Eine Auswahl

Kryptoalgorithmen

Revision : 254

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

$(G, \diamond)$	Verknüpfungsgebilde, bestehend aus Menge (Gruppe) G und Operation $\diamond$
$(R, +, \cdot)$	Ring
$(R[x], +, \cdot)$	Polynomring
$(R_m, +, \cdot)$	Restklassenring
$[M : K]$	Grad der Körpererweiterung M über K
$\|\|x\|\|$	Länge von x in Byte
$\lceil x \rceil$	Kleinste ganze Zahl größer als x (ceil)
$\langle a \rangle$	Vom Element a erzeugte Untergruppe
$\|G : H\|$	Index der Gruppe G über H
$\|G\|$	Ordnung der Gruppe G
$\lfloor x \rfloor$	Größte ganze Zahl kleiner als x (floor)
$[r]_m$	Restklasse zum Modul m
$\{a_1, a_2, \ldots\}$	Menge von Elementen a_i
$a \subseteq b$	a ist Teilmenge von b
$\|\|$	Concatenation
$\#G$	Ordnung der Gruppe G
$\diamond$	Operation $\diamond$
$\equiv$	Kongruenz
$\oplus$	Exklusiv-Oder (Modulo-2, XOR)

$\vee$	(Bit-für-Bit) ODER
$\wedge$	(Bit-für-Bit) UND
δ_{ij}	KRONECKER-Symbol
$\phi(m)$	EULER'sche Totient-Funktion
$\overline{a}$	Teilerfremder Teil von a bezüglich einer anderen Zahl
$a \perp b$	a und b sind teilerfremd
$a \mapsto b$	Abbildung von a auf b
$a \mid b$	a teilt b
$a \setminus b$	Differenzmenge a abzüglich b
$a \subseteq b$	a ist Teilmenge von b
$\mathbb{C}$	Körper der komplexen Zahlen
$\deg[h(x)]$	Grad des Polynoms $h(x)$
$\dim(V)$	Dimension des Vektorraums V
e	neutrales Element
$e_{(+)}$	Null-Element (neutrales Element der additiven Gruppe)
$e_{(\cdot)}$	Eins-Element (neutrales Element der multiplikativen Gruppe)
$\mathbb{F}_p^n$	Erweiterungskörper der Dimension n über p
$\mathbb{F}_q$	Endlicher Körper mit q Elementen
$\mathbb{F}_q^*$	Multiplikative Gruppe des endlichen Körpers $\mathbb{F}_q$
$\gcd(a,b)$	Größter gemeinsamer Teiler von a und b
$\mathrm{GF}(q)$	GALOIS-Körper mit q Elementen
K	Körper
$\mathrm{lcm}(a,b)$	Kleinstes gemeinsames Vielfaches von a und b
$\mod$	Modulo
$\mathbb{N}$	Menge der natürlichen Zahlen

ord(G)	Ordnung der Gruppe G
$\mathbb{P}$	Menge der Primzahlen
p	Primzahl (bzw. Primelement)
$\mathbb{Q}$	Körper der rationalen Zahlen
$\mathbb{R}$	Körper der reellen Zahlen
$\text{rol}^i(x)$	Linksrotation von x um i Bit
$\mathbf{v}$	Vektor v
V	Vektorraum
$\mathbb{Z}$	Menge der ganzen Zahlen
$\mathbb{Z}_m$	Restklassenring/Menge[1] der ganzen Zahlen $\bmod m$
$\mathbb{Z}_p$	Primzahlenkörper
$\mathbb{Z}_p[x]/m(x)$	Polynom-Restklassenring auf dem Grundkörper $\mathbb{Z}_p$

Abkürzungsverzeichnis

AAL	ATM Adaption Layer
ASN	Abstract Syntax Notation
ATM	Asynchronous Transfer Mode
AUTH	Authentication
BT	Block Type
CA	Certification Authority
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CONF	Confidentiality
CRC	Cyclic Redundancy Check
CRL	Certificate Revocation List
D	Decrypt, Data
DB	Data Block
DES	Data Encryption Standard
DH	Diffie-Hellman
DSS	Data Signature Standard
E	Encrypt
EB	Encoded Block
EC	Elliptic Curve
ECB	Electronic Codebook
EC/DSA	Elliptic Curve Digital Signature Algorithm
EC/KAS	Elliptic Curve Key Agreement Scheme
EM	Encoded Message

[1] Die ganzen Zahlen von 0 bis $m-1$

FB	Feedback
FEAL	Fast Data Encipherment Algorithm
HMAC	Hashed MAC
ICV	Integrity Check Values
ID	Identifier
INTEG	Integrity
ISK	Initial Session Key
IV	Initialisation Vector
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
MAC	Message Authentication Code
MD5	Message Digest 5
MSB	Most Significant Bit
OAEP	Optimal Asymmetric Encryption Padding
OID	Object Identifier
OUI	Organizationally Unique Identifier
PDU	Protocol Data Unit
PKCS	Public Key Cryptographic Standards
PS	Padding String
QoS	Quality of Service
RNG	Random Number Generator
RSA	RIVEST-SHAMIR-ADLEMAN
SHA	Secure Hash Algorithm
SN	Sequence Number
SV	State Vector

1. Symmetrische Algorithmen

1.1. Blockchiffren

1.1.1. Einordnung

C.E. SHANNON beschreibt in [Sha49] ein allgemeines (symmetrisches[2]) Kryptosystem entsprechend Abbildung 1 und formuliert auf dieser Grundlage erstmalig grundlegende Theoreme zur theoretischen Sicherheit.

In Teil III widmet er sich auch praktischen Aspekten der Auswahl bzw. des Designs solcher Verschlüsselungsalgorithmen. Dabei bezieht er sich unter anderem auf die folgenden Kriterien zur Bewertung von Kryptosystemen:

- die benötigte Menge an Kryptomaterial für einen erfolgreichen Angriff;
- die Schlüssellänge;
- die Komplexität des Ver- und Entschlüsselungsalgorithmus';
- die Fehlerfortpflanzung;
- und das (nicht) notwendige Padding.

[2] Symmetrisch deshalb, weil für Ver- und Entschlüsselung derselbe Schlüssel K zur Anwendung kommt. Der schraffierte Bereich in Abbildung 1 wurde hinzugefügt um den unsicheren Kanal zu markieren. Damit wird auch die Aussage anschaulich, daß der Schlüssel K auf sicherem Wege zum Empfänger gelangen muß.

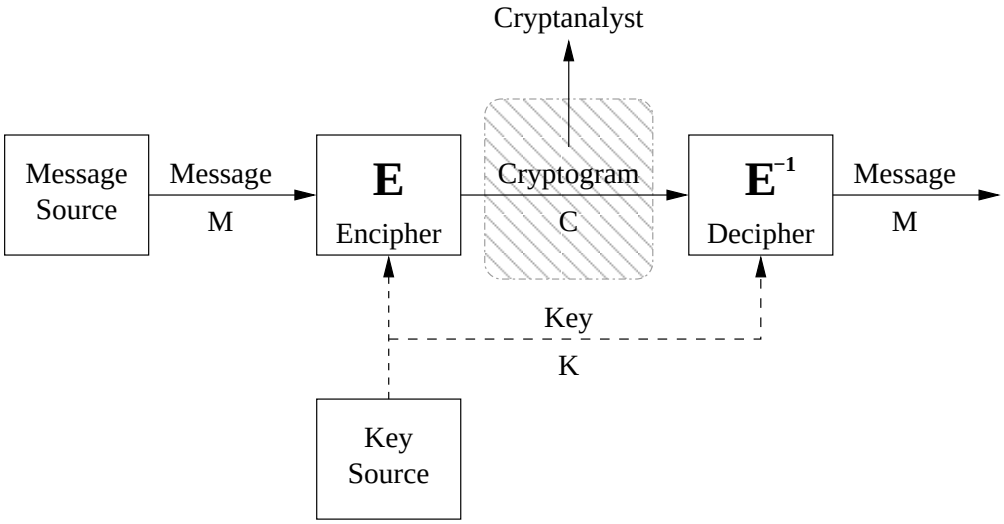


Abbildung 1: SHANNON's Modell

1.1.2. Wirkprinzip

Blockchiffren zeichnen sich dadurch aus, daß sie einen Klartext-Vektor (Plaintext) der Breite n (Bit) in einen gleichgroßen Vektor verschlüsselte Daten (Ciphertext) transformieren. Diese Abbildung E von 2^n möglichen Eingangsmustern auf wieder 2^n Ausgangsmuster ist praktisch immer eineindeutig (affin, reversibel) und wird durch den geheimen Schlüssel K gesteuert. Besteht der Klartext wie in Abbildung 2 aus mehr Bits als die Blockbreite n vorgibt, dann muß er in Teilblöcke dieser Breite segmentiert und der letzte Block unter Umständen mit einem Muster *pad*(ding) aufgefüllt werden.

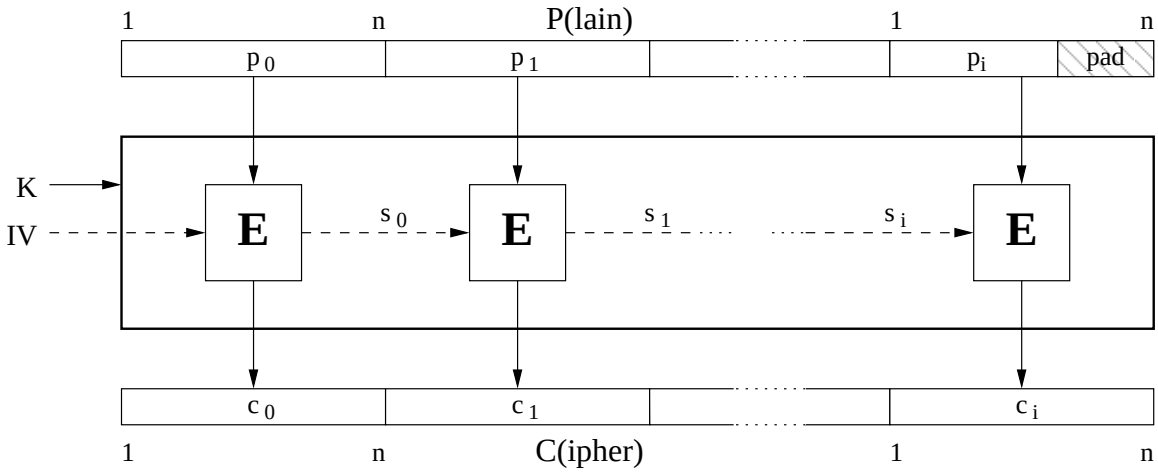


Abbildung 2: Anwendung von Blockchiffren

Die Klartext-Blöcke p_i werden dann nacheinander durch den Algorithmus E in Ciphertext-

Blöcke $c_i = E(p_i)$ transformiert. Dieser Prozeß kann vollständig unabhängig für jede Blockchiffre (bzw. jeden Block p_i) ablaufen[3] oder aber durch die Weitergabe von Zustandsvektoren s_i beeinflußt sein[4]. Den ersten Zustandsvektor nennt man Initialisierungsvektor (IV) und wählt ihn entweder zufällig (dann muß er zum Ort der Entschlüsselung übertragen werden) oder per Vereinbarung, z. B. als statisches Muster. Bei der Entschlüsselung läuft der gesamte Prozeß umgekehrt ab, d. h. derselbe Schlüssel K wird verwendet um mit Hilfe des inversen Algorithmus jeden Klartext-Block $p_i = E^{-1}(c_i)$ wieder zu gewinnen.

1.1.3. Herkömmliche Struktur

Shannon hat schon in vorgeschlagen symmetrische Kryptoalgorithmen durch Anwendung von Substitution (als nichtlineare Komponente) und Permutation zu realisieren. Fast alle bekannten Blockchiffren wenden dieses Prinzip auf der Grundlage einer einfachen Verarbeitungsstruktur nach H. Feistel an [HFS75]. Dazu wird der Plaintext-Block p in zwei Hälften l_0 und r_0 aufgeteilt ($p = l_0 \| r_0$) und an das Feistel-Netzwerk nach Abbildung 3 übergeben. Das Ergebnis $l_1 \| r_1$ wird danach wieder auf den Eingang zurückgeführt und das Verfahren in mehreren Runden wiederholt.

$$l_{i+1} = r_i \qquad\qquad r_{i+1} = l_i \oplus f(r_i, K_i) \tag{1}$$

Der sogenannte Rundschlüssel K_i wird aus dem Schlüssel K abgeleitet (Key Scheduling) und sollte für jede Runde verschieden sein. Die nichtlineare Funktion f realisiert dabei die Substitution (in Abhängigkeit vom Rundenschlüssel), die ständige Kreuzung von linker und rechter Hälfte zusammen mit der Exklusiv-Oder (XOR) Operation eine reversible[5] Permutation (und „Durchmischung").

Die Vorteile des Verfahrens gründen sich auf dessen Einfachheit:

1. Aufwand und Kosten der Realisierung sind überschaubar;
2. Hardware- und Software-Implementierungen sind gleichermaßen möglich;
3. die Skalierbarkeit (nach Sicherheitsanforderungen, Geschwindigkeit oder anderen Kriterien) ist durch Variation der Rundenanzahl gegeben;
4. Ver- und Entschlüsselung können dieselbe Struktur verwenden.

Der letzte Punkt soll noch kurz erläutert werden, wobei von Formel 1 auszugehen ist. Stellen wir diese einfach für die Rückwärtsrichtung (Entschlüsselung) um, so ergibt sich:

[3] In diesem Fall spielt die Reihenfolge der Verschlüsselung der einzelnen Blöcke p_i keine Rolle.

[4] In diesem Fall müßte man, um die Unterschiede in den einzelnen Schritten kenntlich zu machen, eigentlich E_i statt einfach nur E schreiben. Darauf wurde jedoch verzichtet, weil das Subskript an E üblicherweise für den verwendeten Schlüssel reserviert ist, z. B. so: $c_i = E_K(p_i)$.

[5] Operationen die zu einem Informationsverlust führen (And, Or, …) müssen an dieser Stelle vermieden werden.

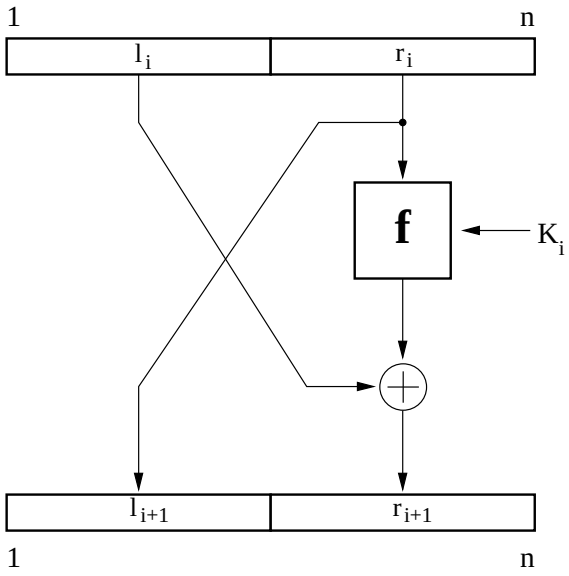


Abbildung 3: Runde einer typischen FEISTEL-Chiffre

$$r_i = l_{i+1} \qquad l_i = r_{i+1} \oplus f(l_{i+1}, K_i), \tag{2}$$

d. h. alle Operationen bleiben erhalten[6]. Was sich ändert ist einzig und allein die Verwendung der Rundenschlüssel, welche bei der Verschlüsselung mit K_0 startete. Bei der Entschlüsselung muß man, wie Formel 2 zeigt, umgekehrt vorgehen, also den Schlüssel K_0 zuletzt benutzen.

1.1.4. Advanced Encryption Algorithm (AES)

Der AES ist definiert im NIST-Standard [FIP01] und mittlerweile auch als [ISO05]. Es gibt so viele Beschreibungen und Implementierungshinweise zum AES, daß ich dies hier nicht weiter ausführe.

1.2. Betriebsarten

Normalerweise bildet eine Blockchiffre *n* bit Klartext auf die gleiche Anzahl verschlüsselter Bits ab (Electronic Codebook, ECB-Modus). Betriebsarten, wie im Folgenden beschrieben, verknüpfen die Eingangs- und Ausgangsvektoren durch Rückkopplungen mittels modularer Arithmetik. Auf diese Weise werden ganz spezielle Eigenschaften erzeugt und kryptoanalytische Nachteile

[6] Insbesondere wird eine Umkehrfunktion f^{-1} nicht benötigt.

der einen oder anderen Betriebsart umgangen. Eine genaue Auflistung der jeweiligen Eigenschaften geben z. B. [ISO06b], [Dwo01] sowie [MvV92, 7.2.2], historische Spezifikationen der Betriebsarten sind in [ANS98, NBS80] zu finden.

1.2.1. Cipher Block Chaining (CBC)

In der Betriebsart CBC wird jeder Plaintext-Block p_i vor der Verschlüsselung mit dem letzten Ciphertext-Block c_{i-1} kombiniert (vgl. auch Abbildung 4).

$$c_i = E_K(p_i \oplus c_{i-1}), \qquad p_i = c_{i-1} \oplus D_K(c_i), \qquad c_{-1} = IV \tag{3}$$

Dadurch ergibt sich eine Abhängigkeit über den gesamten zu verschlüsselnden Klartext, welche z. B. beim CBC-MAC ausgenutzt wird.

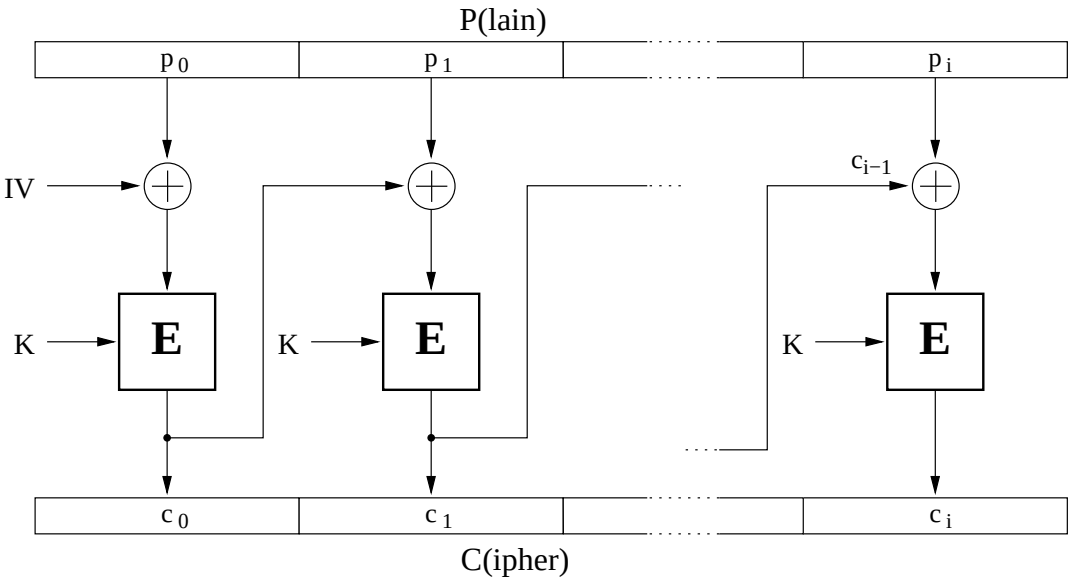


Abbildung 4: CBC-Verschlüsselung

An Hand von Abbildung 5 (oder aus Formel 3) kann man das Entschlüsselungsverfahren leicht erklären. Jeder Block c_i wird zuerst entschlüsselt, danach die Exklusiv-Oder (XOR) Operation

mit Hilfe des “Vorgängers” c_{i-1} rückgängig gemacht[7]. Für den ersten Block wird als Startwert ein Initialisierungsvektor (IV) verwendet, der auch nicht unbedingt geheim sein muß.

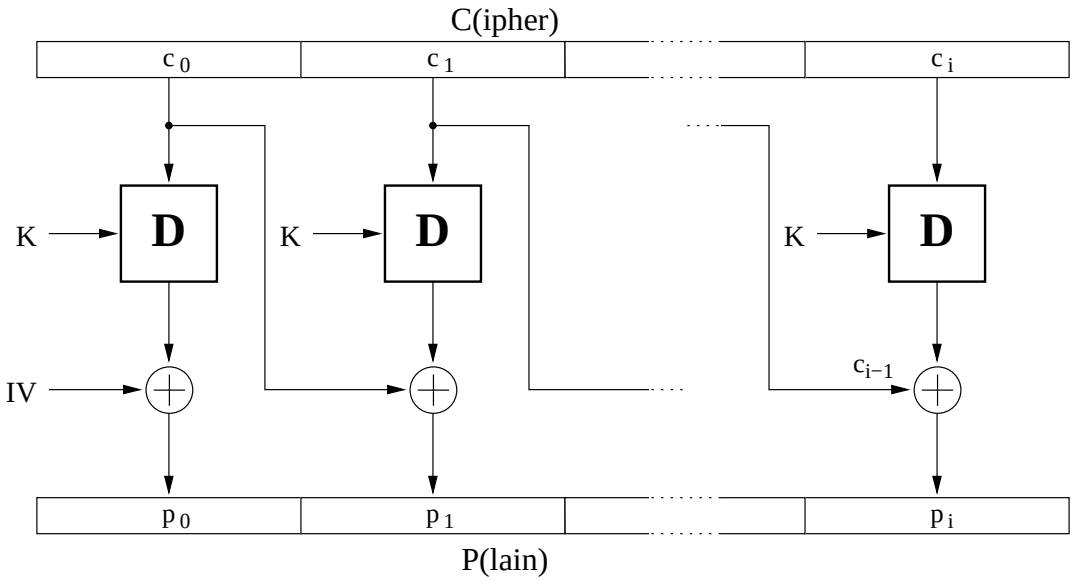


Abbildung 5: CBC-Entschlüsselung

Die Eigenschaften des CBC-Modus bezüglich der Fehlerfortpflanzung lassen sich entweder aus Abbildung 5 oder aber Formel 3 ableiten:

1. Bitfehler im Block c_i wirken sich auf den aktuellen sowie den nächsten Klartext-Block aus. Wegen $p_i = c_{i-1} \oplus D_K(c_i)$ sind in p_i nahezu alle Bits verfälscht[8]. In $p_{i+1} = c_i \oplus D_K(c_{i+1})$ hingegen sind es nur solche, die auch in c_i fehlerhaft waren. War deren Anzahl m so führt die Fehlerfortpflanzung zu $n+m$ verfälschten Bits.
2. Geht die Blocksynchronisation verloren, d. h. wird beispielsweise der Block c_1 in Abbildung 5 gar nicht erst empfangen (und an dessen Stelle c_2 verwendet), so wird der zugeordnete entschlüsselte Block p_2 (jetzt an Stelle von p_1) komplett gestört sein. Alle weiteren Blöcke sind jedoch wieder korrekt[9], weshalb man auch von einer selbstsynchronisierenden Betriebsart spricht (self-synchronizing, ciphertext autokey).

