktion zur Implementierung auf einem Mikrocontroller geeignet. Allerdings kann die Größe des Rundenschlüssels, der vorab berechnet werden muss, bei stark ressourcenbeschränkten Mikrocontrollern möglicherweise nicht vollständig im RAM abgelegt werden. Andere Algorithmen wie beispielsweise AES erlauben die Berechnung des jeweiligen Rundenschlüssels innerhalb der entsprechenden Runde iterativ aus dem Rundenschlüssels der vorherigen Runde. Damit ist eine Wahl zwischen höherer Performance durch vorberechnete Rundenschlüssel und geringerem RAM-Bedarf durch Berechnung der Rundenschlüssel zur Laufzeit möglich. Auf Plattformen, deren RAM-Kapazität die Anforderung von RC5 nicht erfüllen, ist gegebenenfalls auch eine Reduktion der Rundenzahl und der damit erforderlichen Zahl der Rundenschlüssel in Erwägung zu ziehen.

Die 32-Bit Rotation, die auf 8-Bit Mikrocontrollern nicht direkt durchgeführt werden kann, kann mit einer Reihe von Maßnahmen beschleunigt werden. Die Rotation um k Bitstellen kann aufgeteilt werden in $m = k$ div 8 und $n = k$ mod 8. Die Rotationen um $8 * m$ Stellen können dann effizient mittels Byte-Vertauschungen durchgeführt werden. Lediglich die restlichen n Bitrotationen müssen mit Hilfe von vier verketteten 8-Bit Rotationen, die n-mal durchlaufen werden, durchgeführt werden.

Auf einer Architektur, die keine Addition mit Übertrag implementiert,

sinkt die Leistung des Algorithmus beträchtlich, da die ansonsten in einem Takt durchgeführte Addition mit Übertrag durch eine Addition, einen bedingten Sprung, sowie zwei Zuweisungen nachgebildet werden muss. Für diesbezüglich eingeschränkte Plattformen ist RC5 somit weniger geeignet.

5.1.6 RC6

Eine Weiterentwicklung von RC5 wurde unter der Bezeichnung RC6 als Kandidat für den AES Algorithmus eingereicht. Der Unterschied zu RC5 liegt vor Allem in der Verwendung einer zusätzlichen Integer-Multiplikation, um die Diffusion der Daten zu verstärken und damit die Anzahl der Runden zu senken. Weiterhin arbeitet der RC6 Algorithmus nicht mehr auf zwei Worten mit halber Blockgröße, sondern mit vier Worten, die jeweils ein Viertel der Blockgröße (128 Bit, wie alle AES Kandidaten) umfassen. Die Verwendung einer Integer-Multiplikation schließt jedoch eine Implementation auf vielen Mikrocontrollern aus, die keine Hardwaremultiplikation unterstützen. In [65] wird RC6 selbst auf 8-Bit Mikrocontrollern mit Multiplikationseinheit eine schlechte Performance im Vergleich mit anderen AES-Kandidaten bescheinigt.

5.2 AES - Rijndael

Das amerikanische Normen-Institut NIST (National Institut of Standards and Technology) schrieb 1997 die Entwicklung einer symmetrischen Blockchiffre als Nachfolger des DES-Algorithmus aus, der nicht mehr als sicher erachtet wurde. Anwendungsbereich des Advanced Encryption Standard (AES) sollte unter anderem, wie bereits bei DES, die Chiffrierung von nicht eingestuftem Regierungsmaterial sein. Aus den eingeschickten Algorithmen wurde im Jahr 2000 der Rijndael Algorithmus der belgischen Wissenschaftler Joan Daemen und Vincent Rijmen ausgewählt und 2001 als FIPS-197 (Federal Information Processing Standard) veröffentlicht.

5.2.1 Grundlagen des AES Algorithmus

AES ist eine symmetrische Blockchiffre, die auf Blöcken von 128 Bit arbeitet und einen Schlüssel mit 128, 192 oder 256 Bit Länge verwendet. Ursprünglich waren auch Blockgrößen von ebenfalls 192 und 256 Bit definiert, die aber nachträglich aus den Anforderungen für AES-Kandidaten entfernt wurden. AES verwendet keine Feistel-Struktur, wendet also die Rundenfunktion stets auf den gesamten Block an. Bei der Blockgröße von 128 Bit werden bei einem 128-Bit Schlüssel 10 Runden, bei einem 192-Bit Schlüssel 12 Runden und bei einem 256-Bit Schlüssel 14 Runden durchgeführt. Im Folgenden wird nur lediglich AES mit einem 128-Bit Schlüssel besprochen.

AES arbeitet auf jeweils 16 Byte, die in einer 4×4 Matrix angeordnet sind. Ein 16-Byte Block $m = a_0 a_1 ... a_{14} a_{15}$ wird auf folgende Matrix abgebildet:

$$\begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{pmatrix}$$

Die Matrix mit dem aktuellen Wert des zu chiffrierenden Klartextes oder Zwischenergebnisses wird als Zustand (State) bezeichnet. Der Schlüssel wird in derselben Weise in eine Matrixform überführt.

5.2.2 Algebraische Grundlage des AES Algorithmus

AES interpretiert ein Byte $a = a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$ als die Polynomdarstellung

$$f(x) = a_7 x^7 + a_6 x^6 + a_5 x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x^1 + a_0 x^0$$

Weiterhin wird der Körper GF(256) mit der Menge

$$M = \{f \epsilon Z_2[x] : grad(f) \leq 7\}$$

definiert, also der Menge der Polynome mit Grad kleiner oder gleich 7 und den Faktoren 0 und 1. Diese werden repräsentiert durch die Interpretation der binären Darstellung der äquivalenten Bytes als Polynomialdarstellung, also:

$$x^7 + x^6 + x^4 + x^3 = 1101100b = 220$$

Im Körper GF(256) wird die Addition zweier Elemente definiert als die XOR-Verknüpfung der equivalenten Bytes. Zusätzlich wird die Multiplikation zweier Elemente des Körpers definiert als Multiplikation der Elemente modulo eines Polynoms vom Grad 8. Dazu wird das irreduzible Polynom $m(x) = x^8 + x^4 + x^3 + x + 1$ fest gewählt [19].

Das neutrale Element der Addition ist das Polynom $f(x) = 0$ bzw. das Byte 0 und das neutrale Element der Multiplikation ist das Polynom $f(x) = 1$ bzw. das Byte 1. Weiterhin lassen sich mit Hilfe des erweiterten Euklidischen Algorithmus zueinander inverse Polynome finden, also die Polynome a(x) und b(x), so dass

$$\begin{aligned} a(x) * m(x) + b(x) * f(x) &= 1 \\ \Leftrightarrow b(x) * f(x) &= 1[m(x)] \\ \Leftrightarrow f^{-1}(x) &= b(x)[m(x)] \end{aligned}$$

Das Inverse des Polynoms $f(x) = 0$ ist definiert als $f(x) = 0$.

5.2.3 Details des Algorithmus

Der AES Algorithmus verwendet, ähnlich wie DES, eine SBox als nicht-lineare Transformation zum Schutz vor linearen Angriffen auf die Chiffre. Die Konstruktion der SBox besteht in der Verkettung zweier Transformationen. Zunächst wird die nicht-lineare Abbildung $b = inv(a)$, also die Berechnung des Inversen in GF(256), durchgeführt. Diese wird gefolgt von der affinen Abbildung

$$T: \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Die Verkettung $SBox(x) = T(inv(x))$ der beiden Transformationen ergibt eine Transformation ohne Fixpunkt $x = SBox(x)$ und ohne Anti-Fixpunkt $x = \overline{SBox(x)}$ [19]. Die SBox ist unabhängig vom Schlüssel und vom zu verschlüsselnden Klartext, so dass Sie vorab berechnet und als Tabelle mit 256 Einträgen fest im Programmcode hinterlegt werden kann.

Die Transformation SubBytes des AES-Algorithmus ersetzt jedes Element des Zustandes durch sein Ergebnis bei Anwendung der SBox. Beide Operationen der SBox – und damit auch die SBox selbst – sind invertierbar, so dass sich die Entschlüsselung durch Substitution mit dem entsprechenden Element der inversen SBox bewerkstelligen lässt.

Zusätzlich zur Transformation SubBytes wird die Funktion ShiftRows auf den Zustand angewandt. Die Transformation ShiftRows verschiebt die Elemente des Zustands in der zweiten Zeile zyklisch um ein Byte nach links, die der dritten Zeile um zwei und die der vierten Zeile um drei Byte. Shiftrows ist eine lineare Transformation, ihre Umkehrung ist offensichtlich. Die Transformation lässt sich wie folgt darstellen:

$$ShiftRows: \begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{pmatrix} \rightarrow \begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_5 & a_9 & a_{13} & a_1 \\ a_{10} & a_{14} & a_2 & a_6 \\ a_{15} & a_3 & a_7 & a_{11} \end{pmatrix}$$

Die dritte Operation, die auf den Zustand angewandt wird, ist die MixColumns Transformation. MixColumns multipliziert die Spalten der Zustands-

matrix mit der festen Matrix

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

MixColumns ist ebenfalls eine lineare Transformation. Sie ist invertierbar durch die spaltenweise Multiplikation mit der Matrix

$$\begin{pmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{pmatrix}$$

Die vierte Operation des AES-Algorithmus – AddRoundKey – ist die elementweise Addition mit dem Rundenschlüssel. Wie bereits im Abschnitt über die algorithmischen Grundlagen erläutert, entspricht die Addition in GF(256) einer XOR Verknüpfung.

AES verwendet $(r + 1)$ Rundenschlüssel $r_i = r_{i,0}r_{i,1}...r_{i,15}, i = [0..r]$ mit jeweils 16 Byte Länge. Diese werden aus dem ursprünglichen Schlüssel $k = k_0k_1...k_{15}$ wie folgt abgeleitet:

1. $r_{0,n} = k_n; c_1 = 1$

2. für $i > 0$:
   $r_{i,0} = SBox(r_{i-1,13})\ XOR\ c_i$
   $r_{i,1} = SBox(r_{i-1,14})$
   $r_{i,2} = SBox(r_{i-1,15})$
   $r_{i,3} = SBox(r_{i-1,12})$
   $c_{i+1} = c_i * 2$
   für $3 < j < 16$: $r_{i,j} = r_{i-1,j}\ XOR\ r_{i,j-4}$

Die Transformation AddRoundKey ist linear und kann durch erneute Addition invertiert werden. Die Multiplikation $c_{i+1} = c_i * 2$ ist eine Multiplikation in GF(256) und muss somit modulo des irreduziblen Polynoms betrachtet werden. Wie aus der Berechnungsvorschrift der Rundenschlüssel ersichtlich, kann der jeweilige Rundenschlüssel aus dem der vorangegangenen Runde berechnet werden, so dass eine Berechnung des Rundenschlüssels zur Laufzeit möglich ist. Dies erspart bei der Verwendung eines Schlüssels von 128 Bit Länge $16 * (11 - 1)$ Byte RAM gegenüber eines Algorithmus, der die Rundenschlüssel vorab berechnet.
Diese Transformationen werden, wie in Abbildung 5.1 gezeigt, auf den zu chiffrierenden Klartext angewandt.

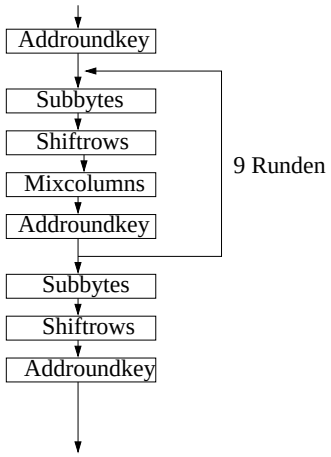

Abbildung 5.1: Struktur des AES-Chiffriervorgangs

5.2.4 Sicherheit

Gegen den AES-Algorithmus in voller Länge existiert derzeit kein erfolgreicher Angriff, wohl aber gegen einen schwächeren Algorithmus, der auf sieben Runden (für einen 128 Bit Schlüssel) verkürzt wurde [25]. Damit ist der Sicherheitfaktor nach [24] mit 1,11 für die AES-Variante mit 128 Bit Schlüssel der geringste aller AES-Kandidaten. Weiterhin ist für AES eine Darstellung der Chiffrierfunktion in einem überdefinierten Gleichungssystem bekannt, so dass möglicherweise ein zukünftiger Angriff möglich ist, sobald ein Verfahren entwickelt wurde, diese Darstellung effizient zu lösen [17]. Dennoch wird AES derzeit als sicher angesehen.

Allerdings muss bei der Verwendung von AES auf eine Unabhängigkeit der Schlüssel untereinander geachtet werden, da in [25] ein (wenn auch nicht praktikabler) related-key Angriff vorgestellt wurde. Sollen aus einem Hauptschlüssel mehrere Sitzungsschlüssel erzeugt werden, kann dies beispielsweise nach dem in 2.10 geschilderten Verfahren geschehen.

Wie im folgenden Abschnitt beschrieben wird, muss bei der Implementierung des Algorithmus auf eine konstante, nicht von den zu verschlüsselnden Daten abhängigen, Laufzeit geachtet werden, da ansonsten die Kenntnis der Laufzeit bei der Verschlüsselung vorgegebener Daten Hinweise auf den Schlüssel geben kann [45].

5.2.5 Implementierung

In Tabelle 5.2.5 werden Leistungsdaten einiger AES-Implementierungen auf Mikrocontrollern aufgezeigt. Dabei wird jeweils nur die Vorwärtsrichtung des Algorithmus betrachtet und eine Berechnung der Rundenschlüssels zur Laufzeit vorgenommen.

CPU	Zyklen	MIPS	Durchsatz in Byte/sec	Codegröße in Byte	RAM in Byte	Quelle
i8051 a	4065	3,3	13120	768	?	[19]
i8051 b	3744	3,3	14254	826	?	[19]
i8051 c	3168	3,3	16835	1016	?	[19]
68HC08	8390	8	15256	919	36	[19]
ATmega163	3847	8	33272	1326	8	[66]

Tabelle 5.2: Leistungsdaten mehrerer AES-Implementierungen

Die Implementierung aus [66] weist jedoch die im Folgenden diskutierte Anfälligkeit durch Timing-Angriffe in der Mixcolumns-Transformation auf, die sich mit 32 Byte zusätzlichem Programmcode und durchschnittlich 80 zusätzlichen Zyklen pro verschlüsseltem Block beheben lässt.

Die MixColumns Transformation und die Berechnung des Rundenschlüssels benötigen die üblicherweise als xtime bezeichnete Funktion, die $f(x) = 2x$ im Körper GF(256) berechnet. Dies lässt sich mit einer bitweisen Linksverschiebung, sowie einer Addition des irreduziblen Polynoms 0x1b im Falle eines Überlaufs bei der Linksverschiebung implementieren. Falls jedoch der Code

$x \leftarrow x << 1$
if carry **then**
  $x \leftarrow x \oplus 0x1b$
end if

verwendet wird, ist die Laufzeit des Algorithmus abhängig vom höchstwertigen Bit von x vor der Rotation. Dies kann zu einer Kenntnis innerer Zustände des Chiffrieralgorithmus führen. In [45] wird ein Angriff diskutiert, der mit Hilfe von 3000 Chosen-Plaintext Messungen pro Byte des Schlüssels diesen mit hoher Wahrscheinlichkeit errechnen kann. Daher sollte der Algorithmus wie folgt umgeschrieben werden:

$x \leftarrow x << 1$
if carry **then**
  $P \leftarrow 0x1b$
else
  $P \leftarrow 0x00$
end if
$x \leftarrow x \oplus P$

Damit ist die Laufzeit des Algorithmus stets konstant. Diese Implementierung erfordert zwei Sprünge, die üblicherweise mehrere CPU-Takte benötigen. Alternativ zu dieser Implementierung nennt [64] für CPUs, die über einen Subtract-with-Carry Operator verfügen, folgende schnellere Alternative (Darstellung in Atmel AVR Assembler Opcodes):

```
LSL R1
MOV R16,R1
SBC R16,R16
ANDI R16,0x1b
EOR R1,R16
```

LSL führt die Linksverschiebung um ein Bit durch. Das zuvor höchstwertige Bit des Eingaberegisters R1 wird im Carry-Bit des Statusregisters abgelegt. Danach wird das Ergebnis der Linksverschiebung in das temporäre Register R16 kopiert und dort von sich selbst, zuzüglich des Carry-Bit, abgezogen. Wenn das Carry-Bit nicht gesetzt war ist das Ergebnis der Subtraktion R16=R16-R16=0. Wenn das Carry-Bit gesetzt war, ist das Ergebnis R16=R16-(1+R16)=255. Das Ergebnis der Subtraktion wird mit der Repräsentation des irreduziblen Polynoms mit der logischen UND-Funktion verknüpft. Damit enthält das Register R16 für den Fall, dass das Carry-Bit gesetzt war den Wert 0x1b, andernfalls den Wert 0x00. Dieser wird dann wie gefordert durch XOR mit dem Ergebnis der Linksverschiebung R1 verknüpft und ergibt das Ergebnis der Funktion $f(x) = 2x$ in GF(256).

5.2.6 Bewertung

Der AES Algorithmus eignet sich insbesondere aus drei Gründen für eine Implementierung auf einem 8-Bit Mikrocontroller. Erstens verwendet er als grundlegende Datenstruktur ein Byte, im Gegensatz zu vielen anderen Algorithmen die 32-Bit Worte verwenden. Dadurch entfallen die in vielen anderen Algorithmen erforderlichen Überträge bei Additionen, Multiplikationen und Rotationen.

Zweitens werden, falls die SBox vorab berechnet wurde, überwiegend XOR und Additionen als Operationen verwendet. Die Schiebevorgänge der Shiftrows-Transformationen sind immer Verschiebungen um ein Vielfaches von 8 Bit, so dass sie sich durch Kopiervorgänge, beziehungsweise durch einen Offset bei der Adressierung ersetzen lassen.

Drittens kann der Rundenschlüssel während einer Runde aus dem Schlüssel der vorherigen Runde berechnet werden, so dass eine Berechnung des Rundenschlüssels während des Ver-, beziehungsweise Entschlüsselungsvorganges durchgeführt werden kann. Dies ermöglicht die Implementierung auf Mikrocontrollern mit sehr kleinem Hauptspeicher, der nicht alle Rundenschlüssel gleichzeitig aufnehmen kann.

Eine Implementierung, die den Großteil der verwendeten Variablen in Prozessorregistern ablegt, ist auf einem Prozessor mit Atmel AVR Architektur möglich. Zur Speicherung des Zustands werden 16 Register benötigt. Zusätzlich werden ein Minimum von zwei temporären Registern und ein Register als Rundenzähler benötigt. Falls ein vorab berechneter Rundenschlüssel verwendet wird, wird noch ein Zeiger auf den jeweiligen Rundenschlüssel benötigt. Wenn der Rundenschlüssel iterativ berechnet wird, werden statt

dessen 16 Byte zur Speicherung des aktuellen Rundenschlüssels, sowie eine zusätzliche Variable benötigt.

5.3 IDEA

Der IDEA-Algorithmus ist eine Entwicklung der Kryptologen Xuejia Lai und James L. Massey aus dem Jahr 1991 und ist vom Schweizer Unternehmen Ascom patentiert. Eine Verwendung in nicht-kommerziellen Projekten ist jedoch ohne Lizenz erlaubt.

IDEA verwendet einen 128-Bit Schlüssel und arbeitet auf 64 Bit grossen Datenblöcken. Der IDEA-Algorithmus verwendet die drei zueinander inkompatiblen algebraischen Gruppen

$(GF(2^{16}), XOR)$
$(GF(2^{16}), +)$
$(GF(2^{16}+1), *_{IDEA})$

Dabei wird in der modifizierte Multiplikation $*_{IDEA}$ der Faktor 2^{16} als 0 interpretiert. Durch die Verwendung von drei zueinander inkompatiblen Gruppen ist es möglich, die Rundenzahl auf acht zu reduzieren. Dadurch wird die Geschwindigkeit des Algorithmus im direkten Vergleich zu konkurrierenden Algorithmen stark erhöht. IDEA verwendet ebenfalls iterativ berechenbare Rundenschlüssel. Abbildung 5.2 zeigt die Struktur des IDEA-Verschlüsselungsvorgangs. IDEA verwendet neun Rundenschlüssel mit jeweils 6*16 Bit Breite. Vom letzten Rundenschlüssel K9 werden lediglich 4*16 Bit verwendet.

5.3.1 Sicherheit

Der derzeit beste Angriff [8] auf IDEA ist auf einen von den üblicherweise 8,5 auf 4,5 Runden reduzierten Algorithmus beschränkt und erfordert zudem 2^{64} Klartext-Chiffrat-Paare, sowie einen Aufwand von 2^{112}. Somit verfügt IDEA derzeit noch über einen grossen Sicherheitsfaktor. Allerdings wurden mehrere teilweise sehr grosse Klassen schwacher Schlüssel identifiziert [12], die gegebenenfalls die Sicherheit des Algorithmus herabsetzen können. Die Wahrscheinlichkeit, einen Schlüssel selbst der größten Klasse zu wählen liegt jedoch bei 2^{-64} und ist somit vernachlässigbar gering.

5.3.2 Implementierung

Wie bereits erwähnt, verwendet IDEA 16-Bit Multiplikationen, die auf einer 8-Bit Plattform mittels vier 8-Bit Multiplikationen, sowie sechs Additionen (vier davon mit Übertrag) durchgeführt werden müssen. Eine Implementierung auf einer Plattform, die nicht über eine Hardwaremultiplika-

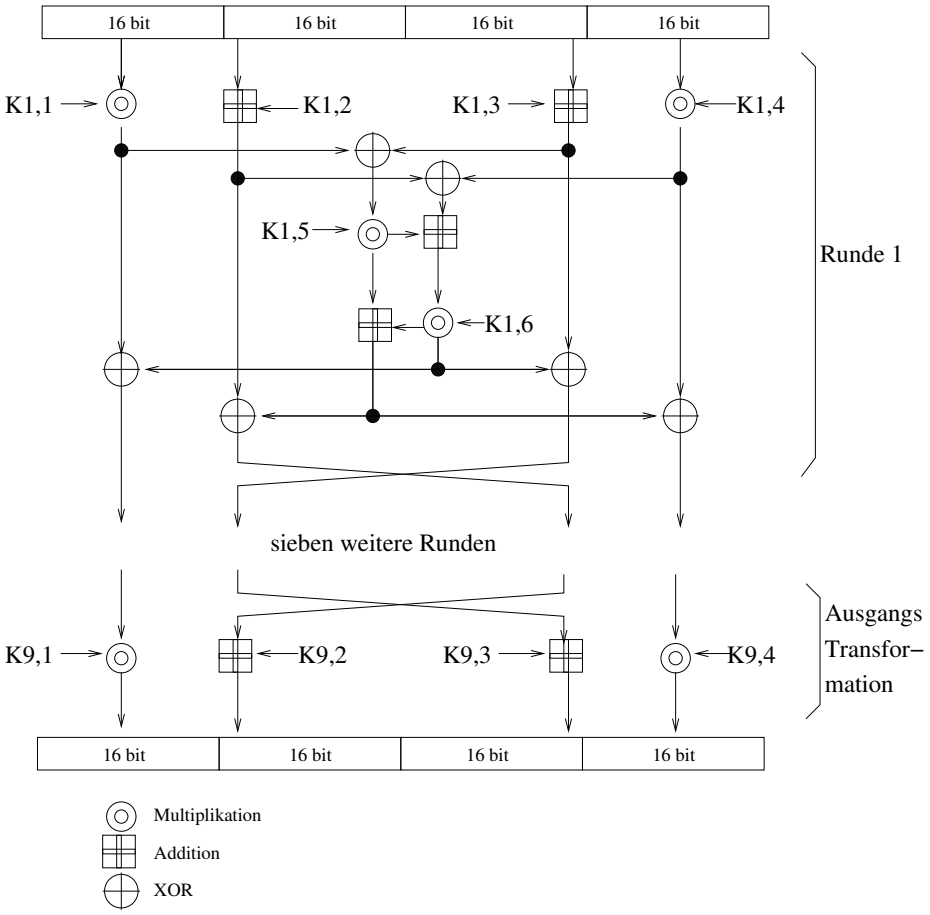

Abbildung 5.2: Struktur des IDEA-Chiffriervorgangs

tion verfügt ist sinnlos, da für die Verschlüsselung eines Blocks mit einer Größe von 64 Bit 136 8-Bit Multiplikationen durchgeführt werden müssen. Auf Mikrocontrollern, die über eine 8-Bit Multiplikation verfügen, erreicht IDEA dagegen einen sehr hohen Durchsatz. Tabelle 5.3 zeigt die Werte einer IDEA-Implementierung auf einem Atmel-Mikrocontroller. Die Implementierung verwendet einen vorberechneten Rundenschlüssel und ist in nicht optimiertem C-Code mit einer Assembler Implementierung der 16-Bit Multiplikation ausgeführt.