[7] Da in diesem Modus der Ciphertext direktes Resultat der jeweiligen Blockverschlüsselung ist, benötigt man (im Gegensatz zu OFB- oder CFB-Modus) für die Entschlüsselung den inversen Algorithmus.

[8] Der Fehler in c_i geht als Eingangsvektor in den Entschlüsselungsalgorithmus D_K ein und führt so dazu, daß 50% der Bits des Ausgangsvektors fehlerbehaftet sind.

[9] Wobei p_1 natürlich trotzdem fehlt (denn c_1 wurde ja nicht empfangen), was für praktische Anwendungen schon ein schweres Problem darstellt.

3. Ohne Maßnahmen zur Blocksynchronisation bewirken eingefügte oder verlorene Bits, daß auch alle weiteren Klartext-Blöcke fehlerhaft sind[10].

1.2.2. Cipher Feedback (CFB)

Im Gegensatz zum CBC-Modus (siehe Abschnitt 1.2.1) geht der Klartext bei dieser Betriebsart weder direkt noch indirekt über den Verschlüsselungsalgorithmus E_K.

$$c_i = p_i \oplus E_K(c_{i-1}), \qquad p_i = c_i \oplus E_K(c_{i-1}), \qquad E_K(c_{-1}) = \mathrm{IV} \tag{4}$$

Aus diesem Grund wird die inverse Operation zwar nicht benötigt, die Eigenschaften der CFB-Betriebsart sind aber trotzdem vergleichbar zum CBC:

1. sie ist selbstsynchronisierend;
2. hat eine beschränkte Fehlerfortpflanzung und
3. ist (zusätzlich) für Blockbreiten kleiner der des Algorithmus geeignet.

Die ersten beiden Punkte lassen sich aus den Blockbildern 6 und 7 erkennen oder über Formel 4 verifizieren.

Wegen der Rückführung des Ciphertextes wirkt sich ein Empfangsfehler im Block c_i nur auf den aktuellen Klartext $p_i = c_i \oplus E_K(c_{i-1})$ und den darauffolgenden Block $p_{i+1} = c_{i+1} \oplus E_K(c_i)$ aus. In $p_{i+2} = c_{i+2} \oplus E_K(c_{i+1})$ ist kein Einfluß von c_i gegeben, weshalb (wie im CBC-Modus) ein Fehlerburst von m Bits durch diese Art der Rückführung zu $n+m$ fehlerhaften Bits verbreitert wird.

Blockbreitenreduktion (Fall $r < n$) Die Betriebsart CFB ist auch für Nachrichtenblöcke geeignet, deren Anzahl von Bits r kleiner als die Blockbreite n des Algorithmus ist. Man benutzt in diesem Fall einfach nur die ersten r Output-Bits des Algorithmus, muß jedoch im Eingangsvektor die restlichen $n-r$ Bits auf einen konstanten Wert setzen (in [ISO06b] auf “1”, vgl. Abbildung 8).

Pipelining (Fall $r > n$) Bei Einsatz eines Feedback (FB) Registers im Rückkopplungszweig wird es möglich den Kryptoalgorithmus zu parallelisieren[11]. Man geht dazu wie in Abbildung 9

[10] Dieses Verhalten gilt für alle blockorientierten Verschlüsselungsalgorithmen.
[11] Pipelining läßt sich auch mit dem Verfahren bei reduzierter Blockbreite (vgl. Abbildung 8) kombinieren.

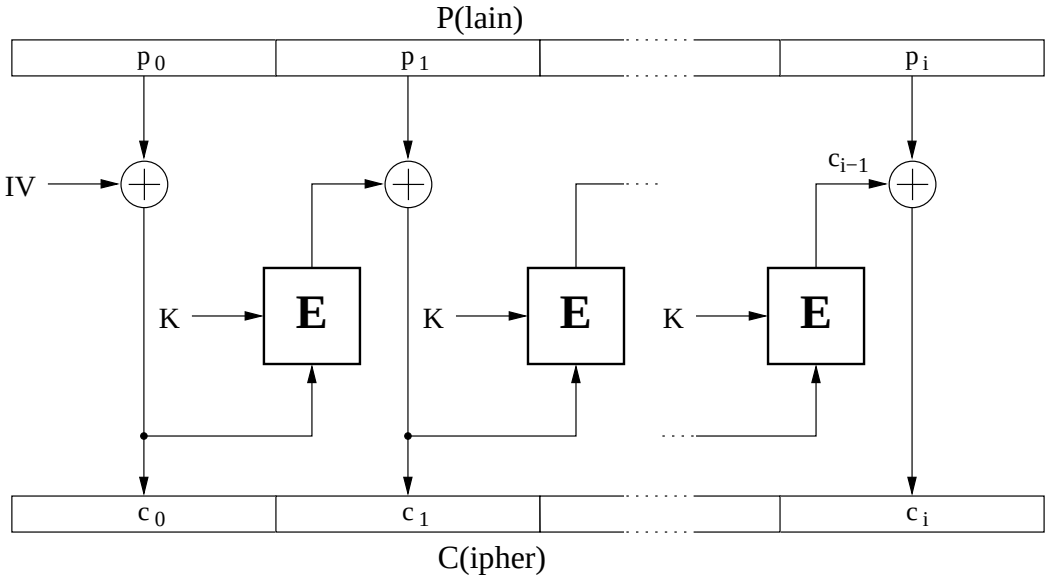


Abbildung 6: CFB-Verschlüsselung

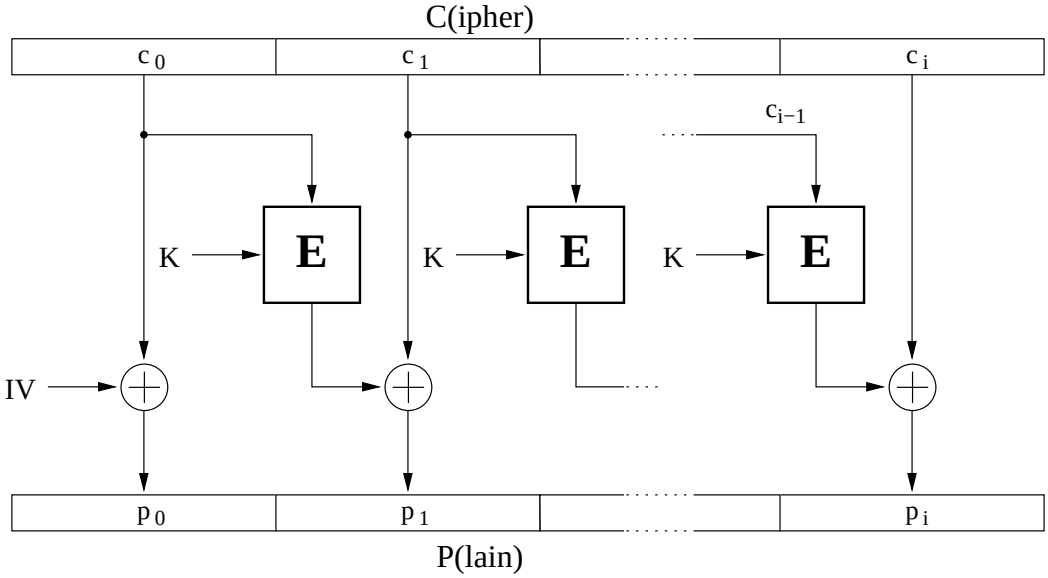


Abbildung 7: CFB-Entschlüsselung

skizziert vor.

Die damit einhergehende Verzögerung verschleppt allerdings auch die Auswirkung von Bitfehlern, was oftmals unerwünscht ist. Denn, sollte ein empfangener Cipherblock fehlerhaft sein, so wirkt sich dies auf den aktuellen und den *k*-ten darauffolgenden Block aus (wenn mit *k* die Tiefe des FB bezeichnet wird).

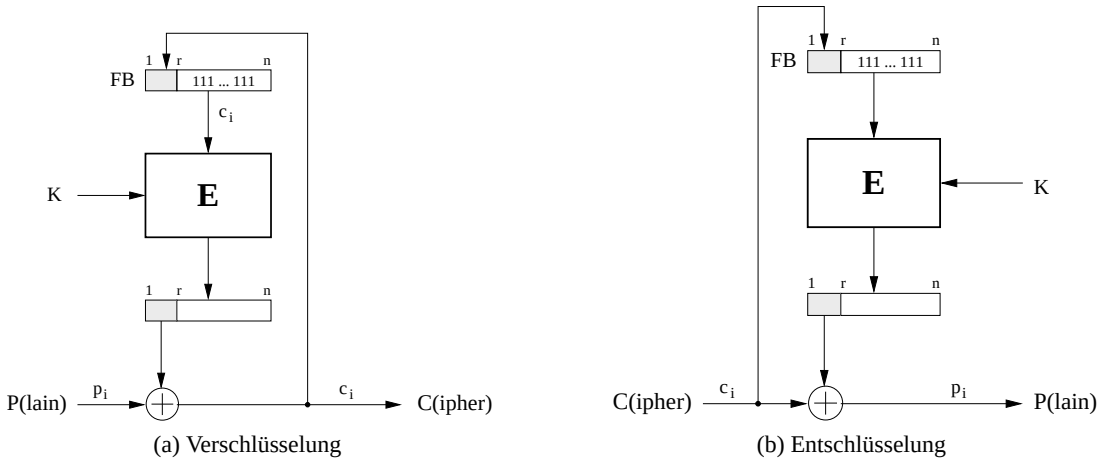


Abbildung 8: CFB mit verringerter Blockbreite

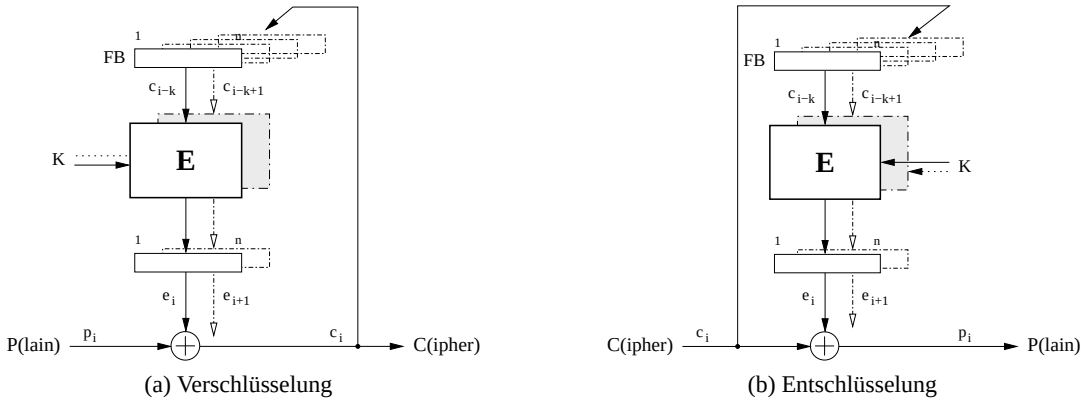


Abbildung 9: CFB mit Pipelining

1.2.3. Output Feedback (OFB)

Auch in der Betriebsart OFB ist man prinzipiell in der Lage Blockbreiten $r < n$ zu verarbeiten, verliert aber die Eigenschaft der Selbstsynchronisation. Das Prinzip dieser Betriebsart besteht darin, daß sowohl auf Sende- als auch Empfangsseite der gleiche (Pseudozufalls-) Strom e_i erzeugt und dann direkt zum Ver- bzw. Entschlüsseln verwendet wird (siehe XOR-Operation in Abbildung 10).

Vorteilhaft ist, daß Bitfehler in einem Cipherblock c_i ohne jegliche Streuwirkung transparent auf den Klartext p_i abgebildet werden – die Fehlerfortpflanzung demzufolge sehr begrenzt ausfällt[12]. Gehen jedoch ganze Blöcke c_i verloren, so ist die Synchronisation bleibend gestört und

[12]Diese Eigenschaft ist beonders wichtig für Anwendungen und Protokolle, die eine bestimmte Quality of Service

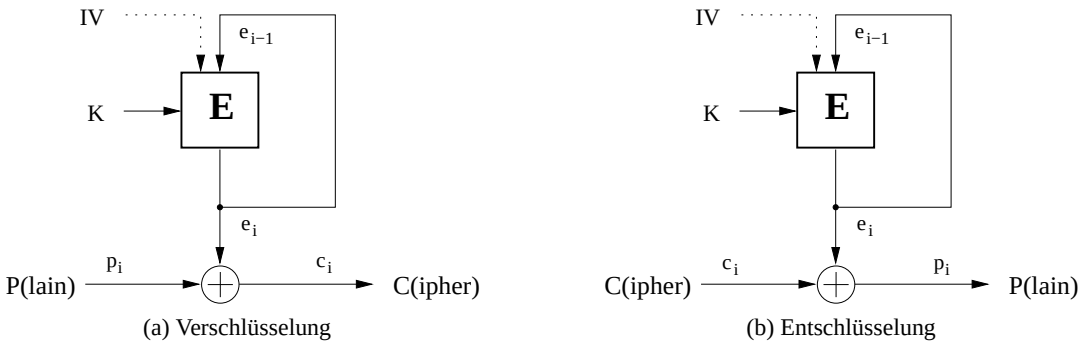


Abbildung 10: Betriebsart OFB

der gesamte Klartext ab diesem Zeitpunkt dauerhaft fehlerhaft[13]. Zur Resynchronisation ist es deshalb unbedingt notwendig, daß von Zeit zu Zeit (je nach Anforderung) ein neuer Initialisierungsvektor in Richtung Empfänger übertragen wird.

1.2.4. Counter (CTR)

Die CTR-Betriebsart[14] arbeitet ähnlich wie der OFB-Modus, nur daß der "Pseudozufall" aus einer einem Zähler mit sehr großer (praktisch nicht erreichbarer) Periode stammt [LRW00]. Das Beispiel in Abbildung 11, welches für ATM gilt (vgl. [AF99, Annex 6.4.4]), soll das Prinzip verdeutlichen.

Entscheidend ist, daß der Counter für jeden zu ver- oder entschlüsselnden Block einen anderen Wert (Zustandsvektor) liefert. Er besteht in diesem Fall aus:

LFSR Das Schieberegister (LFSR) hat eine Länge von 21 Bit und ist durch das irreduzible (und primitive) Generatorpolynom $g(x) = x^{21} + x^2 + 1$ charakterisiert.

I/R Das Initiator/Responder-Bit (I/R) identifiziert den Anrufenden (Calling Party) bzw. Angerufenen (Called Party) und schützt damit vor Angriffen auf schlüsselgleiche Texte.

SEQ Die Sequenznummer (SEQ) wird von höheren Protokollen übernommen (nur AAL-3/4 und AAL-1).

SEG Die Segmentnummer (SEG) identifiziert das ver- bzw. entschlüsselte Segment einer ATM-Zelle.

(QoS) zusichern.

[13] Die Betriebsart OFB wird aus diesem Grund als nicht-selbstsynchronisierend bezeichnet.

[14] Eine Weiterentwicklung der Betriebsart CTR stellt der Galois/Counter Mode (GCM) nach [Dwo07] dar.

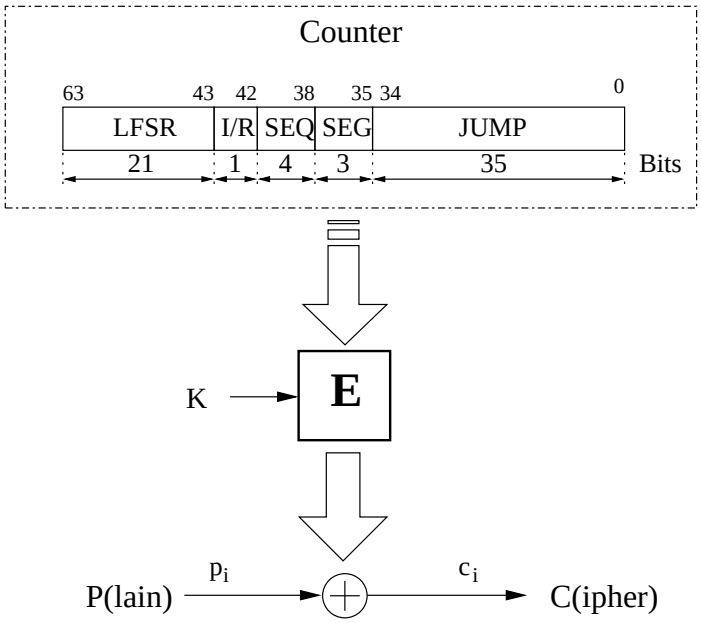


Abbildung 11: ATM Counter-Modus

JUMP Die Jump Number (JUMP) wird bei jeder Session Key Changeover OAM-Zelle und bei einem *End Of Message* (EOM) im AAL-5 erhöht.

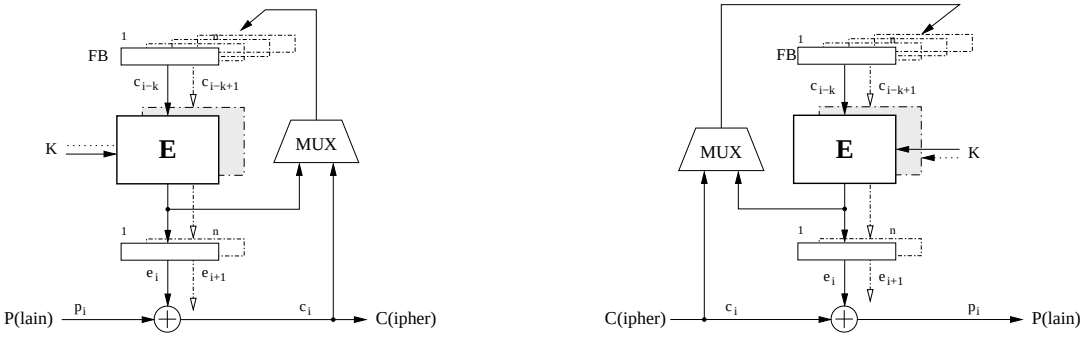
Bezüglich der Synchronisation und Fehlerfortpflanzung gelten dieselben Aussagen wie für die Betriebsart OFB. Vorteilhaft ist der CTR-Modus insbesondere für Hochgeschwindigkeitsanwendungen, denn:

- das Parallelisieren von Kryptooperationen ist möglich, da überhaupt keine Rückkopplung vorhanden ist;
- Ver- und Entschlüsselung verlaufen prinzipiell gleich (solange Synchronisation herrscht), was die Realisierung vereinfacht und Ressourcen spart.

1.2.5. Mischarten

Will man (in Verbindung mit hohen Datenraten) die Eigenschaft der Selbstsynchronisation mit einer geringen Fehlerfortpflanzung kombinieren, so bietet sich eine Mischung von OFB und CFB als Betriebsarten an. Dazu muß man nur die Rückkopplungen beider Betriebsarten (umschaltbar) miteinander vereinen, was in Abbildung 12 durch einen Multiplexer (MUX) realisiert wird.

Algorithmus beschreibt den Entschlüsselungsvorgang, wobei statistische Selbstsynchronisation (siehe z. B. [Rul97] für Details) eine Möglichkeit darstellt um den Zeitpunkt der Umschaltung von einer Betriebsart in die andere zu bestimmen.



(a) Verschlüsselung

(b) Entschlüsselung

Abbildung 12: Kombination von CFB- und OFB-Modus

Algorithmus 1 Synchronisation

if synchron **then**
  for ever **do**
    $c_i = p_i \oplus e_i$
    FB = e_i {OFB-Mode}
  end for
else
  $c_i = p_i \oplus e_i$
  FB = c_i {CFB-Mode}
end if

2. Hash-Algorithmen

2.1. Einführung

Kryptografische Hash-Funktionen bilden das Rückgrat von Kryptoverfahren und -algorithmen [MvV92]. Wie die aus der Informatik bekannten Hash-Funktionen auch, bilden sie eine große Eingangsmenge $\{x\}$ eindeutig (deterministisch) auf eine viel kleinere Ausgangsmenge $y = \mathrm{H}(x)$ ab. Sie haben jedoch folgende ergänzende Eigenschaften, welche deren algorithmische Komplexität mit dem jeweiligen Stand der Technik in Verbindung bringen:

1. Es ist praktisch unmöglich zwei Eingangswerte $x \neq x'$ zu finden (wahlfrei), die denselben Hashwert y besitzen (Kollisionsfreiheit).
2. Der Algorithmus ist eine Einweg-Funktion, d. h. für einen gegebenen Hash-Wert y kann man praktisch keinen zugehörigen Wert $x = \mathrm{H}^{-1}(y)$ berechnen (1. Urbild-Festigkeit).[15]
3. Es ist praktisch unmöglich zu einem vorgegebenen Eingangswert x einen weiteren Wert

[15] Eigentlich existiert H^{-1} nicht einmal (oder nur theoretisch).