5.3.3 Bewertung

IDEA eignet sich aufgrund der Verwendung der Multiplikation nur für Plattformen, die über eine Hardwaremultiplikation verfügen. Dies sind die im Kapitel 4 der Klasse II zugeordneten Geräte. Auf diesen erreicht IDEA jedoch aufgrund der geringen Rundenzahl einen guten Durchsatz. Gleichzeitig hat

CPU	MIPS	ROM	RAM	Durchsatz	Key-Setup
ATMega32	8	824 Byte	104 Byte	22998 Byte/Sek	0,38 ms

Tabelle 5.3: Leistungsdaten einer IDEA-Implementierung
Quelle: eigene Implementierung

IDEA einen geringen RAM-Bedarf aufgrund der Möglichkeit, die Rundenschlüssel iterativ zu berechnen. IDEA profitiert stark vom Vorhandensein eines add-with-carry Befehls, da als grundsätzliche Datenstruktur ein 16-Bit Wort verwendet wird und die 16-Bit Addition häufig im Algorithmus benötigt wird. Unter Verwendung eines 8-Bit add-with-carry lässt sich die 16-Bit Addition mit nur zwei Opcodes durchführen.

Auf Systemen mit geringen Anforderungen an die Sicherheit ist gegebenenfalls auch ein auf 5,5 Runden reduzierter Algorithmus denkbar, da lediglich der auf 4,5 Runden verkürzte Algorithmus bislang gebrochen wurde.

5.4 Skipjack

Der Skipjack Algorithmus wurde ursprünglich für den Clipper-Chip entwickelt, einem Projekt der U.S.-Regierung zur Schaffung eines Standards für verschlüsselte Kommunikation für Firmen und Privatpersonen. Skipjack verwendet vergleichsweise einfache Operationen und ist dadurch besonders für eine Implementierung in Hardware geeignet. Skipjack verwendet einen 80-Bit Schlüssel, der nicht zu Rundenschlüsseln erweitert wird. Dadurch ist der RAM-Bedarf sehr gering. Neben einer SBox mit 256 Einträgen werden lediglich 8-Bit Additionen und XOR Operationen verwendet. Allerdings benötigt Skipjack 32 Runden. Die 32 Runden sind in vier Gruppen mit jeweils

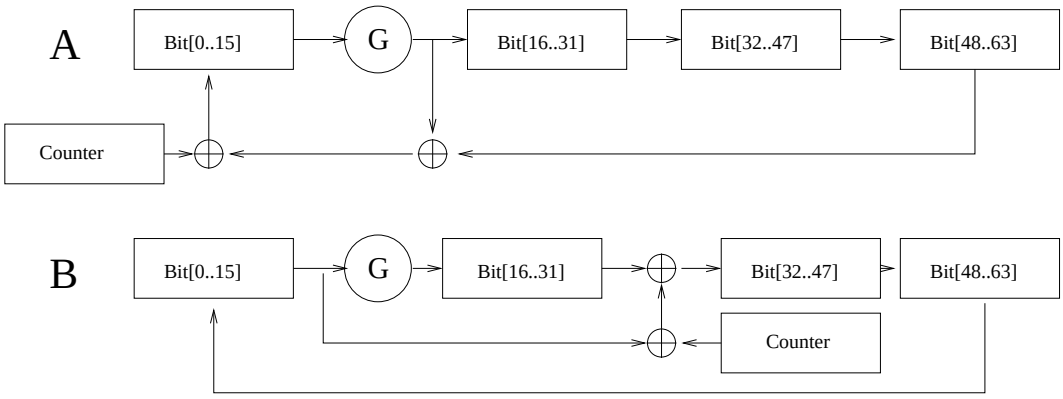

Abbildung 5.3: A und B Rundenfunktionen des Skipjack Algorithmus [1]

acht Runden aufgeteilt und führen jeweils abwechselnd acht mal die Rundenfunktion A beziehungsweise B aus. Beide Rundenfunktionen sind linear rückgekoppelte Schieberegister auf Wort-Basis mit 16-Bit breiten Datenworten. Die Struktur der Rundenfunktionen ist in Abbildung 5.3 aufgezeigt. Die Rundenfunktionen verwenden einen Zähler, der mit 0 beginnend nach jeder Runde um den Wert 1 erhöht wird und die Funktion G, die ein 16-Bit Wort in ein 16-Bit Wort überführt und deren Struktur in Abbildung 5.4 angegeben ist. In der Funktion G wird wiederum auf die Funktion F verwiesen. Diese ist eine fest vorgegebene SBox, die eine Abbildung eines Bytes auf ein Byte vornimmt. Weiterhin verweist die G-Funktion auf vier Byte des zehn Byte grossen Schlüssels, wobei jede Runde zyklisch auf die jeweils vier folgenden Byte des Schlüssels zugreift.

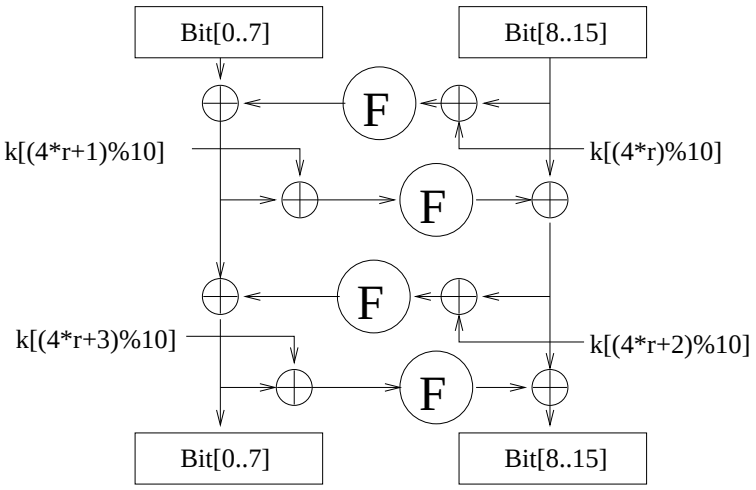

Abbildung 5.4: G-Funktion des Skipjack Algorithmus [1]

5.4.1 Sicherheit

Skipjack mit 32 Runden ist bislang ungebrochen, allerdings ist der Algorithmus aufgrund der Beschränkung auf 80-Bit Schlüssellänge für hohe Sicherheitsanforderungen durch die Möglichkeit, den Schlüsselraum vollständig zu testen, nicht geeignet. Zudem wurden erfolgreich Angriffe gegen minimal vereinfachte Skipjack Algorithmen mit 31 anstelle von 32 Runden oder einem Algorithmus der insgesamt 317 anstelle von 320 XOR Funktionen verwendet durchgeführt [11, 10]. Die Anwendung ist daher lediglich in Bereichen mit niedrigeren Sicherheitsanforderungen denkbar.

CPU	MIPS	ROM	RAM	Durchsatz	Quelle
PIC16F84A	5	403 Byte	2 Byte	14837 Byte/Sek	a)
ATMega32	8	708 Byte	0 Byte	23391 Byte/Sek	b)

Tabelle 5.4: Leistungsdaten zweier Skipjack-Chiffren auf Mikrocontrollern
Quellen: a) [69], b) eigene Implementierung

5.4.2 Implementierung

Bei einer Implementierung der Skipjack-Chiffre auf der Atmel AVR Architektur ist es möglich, alle in der Rundenfunktion benötigten Variablen in Prozessorregistern abzulegen. Da ein Zugriff auf ein Prozessorregister deutlich schneller als ein Zugriff auf den Hauptspeicher durchgeführt werden kann, kann somit ein höherer Durchsatz erreicht werden. Tabelle 5.4 zeigt die Leistungsdaten zweier Implementierungen der Skipjack-Chiffre auf Mikrocontrollern.

Die RAM-Angaben verstehen sich zuzüglich 8 Byte Datenpuffer zur Speicherung der zu verschlüsselnden Daten (die Chiffrierung der Daten erfolgt in-place), sowie 10 Byte zur Speicherung des Schlüssels.

5.4.3 Bewertung

Als Chiffre, die neben Additionen von Bytes lediglich 8-Bit XOR und eine 8-Bit Tabelle verwendet, eignet sich Skipjack für die Implementierung auf stark ressourcenbeschränkten Systemen oder in Hardware. Insbesondere der geringe RAM-Bedarf durch die nicht vorhandenen Rundenschlüssel kann auf Plattformen mit wenig RAM die Skipjack-Chiffre attraktiv erscheinen lassen. Allerdings ist die Leistungsfähigkeit der Chiffre gering. Trotz der einfachen Rundenfunktion ist Skipjack bezüglich des maximalen Durchsatzes einer auch nur mäßig optimierten IDEA oder AES Chiffre, die zudem beide einen 128-Bit anstelle eines 80-Bit Schlüssels verwenden, deutlich unterlegen. Somit ist Skipjack nur attraktiv, wenn der geringe RAM- oder ROM-Bedarf der Implementierung von entscheidender Bedeutung ist.

5.5 RC4

Bei RC4 handelt es sich um eine symmetrische Stromchiffre, die 1987 von Ron Rivest entwickelt wurde. Die Funktionsweise des RC4-Algorithmus wurde bis 1994 erfolgreich geheimgehalten, bis in einer Internet-Mailingliste ein Algorithmus anonym veröffentlicht wurde, der einen zu RC4 identischen Schlüsselstrom erzeugt. Da aus rechtlichen Gründen die Bezeichnung RC4 in eigenen Implementierungen nicht verwendet werden darf und bis heute nicht bestätigt ist, dass es sich bei dem veröffentlichten Algorithmus tatsächlich um RC4 handelt, wird häufig von ARC4 (alleged RC4) gesprochen. RC4

wird unter anderem als Verschlüsselungsalgorithmus im SSL (secure socket layer) Protokoll eingesetzt.

RC4 verwendet einen Schlüssel variabler Länge zwischen 8 und 2048 Bit, der den internen Zustand von 256 Byte Länge initialisiert. RC4 benötigt pro Byte zu chiffrierenden Daten deutlich weniger Operationen als die vorgestellten Blockchiffren und ist somit deutlich schneller. Durch die sehr einfache Beschreibung ist der Algorithmus zudem sehr kompakt. Der interne Zustand von 258 Byte muss jedoch im RAM gehalten werden, da er sich während der Generierung des Schlüsselstroms ändert. Dadurch ist der RAM-Bedarf von RC4 deutlich höher als bei allen anderen vorgestellten Algorithmen.
Der RC4 Algorithmus wird wie folgt initialisiert. Die Permutationstabelle S besitzt 256 Einträge, der Schlüssel K hat eine Länge von l Byte.

for $i = 0$ to 255 **do**
  $S[i] \leftarrow i$
end for
$j \leftarrow 0$
for $i = 0$ to 255 **do**
  $j \leftarrow (j + S[i] + K[i \bmod l]) \bmod 256$
  vertausche $S[i]$ und $S[j]$
end for

Die Generierung eines Byte des Schlüsselstroms besteht aus folgenden Operationen.

$i \leftarrow (i + 1) \bmod 256$
$j \leftarrow (k + S[i]) \bmod 256$
vertausche $S[i]$ und $S[j]$
Ausgabe von $S[S[i] + S[j] \bmod 256]$

Die Variablen i und j werden vor der Generierung des ersten Byte des Schlüsselstroms auf den Wert 0 initialisiert. Wie bei allen additiven Stromchiffren erfolgt die Chiffrierung eines Klartext-Bytes durch XOR-Verknüpfung mit dem entsprechenden Byte des Schlüsselstroms und die Dechiffrierung durch die nochmalige XOR-Verknüpfung von Chiffratstrom und Schlüsselstrom.

5.5.1 Sicherheit

RC4 gilt als kryptographisch sicher wenn die drei folgenden Vorbedingungen erfüllt sind:

- Aufgrund von Schwächen im Keyscheduling-Algorithmus wird von Mironov in [54] empfohlen, die ersten 512 Byte des Schlüsselstroms nicht zu verwenden.

- Wie bei allen synchronen Stromchiffren darf auch bei RC4 der Schlüsselstrom nicht mehrfach verwendet werden. Um dennoch einen Schlüs-

CPU	MIPS	ROM	RAM	Durchsatz
ATMega32	8	332 Byte	258 Byte	98765 Byte/Sek

Tabelle 5.5: Leistungsdaten einer RC4 Implementierung
Quelle: eigene Implementierung

sel mehrfach verwenden zu können, kann ein zufälliger und öffentlicher Initalisierungsvektor verwendet werden, der, angehängt an den eigentlichen Schlüssel, stets unterschiedliche Schlüsselströme gewährleistet. Dieses Vorgehen ist jedoch [26] zufolge unsicher, da mit Hilfe abgefangener Chiffrat-Blöcke der eigentliche Schlüssel errechnet werden kann, wenn einem Angreifer der Initialisierungsvektor bekannt ist. Um diesen Angriff ausschließen zu können rät [26] dazu, eine kryptographische Hashfunktion zu verwenden, die aus Schlüssel und Initialisierungsvektor den Schlüssel der Stromchiffre generiert. Dies dient auch dazu, die Unsicherheit durch die in [26] aufgezeigten schwachen Schlüssel zu beseitigen.

- Weiterhin raten Fluhrer und McGrew in [67] davon ab, mehr als $2^{30.6}$ Byte des Schlüsselstroms zu verwenden.

5.5.2 Implementierung

Tabelle 5.5 zeigt die Daten einer eigenen RC4 Implementierung. Es handelt sich um eine nicht optimierte, vollständig in C programmierte Version, die dennoch einen deutlich höheren Durchsatz erzielt als die vorgestellten Blockchiffren.

5.5.3 Bewertung

Als Stromchiffre arbeitet RC4 nicht mit Datenblöcken, sondern mit einzelnen Bytes. Zur Verschlüsselung eines Bytes werden lediglich drei Additionen zweier Bytes, drei Zugriffe auf die Zustandstabelle, sowie ein XOR benötigt. Der innere Zustand der Chiffre benötigt 258 Byte RAM, die während des gesamten Nutzungszeitraums eines Schlüssels reserviert bleiben müssen. Aufgrund seiner hohen Geschwindigkeit und des kompakten Codes scheint RC4 für ressourcenschwache Umgebungen sehr gut geeignet, sofern die Anforderung von 258 Byte RAM erfüllt werden kann. Allerdings kann RC4 als Stromchiffre nicht in einem CBC-MAC- Verfahren verwendet werden. Damit ist zusätzlich eine beliebige Blockchiffre oder ein kryptographisches Hashverfahren zur Generierung einer CBC-MAC oder HMAC erforderlich. Der hierzu erforderliche Mehraufwand macht zumindest die Eigenschaft des geringen ROM-Bedarfs zunichte. Da die verwendete Hashfunktion beziehungsweise

Blockchiffre zur Generierung eines MAC auf die gesamte Nachricht angewandt werden muss, begrenzt die Geschwindigkeit des MAC Algorithmus die des Gesamtsystems soweit, dass auch der Geschwindigkeitsvorteil von RC4 weniger stark ausgenutzt werden kann.

5.6 RIPEMD-128

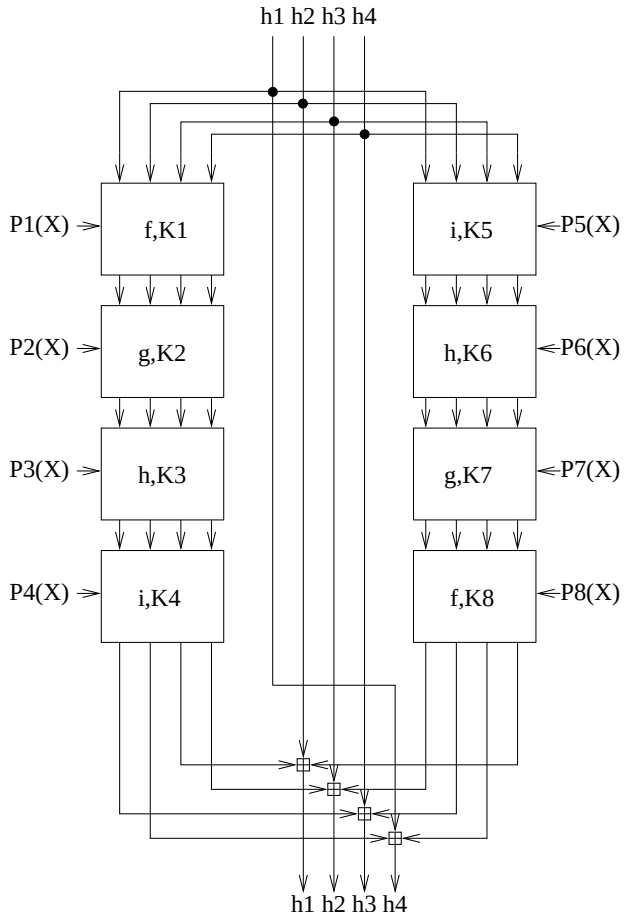

Abbildung 5.5: Struktur des RIPEMD-128 Algorithmus

Der Hashalgorithmus RIPEMD [20] entstand zwischen 1988 und 1992 als Teil des EU-Projektes RIPE (Race Integrity Primitives Evaluation). Der Algorithmus, der einen Hashwert mit einer Länge von 128 Bit berechnet, wurde 1996 zu einer 160-Bit Variante mit der Bezeichnung RIPEMD-160 erweitert und zusätzlich die 128-Bit Version des verbesserten Algorithmus unter der Bezeichnung RIPEMD-128 vorgestellt. Alle drei Verfahren basieren auf dem MD4-Verfahren von Ron Rivest.

Das RIPEMD-128 Verfahren bildet einen Block von 512 Bit Größe auf einen 128-Bit Wert ab. Der Eingabeblock ist in 16 Worte X_0 bis X_{15} mit 32 Bit Länge aufgeteilt. Die in Abbildung 5.5 dargestellte Struktur des Algorithmus zeigt die zwei Zweige in denen jeweils in vier Runden der Eingabeblock zusammen mit der übergebenen initialen Belegung des Hashwertes auf den neuen Hashwert abgebildet wird. In jeder Runde wird für alle sechzehn 32 Bit breiten Worte des Eingabeblocks die Abbildung $T = \mathrm{ROL}((A + f_r(B, C, D) + X_{P_r(i)} + K_r), R_r(i))$ durchgeführt. Dazu werden die Funktionen g, h, i, j wie folgt definiert:

$$\begin{aligned} g(x, y, z) &= x \oplus y \oplus z \\ h(x, y, z) &= (x \wedge y) \vee (\neg x \wedge z) \\ i(x, y, z) &= (x \vee \neg y) \oplus z \\ j(x, y, z) &= (x \wedge y) \vee (y \wedge \neg z) \end{aligned}$$

Zusätzlich sind die beiden Permutationen π und p, sowie die Schrittweite der zyklischen Linksverschiebung ROL $R_r(i)$ fest vorgegeben. In den einzelnen Runden wird die oben angegebene Abbildungsvorschrift mit den Parametern

linker Zweig

Runde 1	Runde 2	Runde 3	Runde 4
$P_1(i) = id(i)$	$P_2(i) = p(i)$	$P_3(i) = p^2(i)$	$P_4(i) = p^3(i)$
$f_1 = g$	$f_2 = h$	$f_3 = i$	$f_4 = j$
$K_1 = 0$	$K_2 = 2^{30}\sqrt{2}$	$K_3 = 2^{30}\sqrt{3}$	$K_4 = 2^{30}\sqrt{5}$

rechter Zweig

Runde 1	Runde 2	Runde 3	Runde 4
$P_1(i) = \pi(i)$	$P_2(i) = p\pi(i)$	$P_3(i) = p^2\pi(i)$	$P_4(i) = p^3\pi(i)$
$f_1 = j$	$f_2 = i$	$f_3 = h$	$f_4 = g$
$K_1 = 2^{30}\sqrt[3]{2}$	$K_2 = 2^{30}\sqrt[3]{3}$	$K_3 = 2^{30}\sqrt[3]{5}$	$K_4 = 0$

auf die Eingabedaten und die Verkettungsvariablen angewandt. Nach Berechnung von T werden die vier jeweils 32 Bit breiten Verkettungsvariablen A, B, C, D abgebildet auf $(A, B, C, D) \leftarrow (D, T, B, \mathrm{ROL}(C, 10))$. Nach Durchlaufen der vier Runden mit jeweils 16 Schritten werden die Ergebnisse der beiden Zweige wie in Abbildung 5.5 gezeigt verknüpft.

Bevor die Nachricht in Blöcke aufgeteilt wird, muss sie auf eine definierte Länge erweitert werden. Dazu wird an die eigentliche Nachricht ein Bit mit dem Wert 1 angehängt, gefolgt von Bits mit dem Wert 0, so dass die Nachrichtenlänge (n*512)+448 Bit beträgt. An diese Nachricht wird die 64 Bit breite Länge der Nachricht in Bit modulo 2^{64} eingetragen, so dass eine Nachricht immer ein Länge von (n*512) Bit aufweist.

CPU	MIPS	ROM	RAM	Durchsatz
ATMega32	8	2180 Byte	100 Byte	3293 Byte/Sek

Tabelle 5.6: Leistungsdaten der RIPEMD-128 Hashfunktion

5.6.1 Sicherheit

Der RIPEMD-128 Algorithmus erfüllt die Anforderungen bezüglich preimage resistance und secondary preimage resistance. Derzeit sind keine Möglichkeiten bekannt, Kollisionen zu erzeugen. Die Größe des Hashwerts von 128 Bit kann jedoch in Umgebungen mit hohen Sicherheitsanforderungen oder mit vielen Nachrichten nicht mehr ausreichend sein, da aufgrund der Birthday Attack bereits bei durchschnittlich 2^{64} Hashwerten eine Kollision auftritt [53]. Diese Anzahl von Nachrichten wird in Embedded Systemen jedoch kaum erreicht, so dass in diesen Anwendungen ein Hashwert mit einem Wertebereich von 2^{128} ausreichend sein müsste.

5.6.2 Implementierung

Die Geschwindigkeit des für 32-Bit CPUs optimierten und mit zwei unabhängigen Zweigen für superskalare CPUs gedachten RIPEMD Algorithmus ist auf ressourcenbeschränkten 8-Bit Systemen gering. Zudem ist der Programmcode sehr umfangreich, vor allem da die notwendigen 32-Bit Additionen und Rotationen mittels 8-Bit Befehlen implementiert werden müssen. Tabelle 5.6 zeigt die Daten einer eigenen Implementierung des RIPEMD-128 Algorithmus auf der Atmel AVR Architektur. Es handelt sich um C-Code ohne Assembler-Optimierungen.

5.6.3 Bewertung

Die Verwendung zweier voneinander unabhängigen Zweigen dient zur Parallelisierung des Algorithmus auf Mehrprozessor-Systemen oder CPUs mit superskalarer Architektur. Da dieser Vorteil auf Embedded Systemen nicht zum Tragen kommt und die Verknüpfung der Ergebnisse der beiden Zweige einen Zusatzaufwand mit sich bringt, ist der RIPEMD-128 Algorithmus anderen Algorithmen, die lediglich einen Zweig verwenden, auf Embedded Systemen unterlegen. Wenn ein Hashalgorithmus erforderlich sein sollte, ist ein anderes Verfahren vorzuziehen.

5.7 SHA-1

Der Hashalgorithmus SHA-1 (Secure Hash Algorithm) basiert wie RIPEMD-128 auf MD4. Der Algorithmus arbeitet ebenfalls mit 512 Bit grossen Datenblöcken und erzeugt einen 160 Bit grossen Hashwert. SHA-1 hat eine deutlich

Runde	f(u, v, w)
1	((u AND v) OR (NEG(u) AND w)) + 0x5a827999
2	(u XOR v XOR w) + $0x6ed9eba1$
3	((u AND v) OR (u AND w) OR (v AND w)) + $0x8f1bbcdc$
4	(u XOR V XOR W) + $0xca62c1d6$

Tabelle 5.7: Rundenfunktionen von SHA-1. Die Funktionen AND, OR, XOR und NEG sind jeweils bitweise zu betrachten

einfachere Struktur im Vergleich zu RIPEMD-128. Der 64 Byte grosse Datenblock X wird in 16 Worte mit 32 Bit Breite aufgeteilt. Diese werden dann auf 80 Worte M derselben Breite erweitert nach der Vorschrift:

for $i = 0$ to 15 **do**
  $M[i] \leftarrow X[i]$
end for
for $i = 16$ to 79 **do**
  $M[i] \leftarrow$ROL$(M[i-16] \oplus M[i-14] \oplus M[i-8] \oplus M[i-3], 1)$
end for

ROL(x,i) ist die zyklische Linksrotation eines 32-Bit Wortes um i Stellen. Für das Berechnen eines Hashwertes werden vier Runden mit jeweils 20 Schritten durchlaufen. Dazu werden vier Rundenfunktionen (siehe Tabelle 5.7) definiert. Jeder Schritt führt dann mit der entsprechenden Rundenfunktion f(x,y,z) die Abbildungsvorschrift

$$(A, B, C, D, E) \leftarrow (ROL(A, 5) + f(B, C, D) + E + M[i], A, ROL(B, 30), C, D)$$

durch, wobei der Zähler i nach jedem Schritt um den Wert 1 erhöht wird. Die jeweils 32 Bit breiten Verkettungsvariablen A bis E ergeben zusammen den 160-Bit breiten Hashwert und werden vor dem Verarbeiten des ersten Blocks mit dem Vektor (0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476, 0xc3d2e1f0) initialisiert.