$x' \neq x$ zu finden, der denselben Hash-Wert y erzeugt (2. Urbild-Festigkeit).

Insbesondere ältere Hash-Funktionen, wie z. B. der MD5 können dies heute nicht mehr gewährleisten, aber auch der SHA-1 gilt seit 2005 als „geknackt". Alternativen wie SHA-256 oder SHA-512 (auch kurz SHA-2 genannt) stehen zwar zur Verfügung, werden aber noch nicht überall konsequent genutzt.[16]

Wirkprinzip Die Berechnung des Hash-Wertes y erfolgt üblicherweise durch Zerlegung des Eingangs-Vektors x in k Blöcke fester Breite, die dann iterativ mit Hilfe einer nichtlinearen Kompressionsfunktion C verarbeitet werden. Abbildung 13 zeigt die unter dem Namen Merkle/Damgård-Konstruktion bekannte Struktur [Mer90, Dam90].[17] Für jeden Zwischenwert gilt ausgehend von $y_0 = \text{IV}$ die Gleichung $y_i = \text{C}(y_{i-1}, x_i)$ und so für das Ergebnis: $y = \text{H}(x) = y_k$.

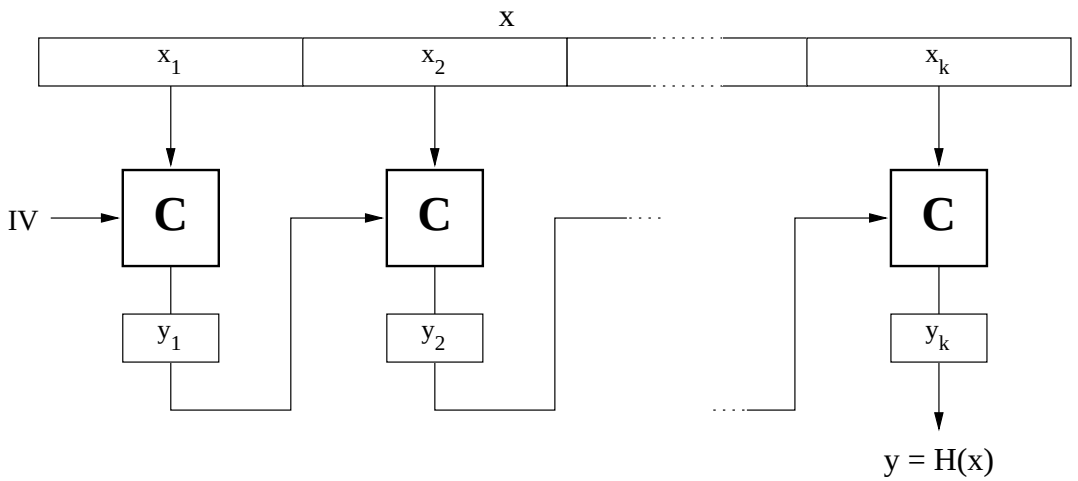


Abbildung 13: Merkle/Damgård-Konstruktion

2.2. Die MD-Familie

Zu den Mitgliedern dieser Hash-Familie, welche von R. Rivest entwickelt wurde, gehören unter anderem der MD2. . . MD5, SHA-1 und SHA-2 [Riv92, NIS08, ISO04]. Am Beispiel des SHA-1 soll deren prinzipielle Arbeitsweise verdeutlicht werden, wofür in Abbildung 14 die zugehörige Kompressionsfunktion C dargestellt ist.

Der SHA-1 ist für 32-Bit Architekturen optimiert, d. h. alle Variablen in Abbildung 14 sind von entsprechender Breite. Die (Eingangs-) Blockbreite x_i beträgt 64 Byte, also 16 Worte zu je 32-Bit. Ein Hash-Wert y_i besteht aus 160-Bit (20 Byte, 5 Worte) und wird iterativ in 80 sogenannten

[16] Da auch die Lebenserwartung dieser Algorithmen begrenzt sein wird, will das NIST bis 2012 den Nachfolger SHA-3 definieren (siehe http://csrc.nist.gov/groups/ST/hash/timeline.html).

[17] Eine gewisse Ähnlichkeit mit den verketteten Betriebsarten der Blockchiffren (vgl. Abschnitt 1.2) ist nicht von der Hand zu weisen.

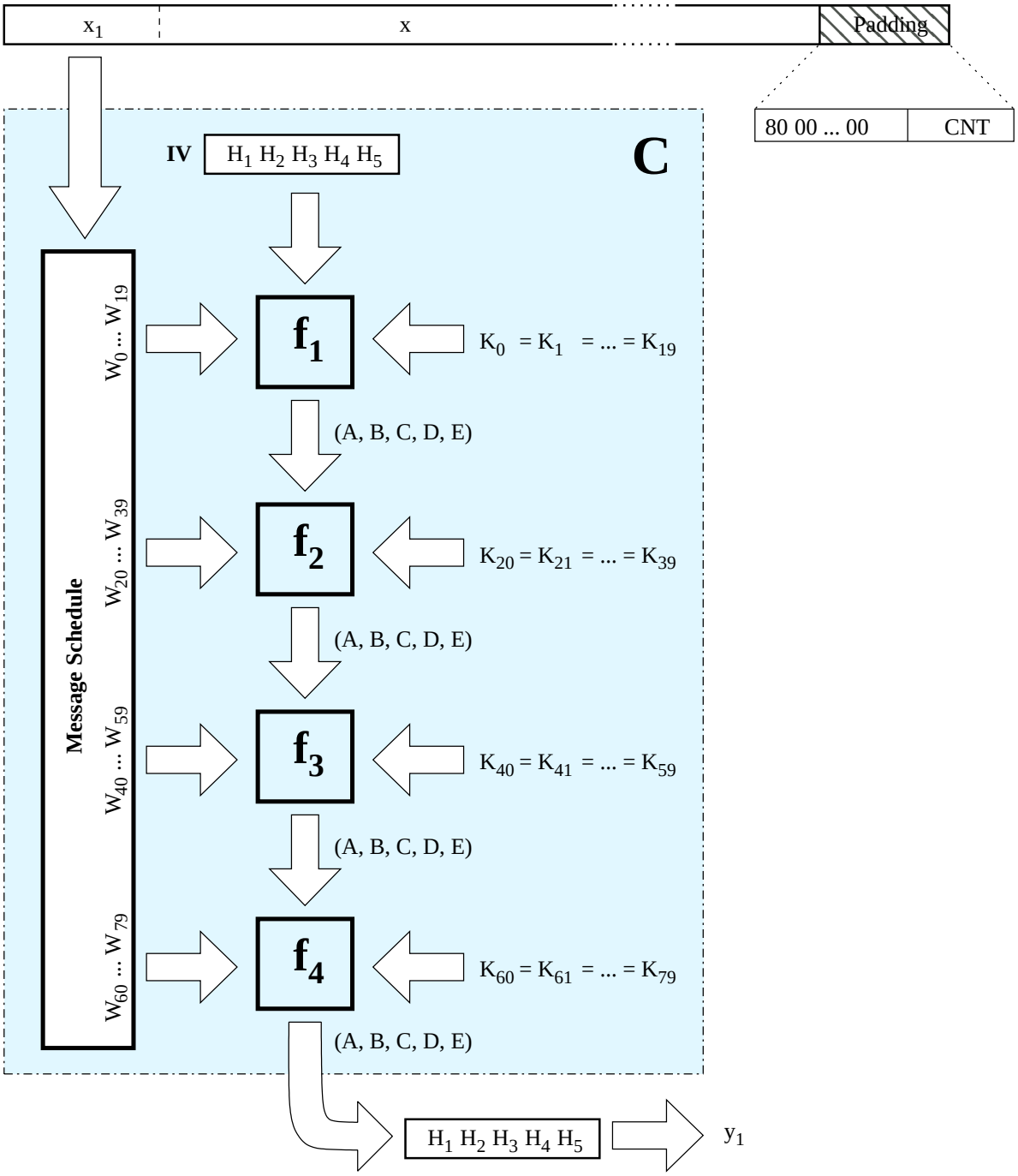


Abbildung 14: Hash-Funktion SHA-1

Runden erzeugt. Speziell beim SHA-1 (nicht so bei SHA-2) werden jeweils 20 Runden mit den gleichen Konstanten und Funktionen realisiert, so daß man die Darstellung auf 4 Funktions-

blöcke $f_1, \ldots, f_4$ reduzieren kann. Der Algorithmus kann folgendermaßen beschrieben werden:

1. Erzeuge im „Message Schedule" aus den 16 Eingangsworten, welche $W_0, \ldots, W_{15}$ zugeordnet werden, iterativ weitere 64 Worte W_i nach folgender Vorschrift:[18]

$$W_i = \mathrm{rol}^1(W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}), \qquad 16 \leq i \leq 79\ .$$

   Darin steht rol für „Rotate Left" und der zugehörige Exponent für die Anzahl der Bits, um die es nach links zu rotieren gilt.

2. Sollte es sich um den ersten Block x_1 handeln, dann initialisiere $H_0, \ldots, H_5$ mit den Konstanten:

$$H_0 = 67452301_{\mathrm{H}} \qquad H_1 = \mathrm{EFCDAB89}_{\mathrm{H}}$$
$$H_2 = 98\mathrm{BADCFE}_{\mathrm{H}} \qquad H_3 = 10325476_{\mathrm{H}}$$
$$H_4 = \mathrm{C3D2E1F0}_{\mathrm{H}}$$

3. Ordne $(H_0, \ldots, H_4)$ dem Vektor (A, B, C, D, E) zu.

4. Berechne 80 Runden $(0 \leq t \leq 79)$, im speziellen Fall von SHA-1 genau $4 \cdot 20$ Runden $(1 \leq n \leq 4)$, nach folgender Vorschrift:

$$\begin{aligned} \mathrm{TEMP} &:= \mathrm{rol}^5(A) + f_n(B, C, D) + E + W_t + K_t \\ E &:= D \\ D &:= C \\ C &:= \mathrm{rol}^{30}(B) \\ B &:= A \\ A &:= \mathrm{TEMP}, \end{aligned}$$

   wobei Additionen immer modulo 2^{32} ausgeführt werden. Die Konstanten K_t sind definiert als:

$$\begin{aligned} K_0 = K_1 \ &= \ldots = K_{19} = 5\mathrm{A827999}_{\mathrm{H}} \\ K_{20} = K_{21} &= \ldots = K_{39} = 6\mathrm{ED9EBA1}_{\mathrm{H}} \\ K_{40} = K_{41} &= \ldots = K_{59} = 8\mathrm{F1BBCDC}_{\mathrm{H}} \\ K_{60} = K_{61} &= \ldots = K_{79} = \mathrm{CA62C1D6}_{\mathrm{H}}\ . \end{aligned}$$

[18] Die iterative Berechnung der W_i eröffnet im Zusammenhang mit deren sequentieller Nutzung die Möglichkeit, nur jeweils 16 Werte vorzuhalten (siehe auch [NIS93, 8. Alternate Method of Computation]).

Die Funktionen $f_1, \ldots, f_4$, welche für jeweils 20 Runden verwendet werden, sind folgendermaßen spezifiziert:

$$
\begin{aligned}
f_1(B,C,D) &= (B \wedge C) \vee (\overline{B} \wedge D) && \text{(Multiplexer)} \\
f_2(B,C,D) &= B \oplus C \oplus D && \text{(Parity)} \\
f_3(B,C,D) &= (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) && \text{(Majority)} \\
f_4(B,C,D) &= B \oplus C \oplus D && \text{(Parity) .}
\end{aligned}
$$

5. Berechne die Ergebniswerte $(H_0, \ldots, H_4)$, welche letztlich den Hash-Wert y_i dieses Blocks ausmachen, wiefolgt:[19]

$$
\begin{aligned}
H_0 &:= H_0 + A & \qquad H_1 &:= H_1 + B \\
H_2 &:= H_2 + C & \qquad H_3 &:= H_3 + D \\
H_4 &:= H_4 + E \ . &&
\end{aligned}
$$

SHA-2 Bei den SHA-2 Hash-Funktionen ist dieses Prinzip erhalten geblieben, die Änderungen liegen im Detail [NIS08]:

- Die Anzahl der Bits im Hash-Wert wurde erhöht und entspricht dem jeweiligen Namen: SHA-224, SHA-256, SHA-384, SHA-512. Dazu gab es zwei wesentliche Modifikationen:[20]

  1. Der Hash-Vektor $(H_0, \ldots, H_4)$ wurde auf $(H_0, \ldots, H_7)$ erweitert, genau wie auch die Zwischenwerte (A, B, C, D, E) auf (A, B, C, D, E, F, G, H).
  2. SHA-384 und SHA-512 sind optimiert für 64-Bit Architekturen, d. h. jedes Wort ist 64-Bit breit und so auch die Blockbreite verdoppelt auf 128 Byte.

- Zur „Rotate Left" Operation sind weitere hinzugekommen: „Rotate Right" und „Shift Left/Right" – insgesamt wurde die Rundenfunktion verkompliziert.
- Innerhalb eines Algorithmus' ist die Rundenfunktion invariant – es wird in jeder Runde aber eine andere Konstante K_t verwendet.
- Beim SHA-224/256 werden anstatt der 80 Runden nur 64 Runden ausgeführt.

[19] Beachte, daß dieser letzte Schritt in Abbildung 14 nicht dargestellt ist.

[20] SHA-224 und SHA-384 sind im Hash-Wert künstlich verkürzte SHA-256 und SHA-512 Varianten (die allerdings mit jeweils anderen Initialwerten für $(H_0, \ldots, H_7)$ starten).

2.3. Padding

Da jede Hash-Funktion für eine fixe (Eingangs-) Blockbreite konzipiert ist, muß man den letzten Block meist auf diese erweitern. Bei den standardisierten Hash-Algorithmen scheint sich folgende Vorgehensweise durchgesetzt zu haben:

1. Füge immer ein einzelnes Bit mit dem Wert 1 an (typisches ISO-Padding), bei Byte-orientierten Implementierung den Wert 80_{H}.
2. Gewährleiste, daß am Ende des letzten Blocks 64 Bit Platz sind für die Aufnahme eines Zählers. Ist dies nicht gegeben, schließe den aktuellen Block durch Auffüllen mit Nullen ab und verarbeite ihn mit der Kompressionsfunktion.
3. Fülle den aktuellen Block, ausgenommen die letzten 64 Bit, mit Nullen auf.
4. Füge einen Blockzähler (`CNT` in Abbildung 14) in die letzten 64 Bit ein und verarbeite diesen Block mit der Kompressionsfunktion.
5. Nehme das Ergebnis y_k dieses letzten Blocks als Hash-Wert $\mathrm{H}(x)$.

3. Asymmetrische Algorithmen

3.1. RSA

Beim RSA-Algorithmus handelt es sich um das bekannteste asymmetrische Kryptoverfahren, benannt nach seinen Erfindern Rivest-Shamir-Adleman [RSA78, AR78].[21] Die mathematische Grundlage bilden Restklassenkörper, verbunden mit der Schwierigkeit große Zahlen zu faktorisieren.

3.1.1. Schlüsselgenerierung

Betrachten wir in der Voraussetzung zuerst die Schlüsselerzeugung, welche im Wesen folgendermaßen abläuft:[22]

1. Wähle zwei große Primzahlen $p, q > 2$, mit $p \neq q$, welche das öffentliche Modul $n = pq$ bestimmen.

   Zahlentheoretische Erläuterungen:

[21] Der Algorithmus wurde in [PKC93] zu einem „Quasi"-Standard erhoben und ist heute fester Bestandteil vieler internationaler Normen und Standards [ISO06a, ISO02a, IEE00, JK03, AF99].

[22] Das Schlüsselmaterial bietet im allgemeinen das größte Angriffspotential. Deshalb sollte man Anforderungen an RSA-Schlüssel, wie sie z. B. in [EBS09, NIS09, IEE00], [MvV92, 8.2.2] und [Wel01, 16.2] formuliert sind, unbedingt berücksichtigen.

- Bei der Restklasse $\mathbb{Z}_n := \{0, 1, \ldots, n-1\}$ handelt es sich um einen Ring, nicht um einen Körper.[23]
- Im Gegensatz dazu bilden $\mathbb{Z}_p$ und $\mathbb{Z}_q$ jeweils einen Primzahlenkörper.
- Läßt man aus $\mathbb{Z}_n$ nur solche Elemente zu, die keinen gemeinsamen Teiler mit dem Modul n haben, wird ein Körper konstruierbar. Die Anzahl der Elemente r in dessen multiplikativer Gruppe $\mathbb{Z}_n^* := \{r \in \mathbb{Z} | 0 < r < n, \gcd(r, n) = 1\}$ ergibt sich mit Hilfe von Euler's Totient-Funktion zu $\varphi = \phi(n) = (p-1)(q-1)$.

2. Nun wird eine Zufallszahl $1 < e < \varphi$ erzeugt, die keinen gemeinsamen Faktor mit φ hat. Diese (auch als öffentlicher Exponent bezeichnete) Zahl bildet die Basis des öffentlichen Schlüssels (n, e).

   Zahlentheoretische Erläuterungen:

   - Wegen $\gcd(e, \varphi) = 1$ gehört e zum endlichen Körper $\mathbb{Z}_\varphi$.
   - Aus $\gcd[e, (p-1)(q-1)] = 1$ läßt sich als notwendige Voraussetzungen $\gcd(e, p-1) = 1$ und $\gcd(e, q-1) = 1$ ableiten.
   - Da $p-1$ und $q-1$ gerade Zahlen sind, muß e wegen des vorangegangenen Punktes auf jeden Fall ungerade sein.

3. Für den privaten Schlüssel (n, d) wird jetzt eine weitere Zahl d so berechnet,[24] daß $de - 1$ ohne Rest durch φ teilbar ist.

   Zahlentheoretische Erläuterungen:

   - Ausgehend von

$$de \equiv 1 \pmod{\varphi} \tag{5}$$

     kann man d auch als Inverse des öffentlichen Exponents e im Körper $\mathbb{Z}_\varphi$ ansehen.

$$d \equiv e^{-1} \pmod{\varphi}$$

3.1.2. Algorithmus

Für die Verschlüsselung wird folgende Operation auf dem Plaintext-Block m ausgeführt, wobei dieser als Zahl $m \in \mathbb{Z}_n$ interpretiert wird:

[23] Das Modul n kann als Produkt zweier ungerader Zahlen allenfalls als ungerade vorausgesetzt werden, jedoch nicht als Primzahl.

[24] Zum Beispiel mit Hilfe des erweiterten euklidischen Algorithmus.

$$c = m^e \bmod n \tag{6}$$

Die inverse Operation der Entschlüsselung wird in gleicher Art und Weise vorgenommen:

$$m' = c^d \bmod n\,. \tag{7}$$

Für den Beweis $m' = m$ wendet man zuerst Gleichung 6 an und bezieht dann die Restklassendarstellung $de = 1 + k\varphi$ ($k \in \mathbb{N}$) ein.

$$\begin{aligned} m' &= (m^e \bmod n)^d \bmod n \\ &= m^{de} \bmod n \\ &= m \cdot m^{\varphi k} \bmod n \\ &= m(m^\varphi)^k \bmod n \\ &= m(m^\varphi \bmod n)^k \bmod n \end{aligned}$$

Die weitere Argumentation beruht darauf, daß $m^\varphi \bmod n = 1$ gilt und so:

$$m' = m(\underbrace{m^\varphi \bmod n}_{1})^k \bmod n = m \bmod n = m\,.$$

Besitzen m und n keinen gemeinsamen Teiler ($\gcd(m,n) = 1$, $m \in \mathbb{Z}_n^*$), dann kann man auf $m^\varphi \bmod n$ einfach Euler's Satz (nach Formel 54) anwenden und ist fertig.

Hat m allerdings gemeinsame Teiler mit n, dann gilt $\gcd(m,n) \neq 1$ und deshalb $m \notin \mathbb{Z}_n^*$. Betrachtet man aber die Faktorisierung von n, dann muß m ein Vielfaches von p oder q sein (wegen $m < n$ jedoch nicht von pq). Unter dieser Voraussetzung wäre die Zerlegung $m = \overline{m}p$ oder $m = \overline{m}q$ möglich, wobei jeweils $\gcd(\overline{m},n) = 1$ gilt. Im Fall $m = \overline{m}p$ (für $m = \overline{m}q$ ganz genauso) läßt sich die Modulo-Division $m^\varphi \bmod n$ durch Kürzen von p und unter Berücksichtigung von $\overline{m} < q$ folgendermaßen vereinfachen:

$$m^\varphi \bmod n = (\overline{m}p \bmod pq)^\varphi \bmod n = (\overline{m} \bmod q)^\varphi \bmod n = \overline{m}^\varphi \bmod q \bmod n = \overline{m}^\varphi \bmod q\,.$$

Wegen $\overline{m} \in \mathbb{Z}_q^*$ kann nun wieder der Satz von Euler zur Anwendung kommen, was den Beweis vervollständigt.