5.7.1 Sicherheit

SHA-1 wird als sicher erachtet. Derzeit sind keine Schwachstellen bekannt die zu Kollisionen führen können. Weiterhin ist aufgrund des größeren Wertebereichs des Hashwertes von 2^{160}, anstelle von 2^{128}, auch eine Kollision zwischen zwei beliebigen Nachrichten sehr unwahrscheinlich. Kritisch wird jedoch angesehen dass die Designgrundsätze von SHA-1 und die Unsicherheit die im Vorgängeralgorithmus SHA letztlich zur Weiterentwicklung von SHA zu SHA-1 führte nicht bekannt sind [63].

CPU	MIPS	ROM	RAM	Durchsatz
ATMega32	8	1670 Byte	152 Byte	7620 Byte/Sek

Tabelle 5.8: Leistungsdaten der SHA-1 Hashfunktion

5.7.2 Implementierung

SHA-1 verwendet in jeder der 80 Runden sieben 32-Bit Additionen. Damit ist eine effiziente 8-Bit Implementierung lediglich auf Architekturen mit add-with-carry Befehl möglich. Aufgrund der grossen Menge von Operationen und der fünf 32-Bit Verkettungsvariablen, ist eine Implementierung, die alle Rundendaten in CPU-Registern ablegt, kaum möglich. Durch die vielen Speicherzugriffe ist somit der Durchsatz des Algorithmus beschränkt. SHA-1 verwendet 32-Bit Linksrotationen um eine und fünf Stellen, sowie eine Rechtsrotation um zwei Stellen. Da diese auf 8-Bit CPUs nicht in einfacher Weise implementiert werden können, erhöht sich die Komplexität der Algorithmen weiter. Tabelle 5.8 zeigt die Daten einer eigenen Implementierung von SHA-1 auf der Atmel AVR Plattform. Obwohl die Implementierung, wie auch die RIPEMD-128 Implementierung, in nicht optimierten C-Code gehalten ist erreicht SHA-1 mehr als den doppelten Durchsatz im Vergleich zu RIPEMD-128. Allerdings benötigt SHA-1 aufgrund der internen Aufweitung der Datenworte mehr RAM. Auch bei SHA-1 ist der Durchsatz deutlich niedriger als bei Blockchiffren.

5.7.3 Bewertung

Durch den größeren Hashwert von 160 Bit im Vergleich zu den 128 Bit des RIPEMD-128 Verfahrens und der Erweiterung der Datenworte (die sich allerdings auch mit nur 16 zusätzlichen Worten implementieren lässt), benötigt SHA-1 mehr RAM im Vergleich zu RIPEMD-128. Allerdings sind aufgrund der einfacheren Struktur und dem Verzicht auf Permutationstabellen deutliche Einsparungen bei der Codegröße möglich. SHA-1 ist gegenüber Blockchiffren deutlich langsamer aber zumindest besser für ressourcenbeschränkte Plattformen geeignet als RIPEMD-128.

5.8 MD2

MD2 ist ein 1992 von RSA Data Security entwickelter kryptographischer Hashalgorithmus mit einer Ausgabe von 128 Bit Breite [36]. MD2 unterscheidet sich von den beiden anderen Hashalgorithmen durch die Verwendung einzelner Bytes als grundlegende Datenstruktur, anstelle der bei SHA-1 und RIPEMD-128 verwendeten 32-Bit Worte. Dadurch ist MD2 für eine Implementierung auf 8-Bit Mikrocontrollern besser als die beiden anderen Verfahren geeignet.

MD2 verlängert die ursprüngliche Nachricht M bis zum nächsten Vielfachen von 16 Byte Länge, selbst wenn die Länge der ursprüngliche Nachricht bereits ein Vielfaches von 16 ist. Die neuen Bytes werden mit der Anzahl angehängter Byte belegt. Zusätzlich wird an die so verlängerte Nachricht M' eine 16 Byte lange Prüfsumme angehängt. Diese berechnet sich aus der bereits verlängerten Nachricht M' wie folgt:

ChkSum[0..15] = 0; $L = 0$
for Block = 0 to $(\frac{|M'|}{16} - 1)$ **do**
  for $i = 0$ to 15 **do**
    ChkSum[i] $\leftarrow$ChkSum[i] $\oplus$ $S[M'[16*\text{Block}+i] \oplus L]$
    $L \leftarrow$ ChkSum[i]
  end for
end for

Die Substitution S ist eine vorgegebene pseudozufällige 8-Bit SBox, die aus den Ziffern von π gewonnen wird. Die Nachricht inklusive Padding und Prüfsumme wird als M'' bezeichnet. Auf M'' wird iteriert auf 16 Byte grossen Blöcke die Rundenfunktion angewandt. Vor Verarbeitung des ersten Blocks wird die 48 Byte grosse Verkettungsvariable X mit 0 initialisiert. Danach wird für jeden der 16-Byte Blöcke von M'' folgende Funktion angewandt:

for $j = 0$ to 15 **do**
  $X[16 + j] \leftarrow M''[\text{Block}*16 + j]$
  $X[32 + j] \leftarrow X[16 + j] \oplus X[j]$
end for
$t \leftarrow 0$
for $j = 0$ to 17 **do**
  for $k = 0$ to 47 **do**
    $X[k] \leftarrow X[k] \oplus S[t]$
    $t \leftarrow X[k]$
  end for
  $t \leftarrow (t + j) \bmod 256$
end for

Nach Verarbeitung des letzten Blocks von M'' werden die ersten 16 Byte von X als Hashwert der Nachricht verwendet.

5.8.1 Sicherheit

In [62] wurde eine Methode vorgestellt, um Kollisionen in einem modifizierten MD2 Algorithmus zu erzeugen, der die Prüfsumme nicht verwendet ansonsten aber zum MD2 Algorithmus identisch ist. Gegen den Algorithmus mit Prüfsumme sind dagegen keine erfolgreichen Angriffe bekannt. Dennoch wird in [60] von einer Verwendung in einer Umgebung, die Kollisionsresis-

CPU	MIPS	Funktion	ROM in Byte	RAM in Byte	Durchsatz in Byte/Sek
ATMega32	8	`md2_compress("a")`	392	90	9411
ATMega32	8	`md2_process("a")`	730	90	4481

Tabelle 5.9: Leistungsdaten der MD2 Hashfunktion

tenz erfordert, abgeraten und lediglich die Verwendung als Einweg-Funktion weiterhin befürwortet.

5.8.2 Implementierung

Aufgrund der Verwendung des Byte als grundsätzliche Datenstruktur und der einfachen Rundenfunktion ist der Programmcode von MD2 deutlich kompakter im Vergleich zu den beiden anderen vorgestellten Hashfunktionen. In Tabelle 5.9 sind die Messergebnisse einer eigenen, nicht optimierten C-Implementierung angegeben. Es werden dabei die beiden Funktionen `md2_compress`, die nur die eigentliche MD2-Kompressionsfunktion durchführt und die Prüfsumme berechnet, sowie `md2_process`, die zusätzlich das Padding und das Hashen der Prüfsumme durchführt, betrachtet. Aufgrund der oben genannten Unsicherheit der Kompressionsfunktion ist es zwingend erforderlich, die `md2_process` Funktion zu verwenden. Der RAM-Bedarf beinhaltet den Übergabepuffer mit 16 Byte Größe, die 16 Byte Prüfsumme, den 48 Byte großen Arbeitspuffer, sowie Arbeitsvariablen. Wie aus den Daten ersichtlich ist, liegt die Leistung des MD2 Hashing-Verfahrens deutlich unterhalb SHA-1. Dies ist darauf zurückzuführen, dass immer mindestens zwei Aufrufe der Kompressionsfunktion erforderlich sind, da nach dem Hashing der eigentlichen Daten die Prüfsumme zu hashen ist. Da in allen drei Hashverfahren jeweils der Hashwert des Strings `"a"` berechnet wurde, ist somit in RIPEMD-128 und SHA-1 jeweils nur ein Aufruf der Kompressionsfunktion, in MD2 sind aber zwei Aufrufe erforderlich. Da in der vorausgesetzten Umgebung der Heimautomatisierung jedoch überwiegend sehr kleine Datenpakete verwendet werden, ist der Durchsatz bei der Verarbeitung kleiner Datenpakete ein wichtiges Kriterium. Eine weitere Erklärung für die schlechte Leistung ist, dass die Blockgrösse des MD2-Algorithmus zwar lediglich 16 Byte beträgt, aber, aufgrund der Erweiterung der Daten, die Rundenfunktion in 18 Runden auf einen Datenblock mit 48 Byte angewandt wird.

5.8.3 Bewertung

Trotz der geringen Codegröße eignet sich MD2 wegen der schlechten Leistung kaum zum Einsatz als Hashverfahren, zudem die Sicherheit des Algorith-

Eigenschaft	RC5	AES	Skip-jack	IDEA	RC4	MD2	SHA-1	RIPE-MD128
Multiplikation	–	–	–	X	–	–	–	–
Addition mit Übertrag	X	–	–	X	–	–	X	X
16 Bit Datenbreite	–	–	–	X	–	–	–	–
32 Bit Datenbreite	X	–	–	–	–	–	X	X
$\leq$ 25 Byte RAM	–	X	X	–	–	–	–	–
$>$ 25 Byte RAM	–	–	–	X	–	–	–	–
$>$ 50 Byte RAM	–	–	–	–	–	X	–	X
$>$ 100 Byte RAM	X	–	–	–	X	–	X	–
$\leq$ 500 Byte ROM	–	–	X	–	X	–	–	–
$>$ 500 Byte ROM	X	X	X	–	–	X	–	–
$>$ 1000 Byte ROM	–	–	–	X	–	–	X	–
$>$ 2000 Byte ROM	–	–	–	–	–	–	–	X

Tabelle 5.10: Überblick über die Anforderungen der Algorithmen

mus in Frage gestellt werden muss. Wegen der Verwendung der Prüfsumme und des dadurch erforderlichen Mehraufwands ist insbesondere in Umgebungen mit vielen kleinen Datenpaketen der Overhead durch den zusätzlich erforderlichen Aufruf für das Hashen der Prüfsumme sehr hoch. Eine Einsatzmöglichkeit, die sich aufgrund der geringen Codegröße anbietet, wäre, zusammen mit einer Stromchiffre wie RC4, einen sicheren Kanal durch Verwendung eines HMAC-Verfahrens zur Authentizitätssicherung zu implementieren. Allerdings benötigen RC4 und MD2 zusammen mehr als 1000 Byte ROM auf der Atmel AVR Plattform, während eine kompakte Blockchiffre wie IDEA, die sich sowohl zur Sicherstellung der Vertraulichkeit als auch (mittels CBC-MAC) der Integrität und Authentizität verwenden lässt, in 700 Byte implementiert werden kann. Zudem kann aufgrund der schlechten Leistung von MD2 die hohe Geschwindigkeit der RC4 Chiffre – eigentlich das Hauptargument für die Verwendung von RC4 – nicht ausgeschöpft werden.

5.9 Überblick

In der Tabelle 5.10 ist ein Überblick über die Anforderungen aller besprochenen symmetrischen Algorithmen gegeben. Tabelle 5.11 zeigt die Erfüllbarkeit dieser Anforderungen auf einigen handelsüblichen Mikrocontrollern und erlaubt somit zumindest eine grobe Entscheidungshilfe, welcher der Algorithmen sich auf einem bestimmten Mikrocontroller effizient implementieren lässt.

Eigenschaft	AVR	Mega AVR	PIC16	PIC18	ST6	ST7	i8051
Multiplikation	–	X	–	X	–	X	X
Addition mit Übertrag	X	X	–	X	–	X	X
$\leq$ 25 Byte RAM	X	X	X	X	X	X	X
$>$ 25 Byte RAM	X	X	X	X	X	X	X
$>$ 50 Byte RAM	X	X	X	X	X	X	X
$>$ 100 Byte RAM	X	X	X	X	X	X	X
$\leq$ 500 Byte ROM	X	X	X	X	X	X	X
$>$ 500 Byte ROM	X	X	X	X	X	X	X
$>$ 1000 Byte ROM	X	X	X	X	X	X	X
$>$ 2000 Byte ROM	X	X	X	X	X	X	X

Tabelle 5.11: Überblick über die Ausstattung der Mikrocontroller

5.10 Asymmetrische Verfahren

Als asymmetrische Verfahren werden heute im Wesentlichen drei Ansätze verwendet. Verfahren, die auf dem Integer-Faktorisierungsproblem basieren, solche, die auf dem diskreten Logarithmus Problem basieren und Verfahren, die auf dem diskreten Logarithmus Problem auf elliptischen Kurven basieren. Die beiden ersten Verfahren sind auf 8-Bit Systemen lediglich mit Hilfe von Coprozessoren möglich, einige Leistungsdaten sind bereits in Tabelle 4.1 angegeben. Lediglich Verfahren, die auf dem ECDLP basieren, können auf 8-Bit Mikrocontrollern ohne zusätzliche Coprozessorunterstützung implementiert werden. Ein weiterer Vorteil der ECC-Kryptographie neben der geringeren Schlüsselgröße ist eine einfachere Schlüsselgenerierung, da im Gegensatz zu den auf Integer Faktorierungsverfahren beruhenden Verfahren keine Primzahl als Schlüssel generiert werden muss, sondern eine beliebige Zufallszahl.

5.10.1 Sicherheit

Asymmetrische Verfahren werden vor allem dazu eingesetzt, Schlüssel eines symmetrischen Verfahrens auf sicherem Wege zu übermitteln. Daher ist es sinnvoll, die Sicherheit der symmetrischen Algorithmen auf die des Asymmetrischen abzustimmen. Es macht wenig Sinn, einen 80-Bit breiten Schlüssel für die Skipjack-Chiffre mit einem asymmetrischen Verfahren zu schützen, dessen Komplexität äquivalent zu einer 128-Bit symmetrischen Chiffre ist. Analog macht es wenig Sinn, eine symmetrische Chiffre mit einem 256-Bit breiten Schlüssel zu verwenden und diesen bei der Übermittlung nur mit einem asymmetrische Verfahren, das einen Aufwand von 2^{128} hat zu schützen.

Tabelle 5.12 zeigt die zu den üblichen Schlüssellängen symmetrischer Verfahren äquivalente asymmetrischen Schlüssellängen. Verglichen wird jeweils

Symmetrische Verfahren	RSA (n in Bit)	DLP (p+q in Bit)	ECC F_p $\|p\|$ in Bit	ECC F_{2^m} m
56	512	512+112	112	?
80	1024	1024+160	192	163
112	2048	2048+224	224	233
128	3072	2073+256	256	283
192	7680	7680+384	384	409
256	15360	15360+512	521	571

Tabelle 5.12: Sicherheitsäquivalenzen von symmetrischen und asymmetrischen Kryptoverfahren [21, 27]

der Aufwand in MIPS-Jahre, die zum Brute-Force-Brechen des Schlüssels erforderlich sind. Angegeben als Schlüsselgröße sind jeweils die Länge des die Stärke des Schlüssel bestimmenden Schlüsselteils.

Die in Abschnitt 5.10.2 beschriebenen Implementierungen erreichen maximal eine Sicherheit, die äquivalent zu einem 80-Bit Schlüssel eines symmetrischen Algorithmus ist.

5.10.2 Implementierung

In [73] wird die Implementierung der Punkmultiplikation auf einem Intel 8051 Mikrocontroller beschrieben. Der Mikrocontroller ist mit 12 MHz getaktet und erzielt eine Leistung von 1 MIPS. Er verfügt über 8-Bit Hardwaremultiplikation, insgesamt 256+512 Byte RAM, sowie 26 Kilobyte ROM. Auf der gewählten Kurve benötigt eine Skalarmultiplikation 1,83 Sekunden, ausgehend von einem festen Punkt und bei der Verwendung von 18 vorberechneten Zwischenpunkten. Die gewählte Kurve über einem Optimum-Extension-Körper der Ordnung 2^{135} gewährleistet dabei lediglich eine Sicherheit, die einem 56-Bit DES-Schlüssel entspricht (vergleiche Tabelle 5.12). Es werden circa 13 Kilobyte ROM benötigt sowie 523 Byte RAM. Eine Implementierung auf der MegaAVR-Plattform, die 16 anstelle von 1 MIPS erreicht, würde die Skalarmultiplikation innerhalb einer akzeptablen Zeit berechnen können, der Aufwand wäre jedoch weiterhin zu hoch aufgrund der geringen Sicherheit des zu einem 56 Bit langen symmetrischen Schlüssel äquivalenten ECC-Schlüssels.

In [34] wird eine leistungsfähigere Implementierung, allerdings auf einem 16-Bit CISC-Mikrocontroller, vorgestellt. Der Mikrocontroller M16C ist mit 10 MHz getaktet und erzielt maximal 10 MIPS. Er verfügt ebenfalls über Hardwaremultiplikation, wird mit 32, 64 und 96 Kilobyte ROM und 2, 3, 4 und 10 Kilobyte RAM angeboten. Eine typische Anwendung des M16C sind Mobiltelefone oder Modems. Es wurde ein Primkörper mit der 160-Bit grosse Primzahl $p = 65112 * 2^{144} - 1$ gewählt. Damit ergibt sich eine Sicherheit äquivalent zu einem 80-Bit breiten Skipjack-Schlüssel. Dies ist für

viele Anwendungen ausreichend. Der Gesamt-ROM-Bedarf für die Funktionen der Kurvenarithmethik, SHA-1 und EC-DSA-Signatur-Generierung und Verifikation beläuft sich auf 4000 Byte. Die Skalarmultiplikation erfordert ausgehend von einem fest gewählten Basispunkt 130 Millisekunden, die EC-DSA Signaturgenerierung insgesamt 150 Millisekunden und die Verifikation 630 Millisekunden. Obwohl der Mikrocontroller mit 20 MIPS nicht deutlich schneller als ein MegaAVR mit 16 MIPS scheint ist er deutlich leistungsfähiger. Da der M16C ein CISC-Prozessor ist kann er oftmals mit einem Opcode eine Berechnung ausführen, die auf einem RISC-Prozessor mehrere Opcodes erfordert. Zudem ist die Tatsache, dass der M16C Prozessor eine 16-Bit CPU ist, von grosser Bedeutung. Eine bei Elliptischer Kurven Kryptographie oftmals verwendete Operation ist die Multiplikation. Da, wie bereits erwähnt, die Durchführung einer 16-Bit Multiplikation auf einer 8-Bit CPU mehr als 20 Befehle erfordert müsste eine 8-Bit CPU deutlich mehr MIPS erzielen als eine 16-Bit CPU um deren Geschwindigkeit zu erreichen.

Weiterhin wird in [30] die Implementierung auf einem 16-Bit RISC-Mikrocontroller TI MSP430x33x beschrieben. Dieser Mikrocontroller verfügt über Hardwaremultiplikation, maximal 32 Kilobyte ROM sowie 1 Kilobyte RAM. Der CPU Takt beträgt 3,8 MHz was einer maximalen Leistung von 3,8 MIPS entspricht. Die in [30] gewählte Primzahl zur Bestimmung des Körpers ist $p = 2^{128} - 2^{97} - 1$ und somit eine verallgemeinerte Mersenne Primzahl. Wie bereits in Abschnitt 3.2.1 erwähnt, ist damit eine besonders einfache modulare Reduktion möglich. Eine Skalarmultiplikation ausgehend von einem zufälligen Punkt ist bei einer Taktrate von 3 MHz in bestenfalls 1,1 Sekunden möglich. Allerdings ist die Sicherheit des Verfahrens durch die Wahl der 128 Bit langen Primzahl nur gering und wird als äquivalent zu einem 56 Bit grossen DES Schlüssel angegeben.

5.10.3 Bewertung

Die Verwendung asymmetrischer Kryptographie ist auf heute verfügbaren 8-Bit Mikrocontrollern, wie sie im Rahmen des für diese Arbeit festgelegten Szenarios verwendet werden, ohne Unterstützung durch kryptographische Coprozessoren kaum möglich. Es ist zwar möglich einen Diffie-Hellman Schlüsseltausch zur einmaligen Vereinbarung eines Schlüssels in gerade noch akzeptierbarer Zeit durchzuführen. Allerdings ist der Diffie-Hellman Schlüsseltausch anfällig gegen Man-in-the-middle Angriffe und somit für auf Funkübertragung basierende Systeme nicht zu gebrauchen. Soll statt dessen ein Verfahren verwendet werden, das mit Hilfe von Zertifikaten den übermittelten öffentlichen Schlüssel auf Korrektheit prüft und daraufhin mit einem asymmetrischen Verschlüsselungsverfahren einen Schlüssel eines symmetrischen Verfahrens übermittelt, sind die Operationen deutlich aufwändiger. Beispielsweise sind für die Kombination DSA für die Zertifikatsverifikation und ElGamal zur Verschlüsselung eines symmetrischen Schlüssels für beide

Kommunikationspartner drei Skalarmultiplikationen erforderlich. Die Implementierung von Woodbury in [73] würde demnach $3 * 1,8 = 5,4$ Sekunden benötigen um einen Schlüssel zu übermitteln, der durch ein Verfahren mit der Stärke eines 56 Bit grossen DES-Schlüssels geschützt wäre. Lediglich unter Verwendung eines in Tabelle 4.1 vorgestellten Coprozessors können auf den in Kapitel 4 vorgestellten Mikrocontrollern asymmetrische Verfahren mit heute üblichen Schlüssellängen und akzeptierbarerem Zeitaufwand implementiert werden.
Dies gilt nur für die Laufzeit des Verfahrens, nicht aber die RAM oder ROM Kapazitäten. Wie den Beispielen der drei Implementierungen zu entnehmen ist, ist die erforderliche ROM-Kapazität deutlich unter den 64 oder 128 Kilobyte die für die vorgestellten Mikrocontroller üblicherweise den Maximalausbau darstellen. Die Implementierung nach [34] kommt beispielsweise mit 4 Kilobyte ROM aus, was die Implementierung von SHA-1 bereits mit einschließt. Somit wären bei dieser Implementierung bis maximal 92 Kilobyte ROM für die Programmlogik oder höhere Sicherheitsschichten verfügbar.

5.11 Zufallszahlengenerator

Wie bereits in Abschnitt 2.11 erläutert, ist die Generierung von Zufallszahlen von großer Bedeutung für viele kryptographische Algorithmen. Da ein in einem Embedded System verwendeter Mikrocontroller in der Regel ohnehin über Messeingänge verfügt, liegt es nahe, zufällige physikalische Beobachtungen zur Generierung von Zufallszahlen heranzuziehen. Abbildung 5.6 zeigt einen Zufallszahlengenerator, der als Rauschgenerator zwei Transistoren verwendet. Da der PN-Übergang des linken Transistors in Sperr-Richtung betrieben wird, fließt zur Basis des linken Transistors lediglich ein sehr schwacher, durch das thermische Rauschen bedingter Strom. Sobald durch diesen thermischen Stromfluss ausreichend Elektronen auf die Basis des rechten Transistors gelangt sind, so dass dessen Durchlass-Spannung erreicht ist, fließt durch ihn ein Strom und die Spannung an der Verstärkerschaltung fällt ab. Befinden sich durch den thermischen Stromfluss wenige Elektronen an der Basis des rechten Transistors wird die für ein Durchschalten benötige Basis-Emitter-Spannung nicht erreicht und der Transistor befindet sich im Sperrzustand. Die Spannung am Eingang der Verstärkerschaltung steigt dann. Die Verstärkerschaltung besteht aus zwei 7404 Invertern, die durch entsprechende Vorspannung in ihrem linearen Bereich betrieben werden und somit invertierende Verstärker darstellen. Durch den dritten Inverter erfolgt noch eine Anpassung auf den Pegel eines TTL-Signals das oftmals für einen Anschluss an einen Mikrocontroller erforderlich ist. Die beiden Pegel LOW und HIGH kommen allerdings nicht gleichmäßig vor. Daher ist es anzuraten, mit der von-Neumann Methode (vergleiche [53], Beispiel 5.10) diese Verzerrungen zu beseitigen. Dazu werden nicht die Pegel LOW und HIGH

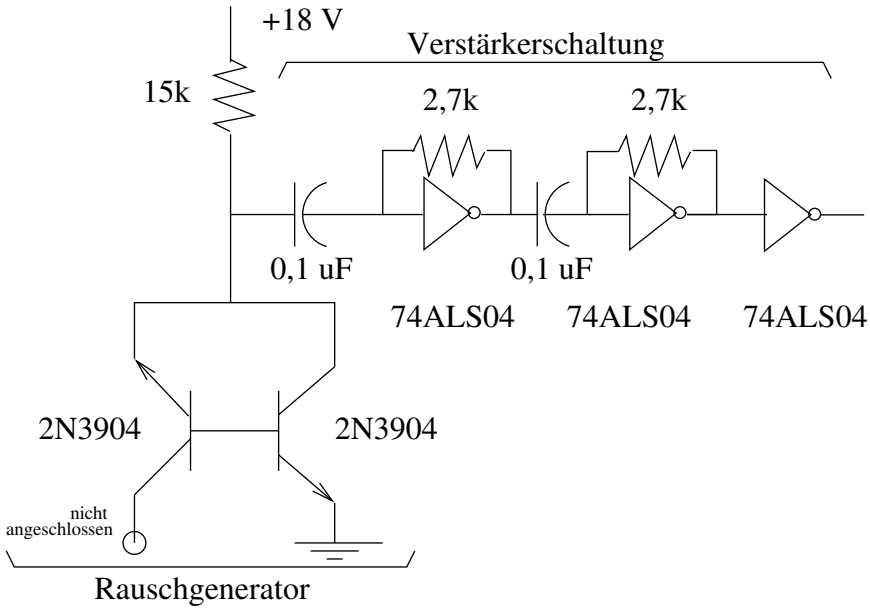

Abbildung 5.6: Hardware-Zufallszahlengenerator [72]

zur Generierung eines zufälligen Bitstroms verwendet werden, sondern die Flanke LOW-nach-HIGH als logische 0 und die Flanke HIGH-nach-LOW als logische 1 interpretiert wird. Die Wahrscheinlichkeit p für die Messung eines LOW-Pegels bedingt, dass die Wahrscheinlichkeit eines HIGH-Pegels $(1-p)$ ist. Die Wahrscheinlichkeit, bei zwei aufeinanderfolgenden Messungen, LOW und HIGH zu messen beträgt folglich $p*(1-p)$ und die, HIGH und LOW zu messen $(1-p)*p = p*(1-p)$. Wird bei beiden Messungen derselbe Pegel gemessen wird das Ergebnis verworfen. Zusätzlich wird in [53] empfohlen, den so generierten Bitstrom durch Anwendung eines kryptographischen Hashverfahrens oder einer Blockchiffre weiter zu entzerren. Weitere Datenquellen für Zufallszahlen könnten auch das thermische Rauschen eines Widerstandes oder einer Diode sein oder die Beobachtung nuklearer Zerfallsprozesse, wobei letzteres deutlich aufwändiger in der Implementierung ist.