3.1.3. Optimierung

Eine Beschleunigung des Verfahrens läßt sich (algorithmisch) vor allem beim Entschlüsseln erzielen.[25] Geht man dazu von

$$m' = c^d \bmod n = c^d - lpq, \qquad l \in \mathbb{N}$$

aus, dann lassen sich (durch Modulo-Division nach p und q) die folgenden zwei Kongruenzen formulieren:

$$m' \equiv c^d \pmod{p} \qquad\qquad m' \equiv c^d \pmod{q}\,.$$

Diese legen eine Anwendung des Chinesischen Restsatzes entsprechend Anhang D.3.1 nahe [Gro00, Wel01].[26]

$$\begin{aligned} m_p &= c^d \bmod p & m_q &= c^d \bmod q \\ m' &\equiv m_p \pmod{p} & m' &\equiv m_q \pmod{q} \end{aligned}$$

Als Voraussetzung benötigt man die Koeffizienten α und β in der Bézout-Darstellung des größten gemeinsamen Teilers (vgl. Formel 60 in Anhang D.1.1)

$$\gcd(p,q) = \alpha p + \beta q = 1,$$

welche z. B. mit Hilfe des erweiterten euklidischen Algorithmus berechnet werden können.[27] Sie stellen, wenn man vorangegangene Gleichung nach p und q modulo-dividiert, gleichzeitig die Inversen

$$\alpha \equiv p^{-1} \pmod{q} \qquad\qquad \beta \equiv q^{-1} \pmod{p}$$

im Körper $\mathbb{Z}_q$ und $\mathbb{Z}_p$ dar. Durch Anwendung des Chinesischen Restsatzes für zwei Kongruenzen kann man nun m' berechnen:

[25] Eine simple Methode den Rechenaufwand beim Verschlüsseln zu reduzieren ist, den öffentlichen Exponent e als verhältnismäßig kleine Zahl zu wählen (im einfachsten Fall zu 3 oder z. B. nach [ITU97, Annex D.6] als Fermat-Zahl $F_4 = 65537$).

[26] Eine Voraussetzung für dessen Anwendung ist: $\gcd(p,q) = 1$, was bei der Wahl von n als Produkt zweier Primzahlen gegeben ist.

[27] Die Anwendung des euklidischen Algorithmus' stellt im allgemeinen eine rechenintensive Operation dar (siehe dazu auch Anhang D.1) . Glücklicherweise können α und β aber im voraus berechnet werden.

$$m' = (\alpha m_q p + \beta m_p q) \bmod n\,. \tag{8}$$

Der Vorteil liegt darin, daß die modulare Exponentiation modulo n, welche die Komplexität $O(\log^3 n)$ hat,[28] auf zwei Operationen gleicher Art, aber mit verringerter Anzahl von Stellen verteilt.

Die weitere Optimierung kann in drei Schritten erfolgen:

1. Wendet man den Chinesischen Restsatz in der Variante nach H. L. Garner an, so werden weitere Vereinfachungen möglich [Gar59]. Denn aus der Restklassendarstellung

   $$m' = m_q + xq \qquad\qquad m' = m_p + yp, \qquad x,y \in \mathbb{Z}$$

   läßt sich eine lineare diophantische Gleichung ableiten (siehe auch Anhang D.2):

   $$m_p - m_q = xq - yp,$$

   welche die Lösungsmenge

   $$x_k = \beta(m_p - m_q) + kp \qquad\qquad y_k = -\alpha(m_p - m_q) - kq, \qquad k \in \mathbb{Z}$$

   besitzt. Üblicherweise benutzt man zur Berechnung von m' die Lösungen für x_k und erhält in geschlossener Darstellung:[29]

   $$m' = m_q + q\beta(m_p - m_q) + kn\,.$$

   Unter der Voraussetzung $m' = m' \bmod n$ kann man vorangegangene Gleichung noch weiter reduzieren, wenn die allgemeingültige Beziehung $(rq) \bmod (pq) = q(r \bmod p)$ berücksichtigt wird.

$$m' = m_q + q[\beta(m_p - m_q) \bmod p] \tag{9}$$

   Die Vorteile dieser Darstellung liegen vor allem darin, daß

[28] Der Ausdruck $\log n$ stellt ein Maß für die Anzahl der notwendigen Bits zur Repräsentation von n dar.

[29] An dieser Stelle muß man darauf hinweisen, daß für den Ausdruck $q\beta \neq 1$ gilt (denn $q\beta \equiv 1$ bezieht sich ausschließlich auf den Körper $\mathbb{Z}_p$).

- eine finale Modulo-Division durch n nicht mehr nötig ist;
- nur ein Bézout-Kofaktor überhaupt benötigt wird;
- wegen der Zwischenreduktion modulo p die Speicherplatzanforderungen reduziert sind.

2. Man kann aber noch weitergehen, indem die Berechnung von m_p und m_q vereinfacht wird. Eine wesentliche Zeitersparnis kommt zustande, wenn man c vor dessen Potenzierung in der Länge reduziert.

$$\begin{aligned} m_p &= c^d \bmod p \\ &= (c \bmod p)^d \bmod p \end{aligned} \qquad \begin{aligned} m_q &= c^d \bmod q \\ &= (c \bmod q)^d \bmod q \end{aligned}$$

3. Eine letzte Optimierung bezieht sich auf den Exponenten d, welcher ebenfalls reduziert werden kann. Dazu muß man sich nur klarmachen, daß durch die Reduktion von c in Punkt 2 das Ergebnis der Potenzierung jeweils in $\mathbb{Z}_p$ oder $\mathbb{Z}_q$ und insbesondere in der zugehörigen multiplikativen Gruppe liegt. Da diese zyklisch ist, wiederholen sich beim Potenzieren die Werte mit der Gruppenordnung $|\mathbb{Z}_p^*| = p-1$ bzw. $|\mathbb{Z}_q^*| = q-1$. Aus diesem Grund kann man d auf die Gruppenordnung reduzieren (vgl. auch Anhang B.1).

$$\begin{aligned} m_p &= (c \bmod p)^d \bmod p \\ &= c^{d \bmod (p-1)} \bmod p \end{aligned} \qquad \begin{aligned} m_q &= (c \bmod q)^d \bmod q \\ &= c^{d \bmod (q-1)} \bmod q \end{aligned}$$

Zu der privaten Schlüsseldarstellung (n, d) gibt es deshalb zwei Optionen:

- ein Tripel (p, q, d), was wegen $n = pq$ trivial erscheint;
- oder die Darstellung als Quintupel $(p, q, d_p, d_q, q_{\text{inv}})$, mit $d_p = d \bmod (p-1)$, $d_q = d \bmod (q-1)$ und $q_{\text{inv}} = q^{-1} \pmod p$.

Wegen der signifikanten Geschwindigkeitsvorteile wird fast immer die Quintupel-Variante verwendet [IEE00, ISO06a, PKC02].[30]

3.1.4. Verschlüsselung nach PKCS #1

Um den RSA-Algorithmus praktisch auf die Verschlüsselung von Daten anzuwenden, müssen diese zuerst geeignet aufbereitet werden. Der Prozeß dieser Formatierung wird in [IEE00, PKC02] als *Message Encoding Operation* bezeichnet, in [PKC93] hingegen als *Encryption Block Formatting*.

[30] In [MvV92, 14.5.2] wird auf der Grundlage einer kurzen Komplexitätsbetrachtung die Beschleunigung mit dem Faktor 4 ausgewiesen.

PKCS #1 v1.5 Mit der Methode nach [PKC93] gestaltet sich die Formatierung noch relativ einfach.[31] Der Datenblock D wird entsprechend Abbildung 15 in die Struktur des sogenannten *Encryption Block* EB eingebunden. Auf den *Encryption Block* EB wird letztlich der RSA-Algorithmus nach Formel 6 angewendet.[32]

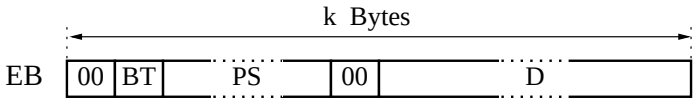


Abbildung 15: Blockformatierung nach PKCS #1 v1.5

Die weiteren Elemente im *Encryption Block* haben folgende Bedeutung:

BT Der *Block Type* zeigt die Verwendung des privaten oder öffentlichen Schlüssels (im anschließenden RSA-Algorithmus) an.

- 00_H Signieren (Verwendung des privaten Schlüssels);
- 01_H genauso wie 00_H (der Unterschied liegt im Element PS);
- 02_H Verschlüsseln (Verwendung des öffentlichen Schlüssels).

PS Ein *Padding String* dient der Anpassung an die Länge des Moduls $k = ||n||$ so, daß $||\text{PS}|| + ||D|| + 3 = k$ gewährleistet ist. Die Länge von PS soll mindestens 8 Byte sein, was im Umkehrschluß die von D auf $k - 11$ Bytes beschränkt. Das Padding selbst hängt vom Typ des Blocks (BT) ab:

- 00_H alle Bytes sind 00_H;
- 01_H alle Bytes sind FF_H;
- 02_H alle Bytes sind zufällig, aber ungleich 00_H.

Der Inhalt von PS ist beim Entschlüsseln (Byte für Byte) auf konforme Kodierung zu prüfen [PKC93, 9.4]. Für die Blocktypen 01_H und 02_H kann der Anfang (und damit auch die Länge) des Datenblocks D ermittelt werden, indem das 00_H-Byte zwischen PS und D gesucht wird. Für den Blocktyp 00_H ist dieses Verfahren nur geeignet, wenn das erste Byte in D immer ungleich 00_H ist. Kann dies nicht vorausgesetzt werden, dann muß die Länge ||D|| a priori bekannt sein.[33]

PKCS #1 v2.1 Nicht zuletzt wegen des sehr erfolgreiche Angriffs nach [Ble98] wird die PKCS #1 v1.5 Formatierung heute nicht mehr empfohlen. Statt dessen sollte grundsätzlich Ver-

[31] Die Formatierung nach PKCS #1 v1.5 wurde in PKCS #1 v2.1 als Verfahren RSAES-PKCS1-v1_5 übernommen [PKC02].

[32] Das führende 00_H-Byte's in EB gewährleistet für den (als Zahl interpretierten) *Encryption Block* EB $< n$ bzw. EB $\in \mathbb{Z}_n$.

[33] Der Blocktyp 00_H wird im *Privacy Enhanced Mail* (PEM) RFC nach [Ken93] nicht verwendet.

sion 2.1 nach [PKC02, IEE00, ISO06a] zum Einsatz kommen, welche *Optimal Asymmetric Encryption Padding* (OAEP) verwendet [BR95].[34]

Den Kern der Blockformatierung nach Abbildung 16 macht eine sogenannte *Mask Generation Function* (MGF, in Abbildung 16 mit `F` bezeichnet) aus, welche selbst wiederum eine Hash-Funktion H verwendet. Die MGF kann dasselbe leisten wie die Hash-Funktion, nämlich eine große Eingangsmenge auf eine kleine Ausgangsmenge abbilden (mit ||H|| soll im folgenden die Breite eines Hash-Wertes bezeichnet sein). Sie kann aber auch das Umgekehrte – eine kleine Eingangsmenge auf eine größere Ausgangsmenge „zerstreuen". PKCS #1 v2.1 legt sich zwar in der Hash-Funktion nicht fest, definiert aber in [PKC02, B.2.1] eine konkrete *Mask Generation Function*, genannt MGF1.[35]

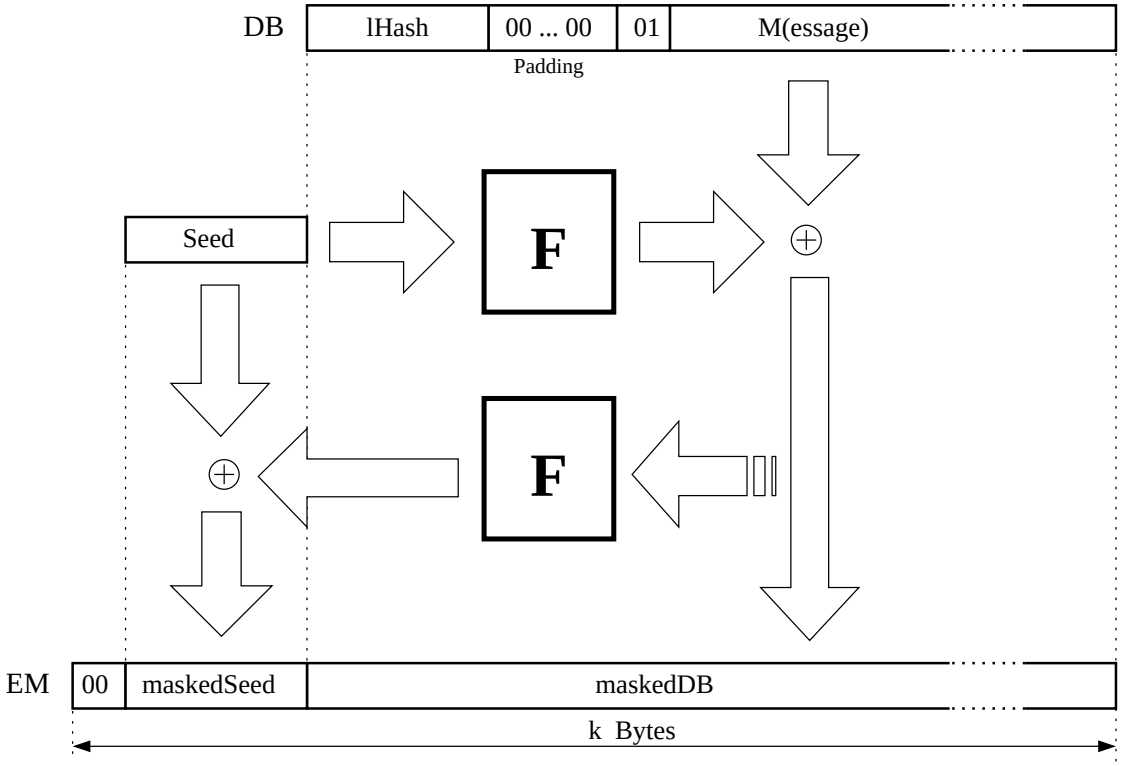


Abbildung 16: Blockformatierung nach PKCS #1 v2.1

Die einzelnen Kodierungsschritte können folgendermaßen beschrieben werden:

1. Ergänze die Nachricht `M` zuerst (und grundsätzlich) durch ein vorangestelltes Byte mit

[34] Ursprünglich galt das von M. Bellare und P. Rogaway entwickelte OAEP (gegen *Chosen Ciphertext Attacks*, im Rahmen des *Random Oracle Models*) als beweisbar sicher. Im Jahre 2001 konnte jedoch V. Shoup einen Fehler in den Ausführungen von [BR95] nachweisen, weshalb OAEP heute „nur" noch als sehr sicher angesehen wird.

[35] Sowohl [PKC02] als auch [IEE00] lassen vom Spezifikationsansatz die Verwendung anderer MGF's zu, nur [ISO06a] definiert aber eine solche (KDF2).

Wert 01_H.

2. Füge weitere 00_H-Byte's hinzu (Padding), so daß ein Datenblock `DB` der Länge $k - ||H|| - 1$ entsteht.[36]
3. Bilde den Hash über ein vereinbartes, konstantes Label L und stelle das Ergebnis als lHash = H(L) an den Anfang des Datenblocks.[37]
4. Erzeuge einen Zufallsstrom `Seed` der Länge ||H||.
5. Berechne maskedDB = DB $\oplus$ MGF(Seed) als Teil der *Encoded Message* (`EM`).
6. Berechne maskedSeed = Seed $\oplus$ MGF(maskedDB), als höherwertigen Teil von `EM`.
7. Stelle ein führendes 00_H-Byte an den Anfang von `EM` (gewährleistet auch hier wieder EM $< n$).

Im Gegensatz zu PKCS #1 v1.5 geht in die RSA-Verschlüsselung $c = \text{EM}^e \bmod n$ hier nun ein Wert EM ein, der keinerlei Rückschlüsse auf den Datenblock `DB` zuläßt.[38]

3.2. Diffie-Hellmann Schlüsselaustausch

Das Verfahren von Diffie und Hellmann nutzt das Problem des diskreten Logarithmus[39] um einen Schlüsselaustausch zu realisieren. Voraussetzung dafür ist eine große Primzahl p (öffentliches Modul) und ein Generator g mit $0 < g < p$, welche beiden Kommunikationspartnern bekannt sind. Der Austausch der Session Keys erfolgt folgendermaßen:

1. Zuerst erzeugt der Initiator des Protokolls ein (geheimes) Element a mit $0 < a < p - 1$ und berechnet damit seine öffentlichen Schlüssel $A = g^a \bmod p$.
2. Dieser Schlüssel A wird im Klartext zum Empfänger (Responder) übertragen.[40]
3. Der Empfänger erzeugt ebenfalls eine Zufallszahl b und berechnet $B = g^b \bmod p$.
4. Anschließend überträgt er $B = g^b \bmod p$ als (unverschlüsselte) Antwort zum Initiator.
5. Beide Partner berechnen dann mit Hilfe von A bzw. B den Sitzungsschlüssel $g^{ab} \bmod p$.
   - Der Initiator berechnet $B^a = (g^b \bmod p)^a \bmod p = g^{ab} \bmod p$.
   - Auf der anderen Seite wird äquivalent $A^b = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p$.

[36] Im Gegensatz zu PKCS #1 v1.5 ist keine Mindestanzahl vorgeschrieben (das Padding kann sogar leer sein).
[37] Praktisch ist L oftmals einfach leer, was für das Hashing kein Problem darstellt.
[38] In PKCS #1 v1.5 waren zumindest `BT` und (teils) `PS` vorherzusagen, was sich dann auch D. Bleichenbacher in seiner *Chosen Ciphertext Attack* zunutze machte [Ble98].
[39] Genauer gesagt das sogenannte Diffie-Hellmann Problem, welches sich durch das Produkt im Exponenten g^{ab} vom Problem des diskreten Logarithmus unterscheidet.
[40] Unverschlüsselt, deshalb sollte das Verfahren immer in Verbindung mit einem Signaturalgorithmus verwendet werden (zum Schutz vor Man-in-the-middle Angriffen).

3.3. Elliptische Kurven

Elliptische Kurven werden in der Normalform nach WEIERSTRASS durch Gleichungen der Form

$$y^2 = x^3 + ax + b$$

definiert (siehe Abbildung 17). Wegen der (nur) zwei freien Parameter a und b ist jede elliptische Kurve durch zwei Punkte P_1, P_2 eindeutig definiert. Insofern ist jeder weitere Punkt durch die Kenntnis von P_1 und P_2 festgelegt.

$$\begin{aligned} y_1^2 &= x_1^3 + ax_1 + b \\ y_2^2 &= x_2^3 + ax_2 + b \end{aligned}$$

Für a ergibt die Differenz beider Gleichungen

$$\begin{aligned} y_2^2 - y_1^2 &= x_2^3 - x_1^3 + a(x_2 - x_1) \\ a &= \frac{y_2^2 - y_1^2 - (x_2^3 - x_1^3)}{x_2 - x_1} \end{aligned}$$

was dann zur Berechnung von b verwendet wird

$$b = y_1^2 - x_1^3 - ax_1 \tag{10}$$

$$= \frac{x_2(y_1^2 - x_1^3) - x_1(y_2^2 - x_2^3)}{x_2 - x_1} \tag{11}$$

Zieht man nun durch P_1, P_2 eine Gerade so berechnet sich ein dritter Punkt $P_3' = (x_3, y_3')$ nach der Geradengleichung

$$\frac{y_3' - y_2}{x_3 - x_2} = \frac{y_2 - y_1}{x_2 - x_1} \tag{12}$$

$$y_3' = \frac{y_2 - y_1}{x_2 - x_1}(x_3 - x_2) + y_2 = \frac{y_2 - y_1}{x_2 - x_1}(x_3 - x_1) + y_1 \tag{13}$$

$$= \frac{y_2 - y_1}{x_2 - x_1}x_3 + \frac{y_1x_2 - y_2x_1}{x_2 - x_1} \tag{14}$$

$$= \lambda x_3 + \frac{y_1x_2 - y_2x_1}{x_2 - x_1} \tag{15}$$

mit

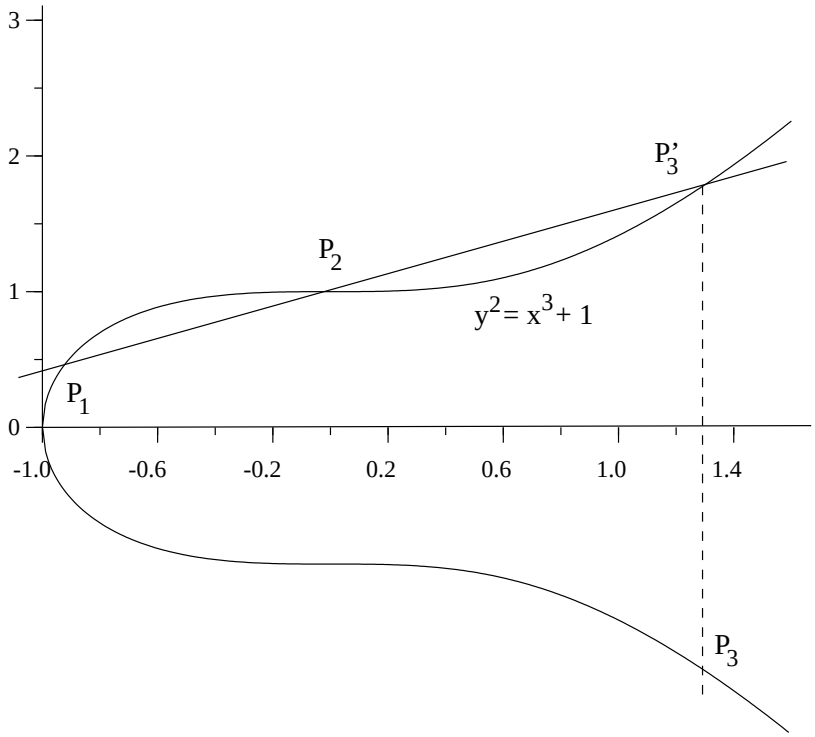


Abbildung 17: Prinzip der Elliptische Kurve

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = \frac{y_3' - y_1}{x_3 - x_1} = \frac{y_3' - y_2}{x_3 - x_2}$$