5.12 Zusammenfassung

Wie Abschnitt 5.9 zu entnehmen ist, sind alle symmetrischen Algorithmen, mit Ausnahme von IDEA, auf allen besprochenen Hardware-Familien implementierbar. Allerdings wurde dabei nicht berücksichtigt, dass üblicherweise nur ein geringer Teil der verfügbaren Ressourcen für kryptographische Algorithmen verwendet werden kann. Somit ist, abhängig vom Ressourcenbedarf der anderen auf dem Mikrocontroller implementierten Programmteile, ei-

ne Implementierung auf einem Gerät mit geringen Ressourcen nicht immer möglich.

Die beiden Hashverfahren RIPEMD-128 und SHA-1 weisen einen deutlich höheren Ressourcenbedarf und einen geringeren Durchsatz als die Blockchiffren auf. Die überwiegende Anwendung von Hashalgorithmen besteht in einem symmetrischen Krypto-System in einem HMAC-Verfahren zur Sicherstellung der Authentizität. In diesem Fall kann durch Substitution des HMAC-Verfahrens durch ein auf dem CBC-MAC Verfahren basierenden die Notwendigkeit des Hashalgorithmus eliminiert werden. In diesem Fall ist eine der vorgestellten Blockchiffren für eine kryptographische Absicherung der Vertraulichkeit *und* der Authentizität ausreichend.

Eine Verwendung von asymmetrischer Kryptographie auf Mikrocontrollern des in dieser Arbeit vorausgesetzten Szenarios muss derzeit ausgeschlossen werden. Die Implementierungen, die in Abschnitt 5.10.2 vorgestellt wurden und die Skalarmultiplikation eines ECC-Punktes in einer akzeptablen Zeit berechnen können, sind auf deutlich leistungsfähigeren CPUs durchgeführt worden.

Kapitel 6

Zusammenfassung und Ausblick

Die vorangegangenen Kapitel haben gezeigt, dass die Anwendung symmetrischer kryptographischer Standard-Algorithmen auf ressourcenschwachen Geräten möglich ist. Da die Mikrocontroller in Sensoren und Aktoren im Szenario der Heimautomatisierung ohnehin eine sehr geringe Auslastung aufweisen, ist eine kryptographische Absicherung der Kommunikation ohne größere Schwierigkeiten möglich. Die Verwendung proprietärer Verfahren, die speziell für die Verwendung in ressourcenbeschränkten Systemen entwickelt wurden, wie das bereits erwähnte Keeloq-Verfahren, ist üblicherweise nicht erforderlich und sollte wegen der nicht verifizierbaren Sicherheit sowie möglichen Interoperabilitätsproblemen nicht verwendet werden. Moderne kryptographische Algorithmen, wie beispielsweise alle AES-Kandidaten, wurden bereits mit der Zielsetzung konzipiert, auch auf ressourcenschwachen 8-Bit Mikrocontrollern lauffähig zu sein.

Einige Teile dieser Arbeit, wie die Verwendung von paarweisen Schlüsseln, lassen sich nicht ohne Weiteres auf andere Szenarien des Ubiquitous Computing, wie beispielsweise die eingangs erwähnten Sensornetze, übertragen. Im Beispiel der aus einem Flugzeug abgeworfenen Sensoren kann keine Garantie gegeben werden, welcher Sensor mit welchem kommunizieren kann, da Sensoren durch den Fall beschädigt oder weit abseits der Anderen zum Liegen kommen könnten. Daher ist es in diesem Szenario besser, nach der in „Secure Pebblenets“ [4] vorgeschlagenen Methode des vorab auf allen Geräten hinterlegten Gruppenschlüssels vorzugehen. Da die Manipulationsresistenz der Geräte vorausgesetzt wird und die Einsatzdauer bei militärischen Anwendungen oder der Überwachung von Waldgebieten bei Bränden üblicherweise auf wenige Tage beschränkt ist, ist die Gefahr durch den gemeinsamen Schlüssel gut einschätzbar. Die Erkenntnisse über die Anwendbarkeit der symmetrischen kryptographischen Algorithmen gelten jedoch auch in diesem Szenario, da auch die manipulationsgeschützten Mikrocontroller

üblicherweise Standard-CPUs verwenden, die lediglich gegen Side-Channel-Angriffe gekapselt sind.

Aufgrund der Verfügbarkeit von Familien ähnlicher Mikrocontroller, die sich lediglich in den RAM/ROM-Kapazitäten unterscheiden, ist es in vielen Fällen möglich, eine kryptographische Absicherung der Kommunikation zu erreichen, indem der bestehende Mikrocontroller der Gerätesteuerung durch ein leistungsfähigeres Modell ausgetauscht wird. Die kryprographischen Algorithmen werden dann auf dem Steuerungs-Controller des Geräts implementiert. Dies verursacht im Vergleich zu einer Verwendung eines kryptographischen Coprozessors deutlich geringere Kosten und ist somit die anzustrebende Implementierungsform.

Auf die Verwendung von Hashing-Algorithmen, wie SHA-1 oder auch RIPEMD-128, sollte wenn möglich verzichtet werden. Beide Algorithmen, sowie die anderen Mitglieder der MD4 Familie, wurden nicht in Hinblick auf ressourcenbeschränkte 8-Bit CPUs entworfen, sondern an die Gegebenheiten typischer 32-Bit PC-Prozessoren ausgerichtet. Auch MD2, das zwar für 8-Bit CPUs ausgelegt wurde, jedoch aus anderen Gründen zu langsam ist, sollte nicht verwendet werden. Falls der Hashalgorithmus ausschließlich zur Berechnung eines Message Authentication Codes erforderlich ist, ist es sinnvoller, statt dessen mit einem CBC-MAC basierenden Verfahren zu arbeiten. In diesem Fall kann dieselbe symmetrische Blockchiffre zur Generierung des CBC-MAC und zur Chiffrierung verwendet werden.

Die Verwendung asymmetrischer Kryptographie ist auf den betrachteten Mikrocontrollern praktisch nur unter Zuhilfenahme eines Coprozessors möglich. Insbesondere bei Zuhilfenahme von Zertifikaten und einer Zertifizierungsstelle, um das Problem der Schlüsselverteilung zu lösen, entsteht ein sehr hohen Aufwand, durch die Notwendigkeit, für einen Schlüsseltausch mehrere asymmetrische Operationen auszuführen. Statt eines kryptographischen Coprozessors kann auch ein leistungsstärkerer 16-Bit Mikrocontroller anstelle des 8-Bit Mikrocontrollers verwendet werden, dessen höhere Leistung jedoch für die anderen vom Gerät ausgeführten Aufgaben nicht erforderlich ist. Die Schlüsselverteilung ist in ressourcenbeschränkten Systemen daher problematisch, da asymmetrische Kryptographie nicht ohne hohe Zusatzkosten einsetzbar ist. Statt asymmetrischer Kryptographie können zur Schlüsselverteilung Offline-Verfahren eingesetzt werden, wie den physikalischen Kontakt zweier Geräte zur Übermittlung eines Schlüssels über den dadurch entstehenden sicheren Kanal.

Durch die Verwendung der Blockchiffre im CTR-, CFB- oder OFB-Modus wird die Länge des Chiffrats gegenüber der des Klartextes nicht vergrößert. Dies ist in Umgebungen mit vor allem kleinen Nachrichten attraktiv. Wird die Blockchiffre im ECB- oder CBC-Modus verwendet, verlängert sich der letzte Block einer Nachricht um durchschnittlich 50%. Dies bedeutet für Nachrichten, die nur aus wenigen Blöcken bestehen, einen hohen Mehrbedarf an Bandbreite. Wegen der hohen Kommunikationskosten sollte daher

eine Blockchiffre immer im CTR-, CFB- oder OFB-Modus betrieben werden. Auch da diese Modi lediglich die Verschlüsselungsfunktion und nicht deren Umkehrung erfordern ist in ressourcenbeschränkten Umgebungen von Vorteil.

Bei der Implementierung eines Authenticated-Encryption Systems zur Bereitstellung eines sicheren Kommunikationskanals empfiehlt es sich, um Code einzusparen, ein CBC-MAC Schema zu verwenden und für CBC-MAC und Verschlüsselung dieselbe Blockchiffre zu verwenden. Auf die Entschlüsselungsfunktion der Blockchiffre kann auch hier verzichtet werden, da auch für den CBC-MAC-Algorithmus lediglich die Hinrichtung der Blockchiffre erforderlich ist. Wie bereits erwähnt, empfiehlt es sich, um EEPROM-Platz zur Speicherung der Schlüssel bekannter Kommunikationspartner einzusparen, die Schlüssel für Verschlüsselung und Authentizitätsicherung aus einem einzigen gemeinsamen Geheimnis herzuleiten. Dies reduziert jedoch nicht den RAM-Bedarf während der Kommunikation zweier Teilnehmer, da zwei unterschiedliche Schlüssel und deren gegebenenfalls erforderlichen Rundenschlüssel abgespeichert werden müssen.

Um den hohen RAM-Bedarf, der durch die Notwendigkeit zweier Schlüssel in generischen AE-Schemata besteht, zu reduzieren, empfiehlt es sich, anstelle der generischen Komposition, ein Schema wie EAX oder CCM zu verwenden. Der Mehraufwand des AEAD-Schemas EAX beschränkt sich, im Vergleich mit generischen AE-Schemata, die eine Blockchiffre zur Verschlüsselung und zur Generierung eines MAC verwenden, auf die Anwendung des OMAC-Algorithmus auf die unverschlüsselten aber authentisierten Daten und die NOnce. Zusätzlich sind beim Keysetup die Generierung einer Zufallszahl und drei Aufrufe der Blockchiffre erforderlich. Dafür wird bei EAX, im Gegensatz zu generischen AE-Schemata, lediglich ein Schlüssel benötigt, anstelle je eines Schlüssels für Verschlüsselung und MAC. Da bei generischen Verfahren keine Evaluation der Auswirkungen auf die Sicherheit des Verfahrens bei Verwendung eines einzelnen Schlüssels möglich ist, müssen dort zwei unabhängige Schlüssel verwendet werden. Andere Verfahren, wie CWC oder CCM, können ebenfalls mit nur einem Schlüssel betrieben werden. Da auf typischen Mikrocontrollern die geringe RAM-Kapazität oftmals die stärkste Beschränkung darstellt, ist es anzuraten, ein solches Schema zu verwenden, um die Notwendigkeit des zweiten Schlüssel zu eliminieren.

Mikrocontroller mit Coprozessoren, die entweder die für asymmetrische Kryptographie erforderlichen Funktionen beschleunigen oder vollständige kryptographische Einheiten enthalten, sind bereits erhältlich. Allerdings bedienen diese Mikrocontroller derzeit beinahe ausschließlich den Smartcard-Markt. Insbesondere mit neuen Standard Anwendungen für Kryptographie, wie der Trusted Computing Initiative [29], die sich unter anderem der Sicherheit von PC-Systemen verschrieben hat, ist ein bessere Verfügbarkeit von kostengünstiger kryptographischer Hardware wahrscheinlich. Eine optiona-

le Erweiterung handelsüblicher Mikrocontroller um einen kryptographischen Coprozessor wäre wünschenswert. Alternativ dazu wäre eine stärkere Verwendung von 16-Bit Mikrocontroller vorstellbar. Wie in [34] gezeigt wurde ist die Leistungfähigkeit eines 16-Bit Mikrocontrollers für die Verwendung asymmetrischer Kryptographie ausreichend. Derzeit sind jedoch 16-Bit Mikrocontroller in der Regel deutlich teuerer im Vergleich zu 8-Bit Geräten mit vergleichbaren Leistungsdaten und außer für die asymmetrische Kryptographie nicht erforderlich. Dadurch würde der Preis eines Geräts mit sicherer Übertragung deutlich über einem Modell liegen, das keine sichere Übertragung anbietet. Dies würde höchstwahrscheinlich zu grossen Akzeptanzproblemen bei den Endkunden führen.