Den Schnittpunkt P_3' mit der elliptischen Kurve

$$\begin{aligned} y_3'^2 = y_3^2 &= x_3^3 + ax_3 + b \\ &= x_3^3 + ax_3 + y_1^2 - x_1^3 - ax_1 \\ &= x_3^3 - x_1^3 + a(x_3 - x_1) + y_1^2 \end{aligned}$$

findet man durch Gleichsetzen[41]

[41] Unter Zuhilfenahme von $a^3 - b^3 = (a-b)(a^2 + ab + b^2)$

$$
\begin{aligned}
x_3^3 - x_1^3 + \lambda(x_3 - x_1)\frac{y_2^2 - y_1^2 - (x_2^3 - x_1^3)}{y_2 - y_1} + y_1^2 &= [\lambda(x_3 - x_1) + y_1]^2 \\
x_3^3 - x_1^3 + \lambda(x_3 - x_1)(y_2 + y_1) - (x_3 - x_1)(x_2^2 + x_2x_1 + x_1^2) + y_1^2 &= \lambda^2(x_3 - x_1)^2 + 2\lambda y_1(x_3 - x_1) + y_1^2 \\
x_3^2 + x_1x_3 + x_1^2 + \lambda(y_2 + y_1) - x_2^2 - x_2x_1 - x_1^2 &= \lambda^2(x_3 - x_1) + 2\lambda y_1 \\
x_3^2 - x_2^2 + x_1x_3 + \lambda(y_2 - y_1) - x_2x_1 &= \lambda^2(x_3 - x_1) \\
x_3^2 - x_2^2 + x_1x_3 + \lambda^2(x_2 - x_1) - x_2x_1 &= \lambda^2(x_3 - x_1) \\
(x_3 - x_2)(x_3 + x_2) + x_1(x_3 - x_2) - \lambda^2(x_3 - x_2) &= 0 \\
x_3 + x_2 + x_1 - \lambda^2 &= 0 \\
x_3 &= \lambda^2 - x_2 - x_1
\end{aligned}
$$

Aus der Geradengleichung ist nun auch y_3' berechenbar

$$
\begin{aligned}
y_3' &= \lambda(x_3 - x_1) + y_1 \\
&= \lambda(\lambda^2 - x_2 - 2x_1) + y_1 \\
&= \lambda^3 - \lambda(x_2 + 2x_1) + y_1
\end{aligned}
$$

Anwendbarkeit Der kryptologische Hintergrund ist nun folgender. Man definiert die Addition zweier Punkte P_1, P_2 zu $P_3 = P_1 + P_2$ so, daß P_3 (wie in Abbildung 17 dargestellt) wieder auf der elliptischen Kurve liegt[42]. Die Menge aller dieser Punkte P_3 soll eine Abelsche Gruppe $(G, +)$ bilden, d.h. es müssen folgende Bedingungen erfüllt sein.

1. Die Addition von $P_1, P_2 \in G$ führt in jedem Fall zu $P_3 \in G = P_1 + P_2$. Sie ist folgendermaßen erklärt:

   a) Normalfall

$$
\begin{aligned}
x_3 &= \lambda^2 - x_2 - x_1 \\
-y_3 &= \lambda(x_3 - x_1) + y_1
\end{aligned}
$$

   b) Für $P_1 = P_2 = P = (x, y)$ (der Anstieg $\frac{y_2 - y_1}{x_2 - x_1} \to \infty$) wird die Tangente angelegt, d.h. λ entspricht der ersten Ableitung der elliptischen Kurve am Punkt P

$$
\lambda = \frac{1}{2}\frac{3x^2 + a}{\sqrt{x^3 + ax + b}} = \frac{3x^2 + a}{2y}
$$

[42] Diese Operation kann man auf dem Körper der rationalen Zahlen $\mathbb{Q}$ oder z.B. auf der Menge der ganzen Zahlen modulo einer Primzahl $\mathbb{F}_l$ definieren (der kryptologisch interessantere Fall).

und es gilt

$$\begin{aligned} x_3 &= \left(\frac{3x^2+a}{2y}\right)^2 - 2x \\ y_3 &= \frac{3x^2+a}{2y}(x-x_3) - y \end{aligned}$$

c) Im Fall $x_1 = x_2$ und $y_2 = -y_1$ wird $P_3 = (0,0)$ definiert.

2. Die Addition sei assoziativ: $P_1 + (P_2 + P_3) = (P_1 + P_2) + P_3$ für $P_1, P_2, P_3 \in G$,
3. Es gibt das neutrale Element $O \in G$ mit $P + O = O + P = P$ für $P \in G$. Nimmt man an, daß $O = (x_1, y_1)$ ist, dann muß die Forderung $P_3 = P_2$ gestellt werden, um (x_1, y_1) zu ermitteln. Dazu verwendet man $x_3 = \lambda^2 - x_2 - x_1$ mit $\lambda = \frac{y_3 - y_2}{x_3 - x_2} \to \infty$

$$\begin{aligned} x_1 &= \lim_{\lambda \to \infty} \lambda^2 - 2x_2 \\ x_1 &= \infty \end{aligned}$$

und damit $y_1 = \lim_{\lambda, x_1 \to \infty} \lambda(x_1 - x_2) - y_2 = \infty$. Das neutrale Element ist damit $O = (\infty, \infty)$, was auch geometrisch einleuchtend ist.

4. Zu jedem beliebigen Element $P \in G$ findet man $-P \in G$, so daß wieder $(-P) + P = O$ gültig ist.

Kryptographische Anwendung Für die kryptographische Anwendung, insbesondere für Verfahren, die auf dem diskreten Algorithmus basieren, wird zuerst die Gruppenpotenz definiert zu $y = x^n = x \diamond x \diamond x \cdots x \diamond x$. Das diskrete Logarithmus Problem besteht nun bekanntermaßen darin die Zahl n zu finden, was bei einer Gruppenaddition per elliptischer Kurve ziemlich schwierig ist. Eines der bekanntesten Beispiele ist die Umsetzung des Diffie-Hellman Schlüsselaustausches (vgl. Abschnitt 3.2) auf elliptische Kurven [Ros99]. Aber auch Signatur- und Public-Key Verfahren können damit realisiert werden [**?**], [**?**].

4. Integrität und Authentizität

Um die Integrität einer Nachricht zu schützen wendet man typischerweise entweder Message Authentication Codes (MACs) oder aber digitale Signaturen an[43]. Erstere sind mit symmetrischen Verfahren verbunden, letztere im Sprachgebrauch fast immer auf asymmetrische Algorithmen bezogen[44]. Praktisch wird bei einer Integritätsprüfung (in den meisten Fällen) gleichzeitig die Authentizität der Nachricht mit Bezug auf den Absender verifiziert.

[43] Die Verwendung von Countern zum Schutz vor Replay-Atacken ist meist optional.
[44] Integrität und Authentizität sind nicht voneinander zu trennen [MvV92, 9.6.1].

4.1. MACs

Die gebräuchlichsten Verfahren der Bildung von MACs sind mittlerweile standardisiert, z. B. in [ISO02b]. Dabei handelt es sich um den CBC-MAC und den HMAC mit kryptographischen Hashfunktionen wie MD5, SHA-1 oder RIPEMD-160.

4.1.1. CBC-MAC

Beim CBC-MAC handelt es sich um ein schon relativ lange in Benutzung befindliches Verfahren zur Erzeugung von MACs [ISO99]. Dabei wird eine Blockchiffre im CBC-Modus genutzt, deren Zwischenergebnisse jedoch nicht weiterverwendet. Statt dessen wird nur das Resultat der letzten Verschlüsselung (immer mit dem Schlüssel K) als kryptographische Prüfsumme (MAC) über die Nachricht M interpretiert. Folgende Formel beschreibt jeden Einzelschritt:

$$c_i = E_K(c_{i-1} \oplus m_i) \text{ mit } c_{-1} = IV.$$

Abbildung 18 stellt das gesamte Verfahren anschaulich dar. Erwähnt werden soll noch, daß

- die Nachricht M bei Bedarf auf die Blockbreite des Algorithmus E zu padden ist
- und immer ein Initialisierungsvektor (IV) benötigt wird[45].

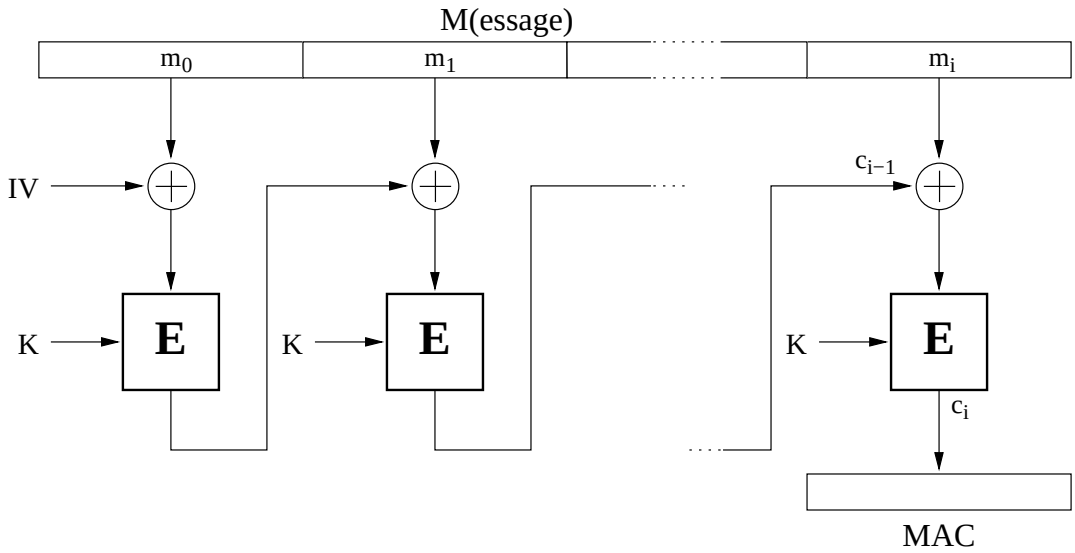
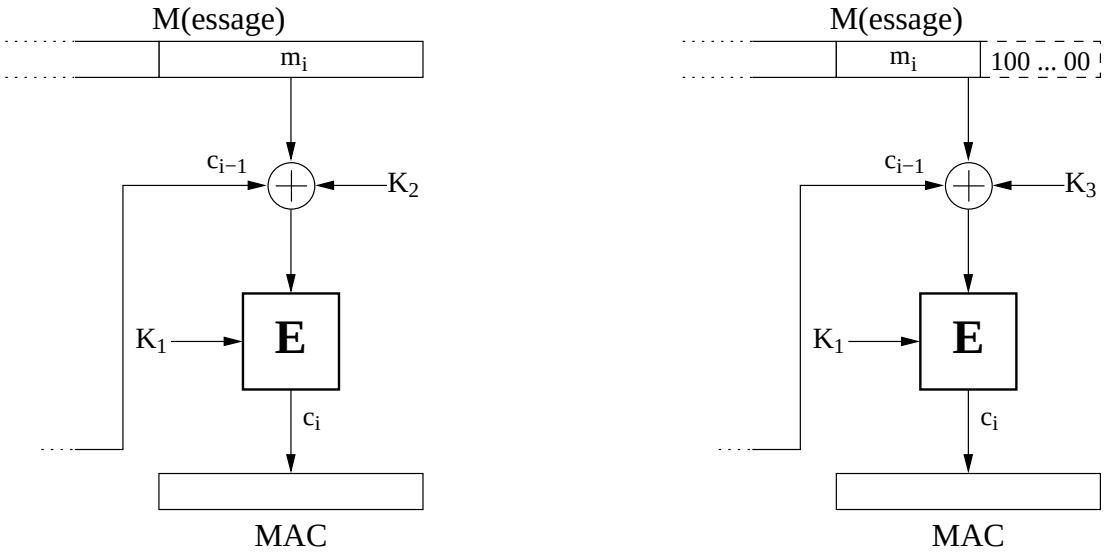


Abbildung 18: Bildung des CBC-MAC

[45] Der IV kann durchaus statisch sein, da ein Codebook-Angriff auf den ersten Block (wie bei der CBC-Verschlüsselung möglich) hier nicht anwendbar ist [MvV92, 9.5.1].

4.1.2. XCBC-MAC

Der XCBC-MAC (Extended MAC) geht im Original auf [BR03] zurück, hat seine Popularität aber wahrscheinlich IPSec zu verdanken [FH03]. Er behebt eine Schwachstelle des CBC-MAC, die nur bei Nachrichten variabler Länge in Erscheinung tritt [MvV92, Example 9.62]. Die Bildungsvorschrift entspricht weitestgehend dem CBC-MAC, unterscheidet sich jedoch wesentlich bei der Behandlung des letzten Blocks (siehe Abbildung 19).



(a) ohne Padding (b) mit Padding

Abbildung 19: Bildung des XCBC-MAC

Die Einzelschritte sind:

1. Aus dem Schlüssel K werden durch Verschlüsselung drei neue Schlüssel erzeugt.

$$K_1 = E_K(01_\mathrm{H} \ldots 01_\mathrm{H})$$
$$K_2 = E_K(02_\mathrm{H} \ldots 02_\mathrm{H})$$
$$K_3 = E_K(\underbrace{03_\mathrm{H} \ldots 03_\mathrm{H}}_{n/8})$$

2. Mit dem Schlüssel K_1 wird, abgesehen vom letzten Block, ein CBC-MAC nach Abschnitt 4.1.1 gebildet.

3. Die Verarbeitung des letzten Blocks wird nach Abbildung 19 vorgenommen, wobei das Vorgehen davon abhängt, ob die Länge der Nachricht M ein Vielfaches der Blockbreite n ist.

   a) Ist die Länge der Nachricht ein Vielfaches der Blockbreite n, dann wird im letzten Schritt

   $$\text{MAC} = E_{K_1}(m_i \oplus c_{i-1} \oplus K_2)$$

   gerechnet.

   b) Sollte jedoch Padding nötig ein, dann wird zuerst ein 1-Bit angehängt und bei Bedarf auf ein Vielfaches von n mit Nullen aufgefüllt. Die Berechnung des MAC geht nach Abbildung 19b vor sich:

   $$\text{MAC} = E_{K_1}(\underbrace{m_i \| (100\ldots00)}_{n} \oplus c_{i-1} \oplus K_3).$$

4.1.3. Hash-MAC (HMAC)

Der HMAC einer Message M wird mittels einer kryptographischen Hashfunktion[46] H berechnet [HK97, ISO02b, NIS07]. Dazu geht man in folgenden Schritten vor:

1. Der geheime Schlüssel K wird durch Anhängen von Null-Bytes auf die Blockbreite der Hashfunktion[47] gebracht (ANSI-Padding).
2. Der so verlängerte Schlüssel wird nun mit einer Wiederholung von 36_H bitweise Exklusiv-Oder (XOR) verknüpft.
3. An diesen ersten Block wird die (zu schützende) Message M angehängt und auf die Verkettung unsere Hashfunktion H angewendet. Als Resultat erhält man: $I(\text{nner}) = H[(K \| 00_H \ldots 00_H) \oplus (36_H \ldots 36_H) \| M]$.
4. Genauso wie in Punkt 1 und 2 wird nun der Schlüssel K nocheinmal verarbeitet, nur das die XOR-Operation mit $5C_H$ erfolgt.
5. An das Ergebnis wird $I(\text{nner})$ angehängt und darauf erneut die Hash-Funktion H angewendet. Als Ergebnis liegt schlußendlich der HMAC der Message M vor[48]: $O(\text{uter}) = H[(K \| 00_H \ldots 00_H) \oplus (5C_H \ldots 5C_H) \| I]$.

Abbildung 20 veranschaulicht das gesamte Verfahren.

[46] Die Hashfunktion bestimmt in wesentlichem Maße die Sicherheit des Algorithmus.

[47] Der Fall, daß die Länge des Schlüssels K die Blockbreite der Hashfunktion überschreitet kommt in der Praxis weniger häufig vor. Tritt er aber ein, so wird nach [HK97] in einem vorgelagerten Schritt $K' = H(K)$ gebildet und an Stelle von K immer mit K' gearbeitet.

[48] Die Bildung von $O(\text{uter})$ ist recht effizient, da nur zwei Blöcke zu verarbeiten sind.

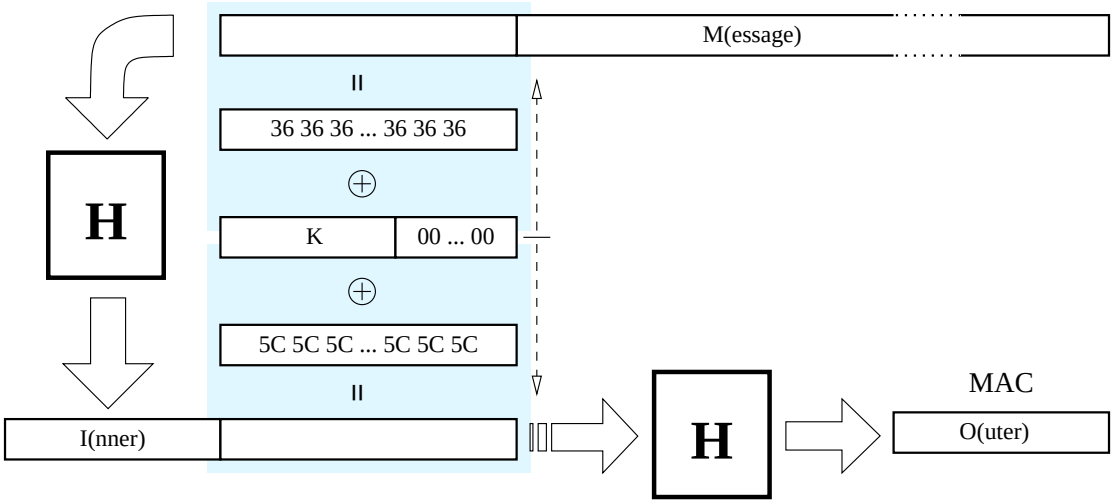


Abbildung 20: Bildung des HMAC

4.2. Signaturen

4.2.1. Einleitung

Elektronische Signaturen dienen der Authentifizierung – im Sinne dessen daß sie (vgl. Signaturgesetz):

- dem Inhaber eines Signaturschlüssels eindeutig zugeordnet sind bzw. ihn identifizieren;
- eine nachträgliche Änderung der Daten (Nachricht), mit denen sie verknüpft sind, sicher erkennbar machen.

Man unterscheidet Signaturen mit „Message Recovery" (siehe z. B. [ISO02a] oder [MvV92]) und Signaturen mit „Appendix". In der Praxis werden Signaturen mit Anhang nach [IEE00, PKC02, NIS09, ANS05] wohl am häufigsten verwendet. Ihre Bildungsvorschrift kann man wiefolgt skizzieren (siehe auch Abbildung 21):

1. H(ash): Berechne den Hash (bzw. Message Digest) der Nachricht M.
2. C(oding): Kodiere das Ergebnis nach einem dem Kryptoalgorithmus (siehe nächster Punkt) angepaßten Schema.
3. E(ncrypt): Verschlüssele den so strukturierten Hash mit dem privaten Teil K_{priv} des Signaturschlüssels.

Die Integrität der empfangenen (und möglicherweise veränderten) Nachricht M' kann im weiteren mit Hilfe der Signatur verifiziert werden. Dazu sind auf der Empfängerseite folgende Schritte nötig:

1. Berechne H(M') in gleicher Art und Weise wie beim Herausgeber.
2. Entschlüssele die Signatur mit Hilfe des öffentlichen Teils K_{pub} des Signaturschlüssels und dekodiere H(M).

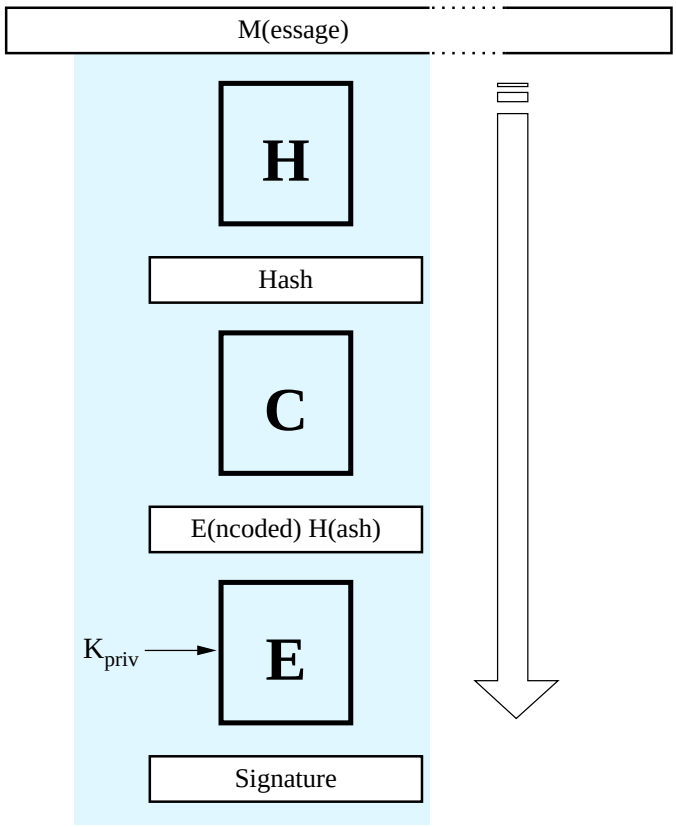


Abbildung 21: Signatur mit Anhang

3. Vergleiche H(M) mit H(M') um die Integrität $M' = M$ zu verifizieren.

4.2.2. RSA-Signaturen nach PKCS #1

Die RSA-Signatur über eine Nachricht m wird berechnet, indem der Hash-Wert von m mit dem privaten RSA-Schlüssel (vgl. Abschnitt 3.1) verschlüsselt wird. Auf der Empfangsseite muß der, mit dem öffentlichen Schlüssel entschlüsselte Hash, mit dem berechneten Wert für die Nachricht übereinstimmen[49].

RSA hat die für digitale Signaturen wichtige Eigenschaft, daß Ver- und Entschlüsselung vertauschbar sind.

$$p = \mathrm{RSA}_{(e,n)}\{\mathrm{RSA}_{(d,n)}[p]\} = \mathrm{RSA}_{(e,n)}\{\mathrm{RSA}_{(d,n)}[p]\} \tag{16}$$

[49] Die (etwas unübliche) Verwendung des privaten Schlüssels für die Verschlüsselung ist auf der Grundlage von Symmetrieformel 16 möglich.

PKCS [PKC02], [BR96]

4.2.3. Digital Signature Algorithm (DSA)

Der *Digital Signature Algorithm* (DSA) ist in [NIS94] standardisiert. wurde für ATM Security in [AF99, Annex 6.8.2] verwendet. Signiert wird der Hash-Wert[50] $h(m)$ der Message m.

Voraussetzung

1. Zuerst wählt man wieder eine große Primzahl p (das Modul) und ermittelt einen primen Teiler der EULER-Funktion $\phi(p) = (p-1)$, bezeichnet mit q.
2. Nun wird eine Zahl h (mit $1 < h < p-1$) gewählt und modulo-exponentiert zu

$$g = h^{(p-1)/q} \bmod p \tag{17}$$

3. Außerdem wird ein privater Schlüssel x mit $0 < x < q$ gewählt.
4. Der öffentliche Schlüssel y wird nach folgender Formel berechnet:

$$y = g^x \bmod p \tag{18}$$

5. Die Daten p, q (optional g, dann muß g nicht erneut berechnet werden) sowie der öffentliche Schlüssel y werden sicher an den/die Empfänger verteilt.

Signieren Zum Signieren wählt man eine beliebige Zahl k, mit $0 < k < q$, und berechnet damit zwei Zahlen r, s nach folgender Vorschrift:

$$r = g^k \bmod p \bmod q \tag{19}$$
$$s = k^{-1}[h(m) + xr] \bmod q \tag{20}$$

Zur Bestimmung des inversen k ist das Lösen der Kongruenz $k^{-1}k \equiv 1 \pmod q$ notwendig. Vorteilhaft können sowohl Gleichung 19 als auch k^{-1} im Voraus (off-line) berechnet werden[51].

Abschließend wird die Message m sowie (r, s) als Signatur zum Empfänger übertragen.

[50] Durch FIPS PUB wird hier i.A. die Verwendung des SHA-1 Algorithmus gefordert.
[51] In sehr seltenen Fällen kann sowohl r als auch s Null werden, dann sollte man eine andere Zahl k wählen.

Verifizieren Das Verifizieren einer empfangenen Signatur $(\hat{r}, \hat{s})$ erfolgt folgendermaßen:

1. Zuerst wird geprüft ob $0 < \hat{r} < q$ und $0 < \hat{s} < q$ gilt.
2. Dann berechnet der Empfänger aus dem empfangenen $\hat{s}$ das Modulo-Inverse $w = \hat{s}^{-1} \bmod q$
3. Damit werden zwei Exponenten berechnet
   a) $u_1 = [h(m)w] \bmod q$ und
   b) $u_2 = (\hat{r}w) \bmod q$
4. Zuletzt wird $v = [g^{u_1} \cdot y^{u_2}] \bmod p \bmod q$ berechnet, was genau $\hat{r}$ entsprechen muß.

Beweis Dazu für richtige Signatur, d.h. $r = \hat{r}$ und $s = \hat{s}$, ausgehend von Gleichung 18 und der Berechnungsformel für v, der Beweis.

$$\begin{aligned}
v &= (g^{u_1} \cdot y^{u_2}) \bmod p \pmod q \\
&= \left[g^{h(m)w - l_1 q} \cdot y^{rw - l_2 q}\right] \bmod p \pmod q \\
&= \left[g^{-(l_1+l_2)q} \cdot g^{h(m)w} \cdot (g^x \bmod p)^{rw}\right] \bmod p \pmod q \\
&= \left[g^{-(l_1+l_2)q} \cdot g^{h(m)w} \cdot g^{xrw}\right] \bmod p \pmod q \\
&= \left\{g^{-(l_1+l_2)q} \cdot g^{[h(m)+xr]w}\right\} \bmod p \pmod q \\
&= \left\{g^{-(l_1+l_2)q} \cdot g^{s^{-1}[h(m)+xr]}\right\} \bmod p \pmod q
\end{aligned}$$

Umformung von Voraussetzung in Formel 20 zu

$$\begin{aligned}
sk &= h(m) + xr \pmod q \\
k &= s^{-1}\,[h(m) + xr] \pmod q
\end{aligned}$$

und Einsetzen liefert für v unter Zuhilfenahme von Gleichung 17

$$\begin{aligned}
v &= \left[g^{-(l_1+l_2)q} \bmod p \cdot g^k \bmod p\right] \bmod p \pmod q \\
&= \left[(g^q)^{-(l_1+l_2)} \bmod p \cdot g^k \bmod p\right] \bmod p \pmod q \\
&= \left[(g^q \bmod p)^{-(l_1+l_2)} \bmod p \cdot g^k \bmod p\right] \bmod p \pmod q \\
&= \left[\left(h^{p-1} \bmod p\right)^{-(l_1+l_2)} \bmod p \cdot g^k \bmod p\right] \bmod p \pmod q
\end{aligned}$$

für jedes $l_v \in \mathbb{Z}$.

Nach FERMATS "kleinem" Satz ist aber $h^{p-1} \bmod p$ kongruent Eins und es ergibt sich für v endgültig

$$\begin{aligned} v &= g^k \bmod p \pmod q \\ v &= r \end{aligned}$$

4.2.4. Efficient Digital Signature Scheme (ESIGN)

ESIGN nutzt wieder das Faktorisierungsproblem großer Zahlen als Einweg-Funktion [Oka90]. Zum Signieren wird aus dem Hashwert der Message $h'(m)$ ein Signaturblock $h(m)$ nach [AF99, Annex 6.8.3.1] gebildet, der dann mit dem *ESIGN*-Verfahren bearbeitet wird..

Voraussetzung

1. Zuerst wählt man zwei große Primzahlen p und q ($p > q$ aber mit gleicher Bitlänge $L_p = L_q = L$), die den privaten Schlüssel (p,q) bilden.
2. Daraus wird das öffentliche Modul $n = p^2 q$ (mit Bitlänge $L_n = 3L$ und Nebenbedingung $h(m) < n$) berechnet, welches zusammen mit einer Zahl $k \geq 4$ den öffentlichen Schlüssel (n,k) bildet.

Signieren

1. Zum Signieren wird jetzt eine Zufallszahl x mit $0 \leq x < pq$ gewählt.
2. Es werden daraus die folgenden Größen berechnet[52]

$$w = \left\lceil \frac{h(m) - (x^k \bmod n)}{pq} \right\rceil = \frac{h(m) - (x^k \bmod n)}{pq} + \frac{\epsilon}{pq}, \quad \epsilon < pq \tag{21}$$

$$y = \frac{w}{k x^{k-1}} \bmod p = \frac{h(m) - (x^k \bmod n) + \epsilon}{pqkx^{k-1}} \bmod p \tag{22}$$

3. und letztlich die Signatur S zu

$$S = x + ypq \tag{23}$$

[52] $y = \lceil x \rceil$ wird als *ceil*-Funktion bezeichnet und definiert die kleinste Zahl x die größer oder gleich x ist.

Verifizieren Der Empfänger verifiziert die Signatur S durch Berechnung von $S^k \bmod n$. Wenn dieser Wert innerhalb der Schranken

$$h(m) \leq S^k \bmod n < h(m) + 2^{2L}$$

liegt, ist die Signatur gültig[53].

Beweis Zuerst setzt man Gleichung 23 in die Verifikationsformel ein und wendet den binomischen Lehrsatz[54] an

$$\begin{aligned} S^k \bmod n &= (x + ypq)^k \bmod n \\ &= \sum_{i=0}^{k} \binom{k}{i} x^i (ypq)^{k-i} \bmod n \\ &= x^k + kypqx^{k-1} \bmod n \end{aligned}$$

Offensichtlich scheint die Substitution des Terms ypq zu lohnen, wozu Gleichung 22 ohne das Modul p geschrieben wird.

$$\begin{aligned} y &= \frac{h(m) - (x^k \bmod n) + \epsilon}{pqkx^{k-1}} - \upsilon p \\ ypqkx^{k-1} &= h(m) - (x^k \bmod n) + \epsilon - \upsilon nkx^{k-1} \end{aligned}$$

Einsetzen ergibt

$$\begin{aligned} S^k &\equiv x^k + h(m) - (x^k \bmod n) + \epsilon - \upsilon nkx^{k-1} \pmod n && (24) \\ &= h(m) + \epsilon - \upsilon nkx^{k-1} \pmod n && (25) \\ S^k &\equiv h(m) + \epsilon \pmod n && (26) \end{aligned}$$

Man sieht sofort, daß der Ausdruck $S^k \pmod n$ in Formel 26 immer größer als $h(m)$ ist.

Wegen der beschränkten Bitlänge L von p, q gilt immer $pq < 2^{2L}$. Kombiniert mit der Bedingung $\epsilon < pq$ erhält man

$$\epsilon < 2^{2L}$$

[53] Diese Bedingung wird häufig auch geschrieben als $h(m) \leq S^k \bmod n < h(m) + 2^{\lceil 2/3 L_n \rceil}$

[54] Binomischer Satz: $(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i}$ mit $\binom{n}{i} = \frac{n!}{i!(n-i)!}$

Nun kann man wieder Zuhilfenahme von Gleichung 26 auch den rechten Teil der Verifikationsungleichung beweisen

$$\begin{aligned} \left[S^k - h(m)\right] \bmod n &< 2^{2L} \\ S^k \bmod n &< h(m) + 2^{2L} \end{aligned}$$

Implementierung Praktisch kann man den Algorithmus durch Offline-Berechnung der Ausdrücke

- $x^k \bmod n$
- $(1/kx^{k-1}) \bmod p$

noch beschleunigen [Oka90], [Sch96].

Anhänge

A. Algebraische Grundstrukturen

A.1. Gruppen

A.1.1. Allgemeine Definition

Eine Gruppe[55] $(G, \diamond)$ ist ein algebraisches System, daß aus einer nicht-leeren Menge von Elementen G besteht, für die eine Operation $\diamond$ mit folgenden Eigenschaften definiert ist [PW72, 2.1], [Kör88, 101 ff.]:

1. die Anwendung der Operation $\diamond$ auf zwei Elemente muß wieder zu einem Element in G führen (Abgeschlossenheit): $\diamond : G \times G \rightarrow G$;
2. für alle $a, b, c \in G$ muß gelten: $a \diamond (b \diamond c) = (a \diamond b) \diamond c$ (Assoziativität);
3. ein neutrales Element $e \in G$ muß existieren, so daß gilt: $x \diamond e = x$ mit $x \in G$ (Identität);
4. für jedes Element $a \in G$ muß ein inverses Element $b \in G$ existieren, so daß $a \diamond b = e$ gilt.

Die Menge G kann aus einer endlichen oder unendlichen Anzahl von Elementen a_i bestehen. Bei einer Aufzählung der Elemente werden diese in geschweifte Klammern eingeschlossen, z. B. so: $\{a_1, a_2, \dots\}$. Ist die Zusatzbedingung $a \diamond b = b \diamond a$ erfüllt, wird die Gruppe kommutativ bzw. Abel'sch genannt.

Die Ordnung (auch Mächtigkeit oder Kardinalität) einer Gruppe ist die Anzahl der Elemente in G, geschrieben als $\mathrm{ord}(G)$, $|G|$ oder auch $\#G$. Ist U eine Teilmenge von G, d. h. $U \subseteq G$, dann wird $(U, \diamond)$ als Untergruppe von $(G, \diamond)$ bezeichnet, wenn mit derselben Operation $\diamond$ auch U alle Eigenschaften einer Gruppe erfüllt [Bos96, 1.], [PW72, 2.4]. Der Zusammenhang zwischen der Ordnung von U und der von G wird durch den Satz von Lagrange beschrieben.

$$|G| = i|U| \tag{27}$$

Die Ordnung $|U|$ ist danach ein Teiler von $|G|$ und i der so genannte Index von G über U (bzw. Index von U in G), welcher auch als $i = |G : U|$ geschrieben wird.[56]

A.1.2. Additive Gruppen

Entsprechend der allgemeinen Definition wird eine Menge G additive Gruppe $(G, +)$ genannt, wenn für sie

1. eine assoziative Operation $a + (b + c) = (a + b) + c$ mit $a, b, c \in G$ definiert ist;

[55] Der Begriff wurde zuerst von Galois benutzt.

[56] Die Bezeichnungsweise des Index in der Form $|G : U|$ ist dabei durch den Quotienten $\frac{|G|}{|U|}$ motiviert. Interessant ist in diesem Zusammenhang auch noch, daß die Ordnung der Untergruppe U höchstens halb so groß sein kann, wie die der Gruppe G.

2. ein neutrales (Null-) Element $e_{(+)} = 0 \in G$ mit der Beziehung $a+0 = 0+a = a$ für alle $a \in G$ existiert;
3. und für sie außerdem ein inverses Element $-a \in G$ zu $a \in G$ mit der Relation $a+(-a) = 0$ definiert ist.

Das bekannteste Beispiel einer solchen Gruppe ist die der ganzen Zahlen $(\mathbb{Z}, +)$.

A.1.3. Multiplikative Gruppen

Für die multiplikative Gruppe $(G, \cdot)$ gilt äquivalent:

- eine assoziative Operation $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ mit $a, b, c \in G$;
- ein neutrales (Eins-) Element $e_{(\cdot)} = 1 \in G$ mit der Identität $a \cdot 1 = 1 \cdot a = a$ für $a \in G$;
- und ein inverses Element $a^{-1} \in G$ zu jedem $a \in G$ mit der Relation $a \cdot a^{-1} = 1$

sind definiert. Zur Multiplikation ist zu bemerken, daß nicht-negative Potenzen eines Elements induktiv definiert sind

$$a^0 = e_{(\cdot)} = 1, \qquad a^{n+1} = a \cdot a^n,$$

also durch n-malige Multiplikation.

$$a^n = \underbrace{a \cdot a \cdots a}_{n\text{-mal}}$$

Das Nullelement $e_{(+)}$ einer additiven Gruppe kann in einer multiplikativen Gruppe nicht enthalten sein, denn es ist grundsätzlich nicht invertierbar (siehe auch Abschnitt A.3 zu den Körpern).

A.2. Ringe

Eine Menge R nennt man einen Ring $(R, +, \cdot)$, wenn auf ihr sowohl Addition als auch Multiplikation mit $R \times R \to R$ erklärt sind [PW72, 2.2]. Speziell müssen folgende Axiome gelten:

1. Bezüglich der Addition bildet R eine Abel'sche Gruppe $(R, +)$.
2. $(R, \cdot)$ ist eine so genannte Halbgruppe,[57] d. h.
   - $(R, \cdot)$ ist abgeschlossen $(\cdot : R \times R \to R)$
   - und es gilt das Assoziativgesetz $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ für $a, b, c \in R$.

[57] Ein inverses Element wird dabei *nicht* gefordert, wenngleich einzelne Elemente von R ein Inverses besitzen können.

3. Außerdem muß für $a, b, c \in R$ das Distributivgesetz gelten $a \cdot (b + c) = ab + ac$.

Existiert zusätzlich noch die Identität $e_{(\cdot)} = 1 \in R$ mit $a \cdot 1 = 1 \cdot a = a$ für alle $a \in R$, dann wird $(R, +, \cdot)$ Monoid oder Ring mit *Eins* genannt. Die Menge der Elemente $r \in R$ in einem Monoid, welche mit $r \cdot r^{-1} = e_{(\cdot)}$ jeweils ein inverses Element besitzen, nennt man Einheitengruppe R^* des Ringes. Ein Ring wird außerdem als *kommutativ* bezeichnet, wenn auch für die Multiplikation das Kommutativgesetz $a \cdot b = b \cdot a$ gilt. Ein typisches Beispiel hierfür ist der Ring $(\mathbb{Z}, +, \cdot)$ der ganzen Zahlen, denn er erfüllt offensichtlich alle Axiome.

A.3. Körper

Ein Körper K wird durch ein System von Elementen (endlich oder unendlich) gebildet, die sich durch die definierten Operationen Addition, Subtraktion, Multiplikation und Division verknüpfen lassen[58] und für die das Distributivgesetz gilt [PW72, 2.3]. Präziser ausgedrückt:

1. $(K, +, \cdot)$ ist ein kommutativer Ring (zu den Eigenschaften vgl. Abschnitt A.2);
2. $(K \setminus \{0\}, \cdot)$ ist eine kommutative Gruppe, d. h. alle Elemente ausgenommen 0 (auch geschrieben als $K^* = K \setminus \{0\}$) müssen ein inverses Element besitzen.[59]

Die bekanntesten Körper sind die der

- reellen Zahlen $(\mathbb{R}, +, \cdot)$,
- komplexen Zahlen $(\mathbb{C}, +, \cdot)$,
- rationalen Zahlen $(\mathbb{Q}, +, \cdot)$
- sowie der endliche Körper $\mathbb{Z}_2 := \{0, 1\}$.

Eine Menge von Funktionen über einem Körper K kann selbst auch wieder ein Körper sein, z. B. die Menge der rationalen Funktionen $\mathbb{R}(x)$ über $\mathbb{R}$, geschrieben als $(\mathbb{R}(x), +, \cdot)$.

A.4. Vektorraum

Eine Menge von Vektoren $V := \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \dots\}$, jeder bestehend aus einem geordneten n-Tupel von Elementen a aus einem Körper K, wird Vektorraum über dem Körper K genannt (vgl. [PW72, 2.5]), wenn:

1. die additive Gruppe $(V, +)$ existiert und vom Abel'schen Typ ist $(+ : V \times V \to V)$;[60]

[58] Jede dieser Operationen muß wieder zu einem Element dieses Körpers führen (Abgeschlossenheit).

[59] Man sagt auch, daß K ein kommutativer Ring mit Eins sei, dessen Einheitengruppe aus allen Elementen außer der Null besteht.

[60] In Bezug auf den Begriff des Vektorraumes ist entscheidend (Abgrenzung zu einer Algebra, vgl. Abschnitt A.5), daß nur die Addition zwischen Vektoren sowie die Multiplikation mit einem Skalar aus K definiert ist.

2. für jeden Vektor $\mathbf{v} \in V$ die Multiplikation mit einem Körperelement (Skalar) a definiert ist und wieder zu einem Vektor $\mathbf{u} = a\mathbf{v} \in V$ führt (Abgeschlossenheit der Abbildung $\cdot : K \times V \to V$);[61]
3. bezüglich zweier Skalare a, b und der Vektoren $\mathbf{u}, \mathbf{v}$ die folgenden Distributivgesetze gelten:

$$a(\mathbf{u} + \mathbf{v}) = a\mathbf{u} + a\mathbf{v}$$
$$\mathbf{v}(a + b) = a\mathbf{v} + b\mathbf{v}$$

4. das Assoziativgesetz $(ab)\mathbf{v} = a(b\mathbf{v})$ gilt;
5. und es ein multiplikativ neutrales Element in Bezug auf die Vektoren in V gibt.

Wenn mit n die Anzahl der Tupel eines Vektors $\mathbf{v}$ bezeichnet wird, dann nennt man den zugehörigen Vektorraum (über dem Körper K) üblicherweise $V_n(K)$ oder kurz $V = K^n$, also z. B. $\mathbb{R}^3$ oder $\mathbb{Z}_2^n$. Entsprechend ist auch das neutrale Element, welches als $\mathbf{0} = (0, 0, \ldots, 0)$ geschrieben wird, ein Vektor mit n Elementen.

Kann man jeden Vektor von V als Linearkombination von unabhängigen Vektoren $\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_n} \in U \subseteq V$ darstellen, also als

$$\mathbf{v} = \sum_{i=1}^{n} \alpha_i \mathbf{u_i},$$

so nennt man $\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_n}$ die Basis, $\alpha_1, \alpha_2, \ldots, \alpha_n$ die Koeffizienten und $n = \dim(V) = |V|$ die Dimension von V.