Allgemein ist die geringe Sensibilität der Endkunden gegenüber Sicherheitsrisiken ein großes Problem für die Einführung sicherer Heimautomatisierungssysteme. Da eine Erweiterung eines Systems um eine sichere Kommunikation immer mit Mehrkosten verbunden ist, ist es zu befürchten, dass Konkurrenzprodukte, die nicht über gesicherte Kommunikation verfügen, dafür aber billiger sind, bevorzugt werden. Da bislang keine sicheren Heimautomatisierungsprodukte erhältlich sind, gibt es keine Studien, die untersuchen, ob Verbraucher bereit sind, für Sicherheit in diesem Bereich einen höheren Preis zu zahlen. Da allerdings oftmals für die Sicherung des Eigenheims große Beträge für Alarmanlagen und gute Schlösser ausgegeben werden, wäre auch eine Bereitschaft zu Investitionen im Bereich der sicheren Heimautomatisierung denkbar.

Literaturverzeichnis

[1] National Security Agency. SKIPJACK and KEA algorithm specification version 2.0, May 1998.

[2] American National Standards Institute. *ANSI X9.31-1998: Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry (rDSA).* 1988.

[3] Ross Anderson and Markus Kuhn. Low cost attacks on tamper resistant devices. In *IWSP: 5th International Workshop of Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 1997.

[4] Bruschi D. Rosti E. Basagni S., Herrin K. Secure pebblenets. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking and computing, Long Beach, CA, USA*, pages 156–163. ACM Press, 2001.

[5] M. Bellare, P. Rogaway, and D. Wagner. A conventional authenticated-encryption mode. manuscript, April 2003.

[6] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of cipher block chaining. In *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358. Springer, 1994.

[7] E. Biham. A note on comparing the AES candidates. In *The Second Advance Encryption Standard Candidate Conference, Rome, Italy*, pages 85–94. National Institute of Standards and Technology, March 1999.

[8] E. Biham, A. Biryukov, and A. Shamir. Miss in the middle attacks on IDEA, Khufu, and Khafre. In Lars Knudsen, editor, *Fast software encryption: 6th International Workshop, FSE'99, Rome, Italy*, volume 1636 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 1999.

[9] Eli Biham. New types of cryptanalytic attacks using related keys. In *Advances in Cryptology - EUROCRYPT 1993, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway*, volume 765 of *Lecture Notes in Computer Science*, pages 398–409. Springer, 1994.

[10] Eli Biham, Alex Biryukov, Orr Dunkelman, Eran Richardson, and Adi Shamir. Initial observations on skipjack: Cryptanalysis of Skipjack-3XOR. In *Selected Areas in Cryptography*, volume 1556 of *Lecture Notes in Computer Science*, pages 362–376. Springer, 1999.

[11] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999.

[12] Alex Biryukov, Jorge Nakahara, Jr., Bart Preneel, and Joos Vandewalle. New weak-key classes of IDEA. In *4th International Conference on Information and Communications Security - ICICS'02*, volume 2513 of *Lecture Notes in Computer Science*, pages 315–326. Springer, December 2002.

[13] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: The insecurity of 802.11. In *Proceedings of the 7th International Conference on Mobile Computing and Networking, Rome, Italy*, pages 180–189. ACM Press, July 2001.

[14] Michael Brown, Darrel Hankerson, Julio López, and Alfred Menezes. Software implementation of the NIST elliptic curves over prime fields. In *Topics in Cryptology - CT-RSA 2001, RSA Conference 2001, San Francisco, CA, USA*, volume 2020 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2001.

[15] Holger Cermann. Design eines Embedded Security Layers für ressourcenbeschränkte Geräte, 2003.

[16] Certicom. The elliptic curve cryptosystem for smart cards, May 1998.

[17] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002.

[18] R. Housley D. Whiting and N. Ferguson. Counter with CBC-MAC (CCM) - Submission to NIST, AES modes of operations, June 2002.

[19] J. Daemen and V. Rijmen. AES proposal: Rijndael. In *First Advanced Encryption Standard (AES) Conference, Ventura, CA, USA*. National Institute of Standards and Technology, August 1998.

[20] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A strengthened version of RIPEMD. In *Fast Software Encryption, Third International Workshop, Cambridge, UK*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 1996.

[21] Certicom Don B. Johnson. ECC, future resiliency and high security systems. online white paper, http://www.certicom.com/resources/download/ECCFut.pdf, March 1999.

[22] EIB - European Installation Bus. Webpage http://www.eiba.com.

[23] ELV - FS20 RF Remote Control System. Webpage http://www.elv.de.

[24] B. Schneier et al. The twofish team's final comments on AES selection, AES round 2 public comments, May 2000.

[25] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting. Improved cryptanalysis of Rijndael. In *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2001.

[26] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In *Selected Areas in Cryptography - 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada*, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2001.

[27] National Institute for Standards and Technology. Recommended elliptic curves for federal government use, July 1999.

[28] P. Gaudry, F. Hess, and N. P. Smart. Constructive and Destructive Facets of Weil Descent on Elliptic Curves. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 15(1):19–46, 2002.

[29] Trusted Computing Group. TCG main specification version 1.1b. Webpage http://www.trustedcomputinggroup.org/.

[30] Jorge Guajardo, Rainer Blümel, Uwe Krieger, and Christof Paar. Efficient implementation of elliptic curve cryptosystems on the TI MSP430x33x family of microcontrollers. In *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea*, volume 1992 of *Lecture Notes in Computer Science*, pages 365+. Springer, 2001.

[31] Helena Handschuh and Howard M. Heys. A timing attack on RC. In *Selected Areas in Cryptography - SAC'98, Kingston, Ontario, Canada*, volume 1556 of *Lecture Notes in Computer Science*, pages 306–318. Springer, 1999.

[32] Helena Handschuh and Pascal Paillier. Smart card crypto-coprocessors for public-key cryptography. *CryptoBytes*, 4(1):6–11, Summer 1998.

[33] Darrel Hankerson, Julio López Hernandez, and Alfred Menezes. Software implementation of elliptic curve cryptography over binary fields. In *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA*, volume 1965 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2000.

[34] Hasegawa, Nakajima, and Matsui. A small and fast software implementation of elliptic curve cryptosystems over GF(p) on a 16-bit microcomputer. *TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems*, 1999.

[35] Fraunhofer IIS. Embedded communication - 868 MHz ISM-Band-Transceiver. Webpage.

[36] B. Kalsiki RSA Data Security Inc. RFC1319, The MD2 Message-Digest Algorithm, 1992.

[37] Microchip Technology Inc. Keeloq authentication products. Webpage http://www.microchip.com.

[38] T. Iwata and K. Kurosawa. OMAC: One-Key CBC MAC. In *Pre-Proceedings of Fast Software Encryption 2003, Lund, Sweden*, 2003.

[39] D. Johnson and A. Menezes. The elliptic curve digital signature algorithm (ECDSA). Technical Report CORR 99-34, Univ. of Waterloo, August 1999.

[40] B. Kaliski and Y. Yin. The security of the RC5 encryption algorithm. *CryptoBytes Vol.1, No.2*, pages 13–14, 1995.

[41] John Kelsey, Bruce Schneier, and David Wagner. Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology*

Conference, Santa Barbara, California, USA, volume 1109 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 1996.

[42] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 1969.

[43] Neal Koblitz. *A course in number theory and cryptography*. Graduate texts in mathematics. Springer-Verlag, 1987.

[44] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[45] Francois Koeune and Jean-Jacques Quisquater. A timing attack against Rijndael. Technical Report CG-1999/1, UCL Crypto Group, Louvain-la-Neuve, 1999.

[46] Tadayoshi Kohno, John Viega, and Doug Whiting. The CWC authenticated encryption (associated data) mode.

[47] Oliver Kömmerling and Markus G. Kuhn. Design principles for tamper-resistant smartcard processors. In *Proceedings of the USENIX Workshop on Smartcard Technology, Chicago, USA, 10–11 May, 1999.*, pages 9–20, 1999.

[48] H. Krawczyk, M. Bellare, and R. Canetti. RFC 2104: HMAC: Keyed-hashing for message authentication, February 1997.

[49] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. Springer, 2001.

[50] Julio López and Ricardo Dahab. An overview of elliptic curve cryptography. Technical Report IC-00-10, Institute of Computing, State University of Campinas, Brazil, May 2002.

[51] M. Matsui and A. Yamagaishi. A new method for known plaintext attack of FEAL cipher. In *Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary*, volume 658 of *Lecture Notes in Computer Science*, pages 81–91. Springer, 1993.

[52] Alfred Menezes. Elliptic curve cryptosystems. *CryptoBytes Volume 1 Number 2*, pages 1–3, 1995.

[53] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press, 1997.

[54] Ilya Mironov. The (not so) random shuffles of RC4. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA*, volume 2442 of *Lecture Notes in Computer Science*, pages 304–319. Springer, 2002.

[55] S. Murphy. The cryptanalysis of FEAL-4 with 20 chosen plaintexts. *Journal of Cryptology, 2(3)*, pages 145–154, 1990.

[56] NIST, U. S. Department of Commerce, Washington, DC, USA. *Digital Signature Standard*, May 1994. FIPS PUB 186.

[57] Jochen Noller. Design eines Embedded Access Control Layers für ressourcenbeschränkte Geräte, 2003.

[58] Adrian Perrig, Robert Szewczyk, Victor Wen, David E. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *MOBICOM 2001, Proceedings of the seventh annual international conference on Mobile computing and networking, Rome, Italy*, pages 189–199. ACM Press, 2001.

[59] Ronald L. Rivest. The RC5 encryption algorithm, from dr. dobb's journal, january, 1995. In *William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996.* 1996.

[60] Matt Robshaw. On recent results for MD2, MD4 and MD5, RSA Laboratories' Bulletin No. 4. Technical report, RSA Laboratories, November 1996.

[61] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA*, pages 196–205. ACM Press, 2001.

[62] N. Rogier and P. Chauvaud. The compression function of MD2 is not collision free. In *Selected Areas in Cryptography*, 1995.

[63] B. Van Rompay, B. Preneel, and J. Vandewalle. On security of dedicated hash functions. In P. H. N. de With and M. van der Schaar-Mitrea, editors, *19th Symp. on Information Theory in the Benelux*, pages 103–110, Veldhoven (NL), 28-29 1998. Werkgemeenschap Informatie- en Communicatietheorie, Enschede (NL).

[64] Fumihiko Sano, Masanobu Koike, Shin-Ichi Kawamura, and Masue Shiba. Performance evaluation of AES finalists on the high-end smart card. In *The Third Advanced Encryption Standard Candidate Conference, New York, New York, USA*, pages 82–93. National Institute of Standards and Technology, April 2000.

[65] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. Performance comparison of the AES submissions. In *The Second Advanced Encryption Standard Candidate Conference, Rome, Italy*, pages 15–34. National Institute of Standards and Technology, March 1999.

[66] Kai Schramm. Embedded Smartcard Controllers - Vergleich aller zehn AES Implementierungen. Webpage http://www.crypto.rub.de/praktika/escm/.

[67] David A. McGrew Scott R. Fluhrer. Statistical analysis of the Alleged RC4 keystream generator. In *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA*, volume 1978 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 2001.

[68] C.E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

[69] Eric Smith. SKIPJACK for the Microchip PIC and Scenix SX. Webpage http://www.brouhaha.com/ eric/crypto/.

[70] Frank Stajano and Ross Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Security Protocols, 7th International Workshop, Cambridge, UK*, volume 1796 of *Lecture Notes in Computer Science*, pages 172–194. Springer, 2000.

[71] Jim Turley. The two percent solution. Webpage http://www.embedded.com/story/OEG20021217S0039.

[72] Will Ware. Hardware Random Bit Generator. Webpage http://willware.net:8080/hw-rng.html.

[73] A. Woodbury, D. Bailey, and C. Paar. Elliptic curve cryptography on smart cards without coprocessors. In *IFIP CARDIS 2000, Fourth Smart Card Research and Advanced Application Conference, Bristol, UK*. Kluver, September 2000.

[74] X10 Home Automation System. Webpage http://www.x10.com.

Erklärung

Hiermit versichere ich, diese Arbeit
selbständig verfaßt und nur die
angegebenen Quellen benutzt zu haben.

(Ulrich Wurst)