A.5. Algebra (Verband)

Eine Algebra über einem Körper K ist eine Erweiterung des Begriffs Vektorraum in Bezug auf die Vektormultiplikation.[62] Speziell sind es folgende Axiome, die eine Algebra auszeichnen:

1. die Menge V ist ein Vektorraum über einem Körper K (vgl. Abschnitt A.4);
2. für zwei Vektoren $\mathbf{u}, \mathbf{v} \in V$ ist ein Produkt $\mathbf{uv}$ definiert und in Bezug auf V abgeschlossen $(\cdot : V \times V \to V)$;
3. das Assoziativgesetz für Vektoren $(\mathbf{uv})\mathbf{w} = \mathbf{u}(\mathbf{vw})$ ist erfüllt;
4. auch das Distributivgesetz läßt sich auf Vektoren anwenden: $(\mathbf{u} + \mathbf{v})\mathbf{w} = \mathbf{uw} + \mathbf{vw}$.

[61] Die mit $\cdot$ gekennzeichnete Operation nennt man auch die äußere,+ die innere Verknüpfung.
[62] Ohne für jeden Vektor ein multiplikativ inverses Element zu fordern. Man spricht deshalb auch manchmal von einem assoziativen Vektorring.

A.6. Zusammenfassung

Tabelle 1 gibt einen Überblick zu den verschiedenen algebraischen Grundstrukturen und ihren Existenzbedingungen.

Tabelle 1: Eigenschaften algebraischer Strukturen

	Addition				Multiplikation				
	assoziativ	kommutativ	neutral	invers	assoziativ	kommutativ	neutral	invers	distributiv
Additive Gruppe	×		×	×					
(Multiplikative) Halbgruppe					×				
Monoid					×		×		
Multiplikative Gruppe					×		×	×	
Multiplikative Abel'sche Gruppe					×	×	×	×	
Ring	×	×	×	×	×		×		×
Schiefkörper	×	×	×	×	×		×	×	×
Körper	×	×	×	×	×	×	×	×	×

B. Endliche Strukturen

B.1. Multiplikative Abel'sche Gruppen

Entsprechend Abschnitt A.1.2 handelt es sich bei endlichen kommutativ-multiplikativen Gruppen um algebraische Strukturen $G^* := (G, \cdot)$, welche bzgl. der Multiplikation

- assoziativ;
- mit einem neutralen Element $e_{(\cdot)} = 1$ ausgestattet;
- eindeutig invertierbar;
- und kommutativ

sind.[63]

[63] Mit diesen Eigenschaften können sie als Einheitengruppe eines kommutativen Ringes mit Eins (vgl. Abschnitt A.2) oder als multiplikative Gruppe eines Körpers (vgl. Abschnitt A.3) aufgefaßt werden.

B.1.1. Ordnung von Elementen

Endliche multiplikative Gruppen sind insbesondere wegen ihrer Eigenschaften beim Potenzieren von Gruppenelementen sehr interessant. Betrachten wir dazu die Folge der Potenzen $r^1, r^2, r^3, r^4, \ldots$ irgendeines Elements $r \in G^*$. Nach dem Prinzip der Abgeschlossenheit (Gruppenaxiom) wird auch jede Potenz von r wieder in G^* liegen. Wegen der endlichen Zahl $|G^*|$ von Elementen, muß sich ab irgendeiner Potenz $l \leq |G^*|$ die Folge wiederholen. Eine solche Wiederholung läßt den Ansatz $r^i = r^l$ zu. Multiplikation mit dem inversen Element von r^i ergibt $1 = r^{l-i}$, was wegen $l > i$ wiederum bedeutet, das es immer ein Element mit

$$r^k = e_{(\cdot)} = 1, \qquad r \in G^* \tag{28}$$

gibt. Die Folge $\{r, r^2, r^3, \ldots, r^{k-1}, r^k = 1 = r^0\}$ nennt man die vom Element r erzeugte zyklische Untergruppe und kennzeichnet sie mit

$$\langle r \rangle = \{ r^n \,\big|\, 1 \leq n \leq k \} \subseteq G^* \,. \tag{29}$$

Unter Zuhilfenahme von $r^k = r^0$ läßt sich die Menge der Elemente auch so definieren:

$$\langle r \rangle = \{ r^{n \bmod k} \,\big|\, n \in \mathbb{N} \} \,. \tag{30}$$

Abbildung 22 stellt die Periodizität der Folge am Beispiel $k = 12$ als Kreisteilung dar.

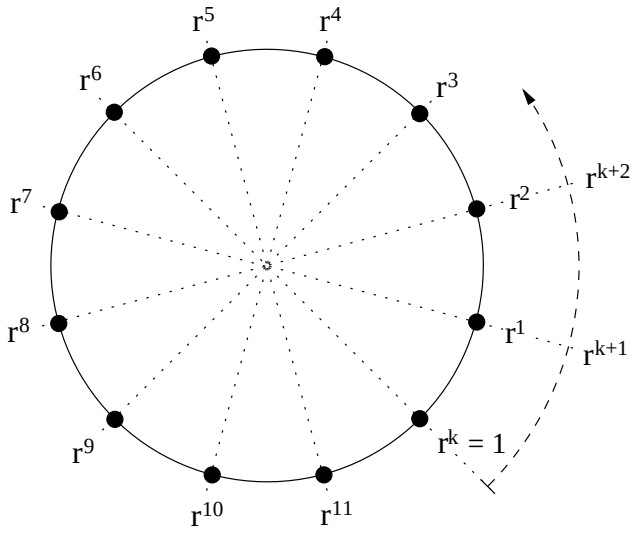


Abbildung 22: Zyklische Untergruppe $\langle r \rangle$

Die Ordnung (Anzahl der Elemente) der Untergruppe $\langle r\rangle$ ist $|\langle r\rangle| = k$. Sie ist gleichzeitig die kleinste Potenz k, die zu $r^k = 1$ führt. Man nennt sie auch Ordnung des Elements r und schreibt statt $\mathrm{ord}(r) = \#r = |\langle r\rangle|$ einfach nur $|r|$. Die Ordnung des neutralen Elements $e_{(\cdot)}$ ist 1, denn der kleinste Exponent k der zu $e^k_{(\cdot)} = e_{(\cdot)}$ führt, ist 1.[64]

Formel 28 gibt uns, wenn man sie mit r^{-1} multipliziert, eine Berechnungsvorschrift für das inverse Element an die Hand:

$$r^{-1} = r^{k-1}\,. \tag{31}$$

B.1.2. Potenzen eines Elements

Betrachten wir jetzt die n-te Potenz irgendeines Elements r und stellen die Frage nach der Ordnung des so erzeugten Elements $s = r^n$. Bezeichnet man dazu mit $k = |r|$ und $l = |s|$ die jeweilige Ordnung, so gilt entsprechend Beziehung 28:

$$r^k = r^{|r|} = 1 \qquad\qquad s^l = s^{|s|} = 1$$

und für weitere Potenzen von r^k:

$$r^{ik} = (r^k)^i = 1^i = \underbrace{1\cdot 1\cdot 1\cdots 1\cdot 1}_{i\text{-mal}} = 1\,. \tag{32}$$

Unter diesen Voraussetzungen kann man

$$s^l = (r^n)^l = r^{nl} = 1 = r^k = r^{ki}$$

formulieren und so $nl = ki$ schlußfolgern (Exponentenvergleich). Da sich unser Interesse auf den kleinsten Exponenten l beschränkt, haben wir es hierbei mit der Frage nach dem kleinsten gemeinsamen Vielfachen von n und k zu tun. Ein Beispiel für $k = 6$ und $n = 4$, also $\mathrm{lcm}(k,n) = 12 = 4\cdot 3 = 6\cdot 2$ zeigt Abbildung 23.

[64] Und dies ist das einzige Element mit der Eigenschaft $|r| = 1$.

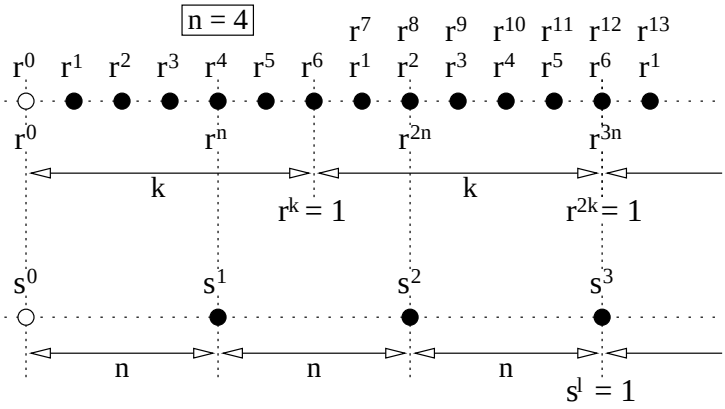


Abbildung 23: Potenzen eines Elements

Obwohl mit $nk = \operatorname{lcm}(n,k)\gcd(n,k)$ auch eine Berechnungsvorschrift zur Verfügung steht, wollen wir aus Verständnisgründen den ausführlichen Weg beschreiten. Dazu wird unter Zuhilfenahme der Abkürzung $d = \gcd(k,n)$ und mittels der Produktdarstellungen $k = \overline{k}d$ und $n = \overline{n}d$ zuerst der gemeinsame Teiler d eliminiert.

$$\overline{n}l = \overline{k}i \tag{33}$$

Wegen der Teilerfremdheit von $\overline{n}$ und $\overline{k}$ kann nur die Multiplikation mit der jeweils anderen Größe zum kleinsten gemeinsamen Vielfachen $\operatorname{lcm}(\overline{n},\overline{k}) = \overline{n}l = \overline{k}i$ führen.

$$i = \overline{n} = \frac{n}{\gcd(k,n)} \qquad l = \overline{k} = \frac{k}{\gcd(k,n)} \tag{34}$$

Die Konsequenzen aus dem Ergebnis

$$l = \left|r^n\right| = \frac{|r|}{\gcd(|r|,n)} \tag{35}$$

sind recht interessant:

1. Die Ordnung eines durch Potenzieren erzeugten Elements $s = r^n$ ist immer kleiner/gleich der Ordnung des Ausgangselements r. Für den Fall $\gcd(|r|,n) = 1$ ist sie maximal (siehe auch Punkt 4).
2. Als Bestätigung für den Satz von Lagrange (vgl. auch Abschnitt A.1.1) ist festzustellen, daß die Ordnung von $s = r^n$ genau die Ordnung von r teilt.[65] Im Sinne der Definition

[65] Das ist allerdings keine neue Erkenntnis, sondern eigentlich der Ausgangspunkt nach Formel 32.

des Index einer Gruppe über deren Untergruppe, hier von $\langle r\rangle$ über $\langle s\rangle$, gilt deshalb: $|\langle r\rangle : \langle s\rangle| = \gcd(|r|, n)$. Die Ordnung $|s|$ ist dabei (entsprechend der Bedeutung eines größten gemeinsamen Teilers) der bezüglich n teilerfremde Anteil in $|r|$.

3. Bei Kenntnis der Ordnung $k = |r|$ ist automatisch die Ordnung jedes Elements in der zyklischen Untergruppe $\langle s\rangle$ bekannt (vergleiche Formel 35 mit Mengendefinition 29).

4. Zwei Elemente $r \neq s$ mit derselben Ordnung sind nach Formel 35 dadurch gekennzeichnet, daß $\gcd(k, n) = 1$ gilt. Die Anzahl der zur Ordnung k teilerfremden Zahlen (die kleiner als k sind) entspricht damit der Anzahl von möglichen Potenzen n, für die $k = |r^n| = |s|$ wird. Aus diesem Grund wird in einer multiplikativen Gruppe die Anzahl der Elemente mit jeweils gleicher Ordnung k genau durch Euler's Totient-Funktion[66] $\phi(k)$ bestimmt.

5. Bei Kombination der Formeln 35 und 31 stellt man fest, daß sich die Ordnung eines Elements r beim Übergang zu dessen Inversen r^{-1} nicht ändert: $|r^{-1}| = |r^{k-1}| = k/\gcd(k, k-1) = k$.

B.1.3. Beziehung zur Gruppenordnung

Variante 1 Die Ordnung der von r erzeugten zyklischen Untergruppe $\langle r\rangle$ ist nach dem Satz von Lagrange (siehe Abschnitt A.1.1) ein Teiler der Gruppenordnung $|G^*|$. Man kann die Ordnung der multiplikativen Gruppe deshalb auch folgendermaßen ausdrücken [Bos96, 1.2, Satz 3]:

$$|G^*| = ik\,. \tag{36}$$

Daß die Ordnung von $\langle r\rangle$ ein Teiler von $|G^*|$ ist, führt in Verbindung mit Gleichung 28 zu:

$$r^{|G^*|} = r^{ik} = (r^k)^i = 1\,. \tag{37}$$

Anschaulich (siehe auch Abbildung 23) bedeutet dies, daß sich die Potenzen r^n nach k Elementen periodisch wiederholen .

$$\underbrace{\underbrace{r^1, r^2, r^3, \ldots, r^k}_{k\text{ Elemente}}, \underbrace{r^1, r^2, r^3, \ldots, r^k}_{k\text{ Elemente}}, \ldots, \underbrace{r^1, r^2, r^3, \ldots, r^k}_{k\text{ Elemente}}}_{|G^*|\text{ Elemente }(i\text{ Perioden})}$$

[66] Euler's Totient-Funktion $\phi(k)$ liefert eine Aussage über die Anzahl der zu k teilerfremden Zahlen, welche kleiner als k sind (siehe auch Abschnitt C.3.2).

Variante 2 Möchte man einen Rückgriff auf den Satz von LAGRANGE vermeiden, so kann für Beziehung 36 auch der folgende Beweis angeführt werden. Multipliziere jedes der $|G^*|$ Elemente aus $G^* = \{r_1, r_2, r_3, \ldots, r_{|G^*|}\}$ mit dem Element r, welches ebenfalls G^* entstammt (r ist eines der r_n, mit $n = 1, 2, \ldots, |G^*|$). Nach dem Prinzip der Abgeschlossenheit muß auch jedes Produkt rr_n wieder in G^* liegen. Außerdem müssen die Produkte paarweise verschieden sein, sonst würde die Multiplikation mit dem inversen Element r^{-1} zu zwei gleichen Elementen führen (Bed. $rr_\nu \neq rr_\mu$). Abgesehen von der Reihenfolge kann deshalb folgende eindeutige Mengenabbildung (Bijektion) angegeben werden: $\{rr_1, rr_2, rr_3, \ldots, rr_{|G^*|}\} \mapsto \{r_1, r_2, r_3, \ldots, r_{|G^*|}\}$. Bildet man jetzt das Produkt aller so erzeugten Elemente

$$rr_1 \cdot rr_2 \cdot rr_3 \cdots rr_{|G^*|} = r_1 r_2 r_3 \cdots r_{|G^*|}$$
$$r^{|G^*|} r_1 r_2 r_3 \cdots r_{|G^*|} = r_1 r_2 r_3 \cdots r_{|G^*|}$$

und multipliziert noch mit den Inversen r_n^{-1}, dann bestätigt sich $r^{|G^*|} = 1$. Einzig mögliche Schlußfolgerung aus $r^{|G^*|} = 1$ und $r^k = 1$ kann aber (konform zu Beziehung 36) nur die sein, daß die Elementeordnung k genau die Gruppenordnung $|G^*|$ teilt.

B.1.4. Generatorelemente

Allgemein nennt man jede, von mindestens einem Element g durch Potenzierung erzeugte multiplikative Gruppe, eine zyklische Gruppe (siehe Abschnitt B.1.1). Umfaßt die von g erzeugte Untergruppe $\langle g \rangle$ alle Elemente der multiplikativen Gruppe G^* (in irgendeiner Abfolge), dann bezeichnet man g als Generatorelement oder primitives Element.

$$\langle g \rangle := \left\{ g^1, g^2, g^3, \ldots, g^{|G^*|-1}, g^{|G^*|} = g^0 = 1 \right\} = G^* \tag{38}$$

Die Ordnung eines solchen Elements muß in diesem Sinne der Gruppenordnung entsprechen.[67]

$$|g| = |G^*|$$

Wie alle anderen Elemente muß auch ein Generatorelement Gleichung 28 erfüllen - aber eben nur für Generatorelemente ist $|G^*|$ der kleinste Exponent, welcher zu $g^{|G^*|} = 1$ führt. Bei Kenntnis eines Generatorelements aus G^* sind so nicht nur alle Elemente der multiplikativen Gruppe bekannt, sondern nach Formel 35 auch deren Ordnung.

Um die Frage zu beantworten, ob es immer mindestens ein Generatorelement gibt, rekapitulieren wir folgende Fakten:

[67] Für den Spezialfall, daß es sich bei der Gruppenordnung $|G^*|$ um eine Primzahl handelt, sind alle Elemente primitiv.

1. Die Ordnung $k = |r|$ eines jeden Elements teilt die Gruppenordnung (Formel 36): $|G^*| = ik$.
2. Jedes Element r kann durch Potenzieren aus g erzeugt werden: $r = g^n$.
3. Die Ordnung $|g|$ eines Generatorelements erfüllt (wie die eines jeden anderen Elements) Formel 35: $|g| = |g^n| \gcd(|g|, n)$.
4. Die Ordnung eines Generatorelements muß der Gruppenordnung entsprechen: $|g| = |G^*|$.

Kombination der formelmäßigen Voraussetzungen ergibt:

$$\begin{aligned} |G^*| &= k \gcd(|G^*|, n) \\ i &= \gcd(ik, n) \end{aligned} \tag{39}$$

und diese Bedingung muß für irgendeine Potenz $n = 1, 2, \ldots, |G^*|$ zu gewährleisten sein (und zwar eindeutig für jedes Element r). Äquivalenz 39 kann aber nur erfüllt werden, wenn man $n = i\overline{n}$, mit $\gcd(k, \overline{n}) = 1$ annimmt. Die Anzahl der teilerfremden Zahlen $\overline{n}$ kleiner als k liefert mit $\phi(k)$ Euler's Totient-Funktion (vgl. Abschnitt C.3.2). Sie ist immer größer als 0, weshalb stets *ein* primitives Element existiert. Aus diesem Grund ist jede multiplikative Abel'sche Gruppe zyklisch, mit $\phi(|G^*|)$ Generatorelementen.[68]

B.1.5. Elemente als Nullstellen

Wenden wir uns jetzt einer etwas anderen Sichtweise auf die Elemente der multiplikativen Gruppe G^* zu, nämlich der Betrachtung über Nullstellen. Dazu gehen wir von dem Polynom $\varphi(x) = x^{|G^*|} - 1$ mit $x, \varphi(x) \in G^*$ aus, welches die Nullstellen bzw. Einheitswurzeln α haben soll.

$$\alpha^{|G^*|} - 1 = 0 \qquad\qquad \alpha^{|G^*|} = 1 \qquad (\alpha \in G^*) \tag{40}$$

Wir stellen nun fest, daß entsprechend des Fundamentalsatzes der Algebra $\varphi(x)$ genau $|G^*|$ Nullstellen haben muß. Nach Formel 36 wird diese Bedingung aber auch durch jedes Element aus G^* erfüllt. Da deren Anzahl genau mit der Anzahl der Nullstellen übereinstimmt, kann es sich bei den Elementen $r \in G^*$ nur um die $|G^*|$ Nullstellen von $\varphi(x)$ handeln [PW72, Satz 6.18]. Das mehrfache Nullstellen nicht vorhanden sind, kann man (in Verbindung mit Gleichung 40) durch Ableitung von $\varphi(x)$ an den Stellen α nachprüfen.

$$\left.\frac{\mathrm{d}\varphi(x)}{\mathrm{d}x}\right|_{x=\alpha} = |G^*| \, x^{|G^*|-1}\Big|_{x=\alpha} = |G^*| \, \alpha^{-1} \alpha^{|G^*|} = |G^*| \, \alpha^{-1} \neq 0$$

Mit diesen Erkenntnissen läßt sich für $\varphi(x)$ eine Linearfaktordarstellung auf Basis der Elemente r angeben.

[68] Zur Anzahl von Elementen mit derselben Ordnung siehe auch Bemerkung 4 auf Seite 55.

$$\varphi(x) = x^{|G^*|} - 1 = \prod_{r \in G^*} (x - r) \tag{41}$$

Ausmultiplizieren der rechten Seite führt mit $\varphi(0) = -1$ noch zu der interessanten Äquivalenz:

$$1 + \prod_{r \in G^*} r = 0\,. \tag{42}$$

Da alle Elemente r der multiplikativen Gruppe G^* als Potenzen eines Generatorelements g darstellbar sind, kann man die Linearfaktordarstellung 41 auch folgendermaßen schreiben:

$$\varphi(x) = \prod_{n=1}^{|G^*|} (x - g^n) = (x-g)(x-g^2)(x-g^3)\cdots(x-g^{|G^*|-1})(x-1)\,.$$

B.2. Endliche Körper

B.2.1. Definition

Jeder endliche Körper ist durch eine beschränkte Anzahl von Elementen gekennzeichnet, auf welche die Körperaxiome von Abschnitt A.3 zutreffen. Man nennt solche algebraischen Strukturen auch Galois-Körper und bezeichnet sie mit $\mathbb{F}_q$ oder GF(q), wobei q die Ordnung (Anzahl der Elemente) des Körpers angibt.[69] Bei der Konstruktion eines endlichen Körpers ist von allergrößter Bedeutung, daß zu den Eigenschaften eines Ringes noch die der multiplikativen Gruppe kommen. Zusätzliches Kriterium ist danach die Existenz des multiplikativ inversen Elements r^{-1} zu jedem $r \in \mathbb{F}_q^*$.

B.2.2. Ordnung

Definiert q die Anzahl der Elemente im Körper, d. h. inklusive Nullelement $e_{(+)}$, dann muß für die Ordnung der multiplikativen Gruppe $\mathbb{F}_q^* := \mathbb{F}_q \setminus \{0\}$ gelten:

$$|\mathbb{F}_q^*| = q - 1\,. \tag{43}$$

[69] Unerheblich ist dabei, ob die Menge der Körperelemente r durch eine Restklassenoperation oder irgendeine andere Methode definiert wird. Bei gleicher Anzahl von Elementen kann man solche Mengen nämlich immer (strukturerhaltend, eineindeutig) aufeinander abbilden [PW72, 6.5].

Entsprechend Abschnitt B.1.3 muß die Ordnung der multiplikativen Gruppe ein Vielfaches der Elementeordnung $k = |r|$ sein: $|\mathbb{F}_q^*| = ik$ (Satz von LAGRANGE). Ein GALOIS-Körper kann deshalb nicht jede beliebige Ordnung annehmen – wegen vorgenannter Bedingung müssen $q = |\mathbb{F}_q^*| + 1$ und die Ordnung k eines jeden Elements $r \in \mathbb{F}_q^*$ teilerfremd sein.[70]

$$\gcd(q,k) = 1 \tag{44}$$

Diese Erkenntnis läßt sich (einerseits logisch, aber auch rein analytisch) aus

$$1 = q - ik\,. \tag{45}$$

mit Hilfe des Satzes von BÉZOUT ableiten. Dazu vergleicht man Beziehung 59 mit Formel 60 aus Abschnitt D.1.1 und stellt fest:

$$\gcd(q,k) = 1, \quad \text{mit} \quad \alpha = 1, \beta = -i\,.$$

Die einfachste Möglichkeit Teilerfremdheit zu gewährleisten, ist die Wahl von q als Primzahl oder als Potenz einer Primzahl. Im ersten Fall nennt man $\mathbb{F}_p$ einen Primzahlenkörper, für $q = p^n$ einen Erweiterungs- oder Binärkörper (weitere Ausführungen zu $\mathbb{F}_{p^n}$ in Abschnitt C.4).

B.2.3. Elemente als Nullstellen

Aus Abschnitt B.1.5 (Gleichung 41) ist bekannt, daß man die Elemente der multiplikativen Gruppe $\mathbb{F}_q^*$ als Nullstellen des Polynoms $\varphi(x) = x^{q-1} - 1$ auffassen kann. Nimmt man jetzt noch das Nullelement $e_{(+)} = 0$ hinzu, dehnt also die Betrachtung von der multiplikativen Gruppe auf alle Elemente des Körpers aus, dann kann man als zusätzlichen Linearfaktor $x - 0$ einbeziehen:

$$\psi(x) = x\varphi(x) = x^q - x = \prod_{r \in \mathbb{F}_q} (x - r)\,. \tag{46}$$

Deshalb bilden die Nullstellen von $x^q - x = x(x^{q-1} - 1)$ einen endlichen Körper $\mathbb{F}_q$.

[70] Im Umkehrschluß können nur solche Elemente r der multiplikativen Gruppe $\mathbb{F}_q^*$ angehören, deren Ordnung k keinen Teiler mit $|\mathbb{F}_q|$ hat.

C. Restklassen

C.1. Definition

Restklassen sind Kongruenzen von Elementen (einer algebraischen Struktur) modulo eines fixen Elements m. Das Rechnen mit solchen Elementen wird als modulare oder Modulo-Arithmetik und m als das Modul bezeichnet. In diesem Sinne definiert man

$$r \equiv a \bmod m \tag{47}$$

als den Rest r, der bei der „Division“ a/m entsteht und sagt: r ist kongruent a modulo m. Im Umkehrschluß sind die Restklassenelemente durch die Relation

$$a = qm + r, \qquad 0 \leq r < m \tag{48}$$

bestimmt. Die Äquivalenzrelation $r \equiv a \bmod m$ kann man z. B. im Ring der ganzen Zahlen $\mathbb{Z}$ definieren, ist aber nicht auf diesen beschränkt. Im allgemeinen Fall $a \in R$ schreibt man für die Restklasse

$$[r]_m = \{a | a \in R, r \equiv a \bmod m\},$$

d. h. die Restklasse $[r]_m$ ist die Menge aller Elemente $a \in R$, die bei der Division modulo m genau den Rest r ergeben. Ein Beispiel für $a \in \mathbb{Z}$ mit einem Modul von $m = 4$ soll das veranschaulichen.

$$[0]_4 = \{\ldots, -12, -8, -4, 0, 4, 8, \ldots\} \qquad [2]_4 = \{\ldots, -10, -6, -2, 2, 6, 10, \ldots\}$$
$$[1]_4 = \{\ldots, -11, -7, -3, 1, 5, 9, \ldots\} \qquad [3]_4 = \{\ldots, -9, -5, -1, 3, 7, 11, \ldots\}$$

Üblicherweise bezeichnet man die Menge aller Restklassen mit

$$R_m = \{[r_0]_m, [r_1]_m, \ldots, [r_{n-1}]_m\},$$

also z. B. für die ganzen Zahlen $a \in \mathbb{Z}$ modulo m:

$$\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z} = \{[0]_m, [1]_m, \ldots, [m-1]_m\}\ .$$

Im Allgemeinen werden Verknüpfungsoperationen (Addition, Multiplikation usw.) zwischen zwei Restklassen dadurch definiert, daß man jeweils einen Vertreter aus den Restklassen auswählt und dann die Operation mit diesem Repräsentanten durchführt.

C.2. Restklassenringe

Um von einem Restklassenring $(R_m, +, \cdot)$ sprechen zu können ist der Nachweis aller Ringaxiome von Abschnitt A.2 in Bezug auf die Menge der Restklassen R_m notwendig [Wel01, 5.]. Dazu geht man von der Modulo-Arithmetik entsprechend Gleichung 47 und 48 aus und prüft jeden Punkt durch Einzelbetrachtung:

1. Bezüglich der Addition muß $(R_m, +, \cdot)$ eine Abel'sche Gruppe bilden, d. h.

   a) die Addition existiert und ist abgeschlossen;[71]

   $$\begin{aligned}[r]_m &= [r_1]_m + [r_2]_m = (a_1 - mq_1 + a_2 - mq_2) \bmod m \\ &= (a_1 + a_2) \bmod m - [m(q_1 + q_2)] \bmod m = (r_1 + r_2) \bmod m \\ &= [r_1 + r_2]_m\end{aligned}$$

   b) sie ist außerdem sowohl assoziativ als auch kommutativ (Abel'sch);

   $$[r_1]_m + ([r_2]_m + [r_3]_m) = ([r_1]_m + [r_2]_m) + [r_3]_m = [r_3]_m + [r_2]_m + [r_1]_m$$

   c) das neutrale Element $e_{(+)} = [0]_m = mq_0$ existiert;

   $$[r]_m + [0]_m = (a - mq + mq_0) \bmod m = [a - m(q - q_0)] \bmod m = a \bmod m = [r]_m$$

   d) jedes Element besitzt ein inverses Element $-[r]_m = (m - a) \bmod m$ mit

   $$[r]_m + (-[r]_m) = a \bmod m + (m - a) \bmod m = m \bmod m = [0]_m.$$

2. Für die Multiplikation muß $(R_m, +, \cdot)$ eine Halbgruppe darstellen, also

   a) ebenfalls abgeschlossen sein;

   $$\begin{aligned}[r]_m &= [r_1]_m \cdot [r_2]_m = (r_1 - mq_1) \cdot (r_2 - mq_2) \bmod m \\ &= [r_1 r_2 + m(mq_1 q_2 - r_1 q_2 - r_2 q_1)] \bmod m = (r_1 \cdot r_2) \bmod m \\ &= [r_1 \cdot r_2]_m\end{aligned}$$

   b) und das Assoziativgesetz für $[r_i]_m \in R_m$ erfüllen;

   $$[r_1]_m \cdot ([r_2]_m \cdot [r_3]_m) = ([r_1]_m \cdot [r_2]_m) \cdot [r_3]_m$$

[71] Die Operation führt immer wieder auf ein Element in R_m.

3. Außerdem muß für alle $[r]_m \in R_m$ das Distributivgesetz erfüllt sein.

$$\begin{aligned}[r_1]_m \cdot ([r_2]_m + [r_3]_m) &= (r_1 - mq_1) \cdot [r_2 + r_3 - m(q_2 + q_3)] \bmod m \\ &= (r_1 r_2 + r_1 r_3) \bmod m \\ &= [r_1]_m \cdot [r_2]_m + [r_1]_m \cdot [r_3]_m\end{aligned}$$

Da $(R_m, \cdot)$ ein neutrales Element $e_{(\cdot)}$ mit $[r]_m \cdot e_{(\cdot)} = [r]_m$ besitzt, handelt es sich sogar um einen *Ring mit Eins*.

$$[r]_m \cdot [1]_m = (r \cdot 1) \bmod m = r \bmod m = [r]_m$$

Aus Anwendungssicht ist sofort zu erkennen, daß insbesondere die Restklassen $R_m = \mathbb{Z}_m$ alle diese Bedingungen für $m \geq 2$ erfüllen und somit einen kommutativen Ring mit Einselement bilden.

C.3. Restklassenkörper

C.3.1. Existenz

Für den Übergang von einem Restklassenring $(R_m, +, \cdot)$ zu einem Restklassenkörper muß man zu jedem $r \in R_m \setminus \{0\}$ das Vorhandensein eines multiplikativ inversen Elements ${[r]_m}^{-1}$, mit $[r]_m \cdot {[r]_m}^{-1} = [1]_m$, fordern. Wir nehmen die Antwort vorweg und proklamieren, daß es ein solches Element in $(R_m, +, \cdot)$ nur dann gegeben kann, wenn m und r teilerfremd sind [PW72, Satz 6.4]. Bezieht man diese Aussage z. B. auf $\mathbb{Z}_m^* = \mathbb{Z}_m \setminus \{0\} = \{1, 2, 3, \ldots, m-1\}$, dann müssen (wenn keine weiteren Forderungen an m gestellt werden) alle Elemente r, für die $\gcd(r, m) = 1$ nicht erfüllt werden kann, ausgeschlossen werden. Sowohl im allgemeinen als auch speziellen Fall von $\mathbb{Z}_m^*$ ist diese Einschränkung grundsätzlich hinfällig, wenn es sich bei m um ein Primelement handelt (ein vollständiges Restklassensystem) – in Bezug auf $\mathbb{Z}_m^*$ also um eine Primzahl $p \in \mathbb{P}$, weshalb $\mathbb{F}_q := \mathbb{Z}_p$ (mit $q = p$). Im Fall des Auschlusses von Elementen (ein reduziertes Restklassensystem) ist die Anzahl der teilerfremden Zahlen durch Euler's Totient-Funktion $\phi(m)$ bestimmt und so die Ordnung der multiplikativen Gruppe $|\mathbb{Z}_m^*| = \phi(m)$, d. h. $\mathbb{F}_q \subseteq \mathbb{Z}_m$ (mit $q = \phi(m) + 1$).

Beweis Ist m kein primes Element, dann läßt es sich mindestens in zwei Faktoren $r, s \in R_m$ zerlegen (die nicht Vielfache von m sind, also $r, s \bmod m \neq 0$). Da nun $m \bmod m = 0$ ist, gilt für die Restklassenmultiplikation $[r]_m \cdot [s]_m = [m]_m = 0$. Soll aber $[r]_m$ eine inverse Restklasse $[r]_m^{-1}$ besitzen, dann kann man beide Seiten des Produktes mit ${[r]_m}^{-1}$ multiplizieren.

$$\underbrace{{[r]_m}^{-1}[r]_m}_{e_{(\cdot)}=1} \cdot [s]_m = [s]_m = {[r]_m}^{-1}[m]_m = 0$$

$[s]_m$ ist aber nach Voraussetzung nicht 0, demzufolge kann ein Inverses zu $[r]_m$ für den Fall dieser Zerlegung nicht existieren.

Im Gegenzug bleibt noch nachzuweisen, daß, wenn m ein Primelement ist, für jede Restklasse $[r]_m$ aus R_m ein inverses Element $[r]_m{}^{-1}$ auch wirklich existiert. Zu diesem Zweck betrachten wir alle $[r]_m \in R_m$, ausgenommen die Restklasse $[1]_m$, welche bei der Inversion auf sich selbst abgebildet wird. Da wegen der Modulo-Reduktion (wir verwenden jetzt wieder den Vertreter der Restklasse) immer $m > r$ gilt, kann nur r ein Teiler von m sein und nicht umgekehrt. Aber auch dies ist nicht möglich, wenn nach Voraussetzung m relativ prim zu r ist. Deshalb kann der größte gemeinsame Teiler von r und m nur das Einselement sein. Berücksichtigt man jetzt noch die aus dem euklidischen Algorithmus stammende Erkenntnis (Satz von Bézout, vgl. Abschnitt D.1), daß der größte gemeinsame Teiler $d = \gcd(r, s)$ zweier Elemente r, s in der Form $d = \alpha r + \beta s$ mit $\alpha, \beta \in \mathbb{Z}$ darstellbar ist, dann gilt:

$$\begin{aligned} \gcd(r, m) = 1 = \alpha r + \beta m \equiv \alpha r \bmod m \\ [1]_m = [\alpha]_m \cdot [r]_m \,. \end{aligned} \tag{49}$$

Das inverse Element von $[r]_m$ ist demzufolge

$$[r]_m{}^{-1} = [\alpha]_m,$$

wobei dessen Berechnung z. B. mit Hilfe des erweiterten euklidischen Algorithmus (siehe Abschnitt D.1.3) möglich ist.[72]

C.3.2. Multiplikative Gruppe

Da es sich bei den Restklassenkörpern um spezifische Galois-Körper handelt, kann man einige Formeln von Abschnitt B.2 konkretisieren. Im folgenden sollen deshalb die Körperelemente und deren Ordnung im Zusammenhang mit der multiplikativen Gruppe $\mathbb{F}_q^*$ betrachtet werden.

Ordnung von Elementen Abgesehen von den allgemein geltenden Ordnungsrelationen (siehe Abschnitt B.1) gibt es speziell für den Restklassenkörper $\mathbb{Z}_p$ noch eine erwähnenswerte Ausdrucksmöglichkeit für die von einem Körperelement generierte zyklische Untergruppe:

$$\langle r \rangle = \{ r^n \bmod (r^k - 1) \,\big|\, n \in \mathbb{N}, r \in \mathbb{Z} \} \,. \tag{50}$$

Einsichtig wird die Schreibweise sofort, wenn man sie für jeden Exponent n expandiert.[73]

[72] Diese Rechnung kann man auch in einem Ring ausführen, dann muß das Ergebnis jedoch nicht eindeutig sein (vgl. auch Bemerkungen auf Seite 82).

[73] Letztlich eine Spezialisierung der Mengendarstellung 30 von Abschnitt B.1.1 für den Fall $\mathbb{F}_q := \mathbb{Z}_p$.

$$
\begin{aligned}
r^0 \bmod (r^k - 1) &= r^0 \\
r^1 \bmod (r^k - 1) &= r^1 \\
r^2 \bmod (r^k - 1) &= r^2 \\
&\vdots \\
r^{k-1} \bmod (r^k - 1) &= r^{k-1} \\
r^k \bmod (r^k - 1) &= 1 = r^0 \\
&\vdots
\end{aligned}
$$

Kleiner Satz von Fermat Eine für die praktische Anwendung von Restklassenkörpern sehr wichtige Folgerung aus Gleichung 37 ist der (für den Restklassenkörper $\mathbb{Z}_p$ geltende) kleine Satz von Fermat [PD75, II]. Er resultiert sofort aus $r^{q-1} \equiv 1 \pmod{m}$, wenn man berücksichtigt, daß die Ordnung der multiplikativen Gruppe $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\} = \{1, 2, 3, \ldots, p-1\}$ genau $q - 1 = p - 1$ ist.

$$r^{p-1} \equiv 1 \pmod{p} \tag{51}$$

Aus dieser Kongruenz kann man wegen $0 \equiv p \pmod{p}$ außerdem ableiten, daß p stets ein Teiler von $r^{p-1} - 1$ ist.[74]

$$r^{p-1} - 1 = np \equiv 0 \pmod{p} \tag{52}$$

Aus Kongruenz 51 kann man außerdem eine Möglichkeit gewinnen, um r in $\mathbb{Z}_p^*$ zu invertieren.

$$r^{-1} \equiv r^{p-2} \pmod{p}$$

Satz von Wilson Betrachtet man die multiplikative Gruppe eines Restklassenkörpers $\mathbb{Z}_{p>2}$, so handelt es sich bei den Zahlen $r = 1, 2, \ldots, p-1$ um die $p - 1$ Nullstellen α des Polynoms $\varphi(x) = x^{p-1} - 1 \pmod{p}$. Anwendung von Gleichung 42 auf $\mathbb{Z}_p^*$ führt folgerichtig zum Satz von Wilson [Sho05, 2.8.1], [PD75, II], [Bun08, 2].[75]

$$(p-1)! \equiv -1 \equiv p - 1 \pmod{p} \tag{53}$$

[74] Oft auch in den Varianten $r^p \equiv r \pmod{p}$ oder $r^p - r \equiv 0 \pmod{p}$ verwendet.

[75] Er ist sowohl notwendige als auch hinreichende Bedingung dafür, daß es sich bei p wirklich um eine Primzahl handelt.

Satz von Euler L. Euler hat für natürliche Zahlen eine sogenannte Totient-Funktion $\phi(m)$ definiert, welche die Anzahl der positiven Zahlen (größer als 0 und kleiner als m) teilerfremd zu m ausdrückt.[76] Mit Hilfe dieser Funktion hat er Fermat's kleinen Satz folgendermaßen verallgemeinert [Sho05, 2.6], [Bun08, 2], [PD75, II]:

> Sind zwei Zahlen $r, m \in \mathbb{N}$ relativ prim zueinander, d. h. sie haben keinen gemeinsamen Teiler (und so ist $\gcd(r,m) = 1$), dann gilt:

$$r^{\phi(m)} \equiv 1 \pmod{m} . \tag{54}$$

Der Beweis ist mit den Betrachtungen von Abschnitt B.1 zur Ordnung der multiplikativen Gruppe in $\mathbb{Z}_m$ zu erbringen. Danach entspricht die Gruppenordnung $|\mathbb{Z}_m^*|$ der Anzahl invertierbarer Elemente (solche mit $\gcd(r,m) = 1$), d. h. mit Euler's Totient-Funktion genau $|\mathbb{Z}_m^*| = \phi(m)$.[77] Berücksichtigt man jetzt noch die Gruppenordnung nach Formel 37, dann bestätigt sich Euler's Satz in Form von Gleichung 36.

$$r^{|\mathbb{Z}_m^*|} \equiv 1 \pmod{m}$$

Speziell im Restklassenkörper $\mathbb{Z}_p$ entspringt aus Kongruenz 54 mit $\phi(p) = |\mathbb{Z}_p^*| = p - 1$ sofort der kleine Satz von Fermat.

Für einen speziellen Fall, nämlich das Produkt zweier Primzahlen $m = pq$, ist es sehr wünschenswert die Totient-Funktion zu kennen.[78] Sind p und q nach Voraussetzung Primzahlen, dann können nur die Zahlen (kleiner als m) gemeinsame Teiler mit m haben, die Vielfache von p *oder* q sind. Vielfache von p die kleiner als m sind, gibt es aber genau $q-1$, was für q äquivalent gilt (nämlich $p, 2p, 3p, \ldots, (q-1)p$ und $q, 2q, 3q, \ldots, (p-1)q$). Somit muß man von den $m-1$ Zahlen kleiner als m genau $p-1+q-1 = p+q-2$ subtrahieren, was zu

$$\begin{aligned} \phi(m) &= m - 1 - (p + q - 2) \\ &= pq - (p+q) + 1 \\ &= (p-1)(q-1) \end{aligned}$$

führt.[79] In ähnlicher Art und Weise kann man auch die folgende Formel ableiten:

[76] Die Zahl Eins wird immer als teilerfremd angesehen, d. h. $\phi(m) \geq 1$.

[77] Manchmal wird Euler's Totient-Funktion auch als Ordnung der multiplikativen Gruppe $\mathbb{Z}_m^*$ definiert.

[78] Dieser Fall hat besondere Bedeutung im Zusammenhang mit dem RSA-Algorithmus (siehe z. B. [Sch96, MvV92, Wel01] oder [RSA78]).

[79] Für den allgemeineren Fall zweier teilerfremder Zahlen gilt: $\phi(pq) = \phi(p)\phi(q)$. Man erkennt außerdem, daß in beiden Fällen $\phi(pq)$ dem kleinsten gemeinsamen Vielfachen entspricht: $\phi(pq) = \operatorname{lcm}(p,q)$.

$$\phi(p^n) = \phi(p \cdot p^{n-1}) = (p-1)p^{n-1} \,.$$

C.3.3. Beispiele

Körper $\mathbb{Z}_2$ Der Körper $\mathbb{F}_2 := \mathbb{Z}_2$ ist von besonderer praktischer Bedeutung, denn er bildet häufig die Grundlage technischer Realisierungen. Die beiden Elemente von $\mathbb{Z}_2$ werden mit $\{[0]_2, [1]_2\}$ oder kürzer mit $0, 1$ bezeichnet. Wegen ihrer einfachen Implementierung sind die Operationen in $\mathbb{Z}_2$ besonders effizient.

1. Die Addition (modulo 2) ist mit

$$\begin{aligned} 0+0 &= 0 & 1+1 &= 2 \bmod 2 = 0 \\ 0+1 &= 1 & 1+0 &= 1 \end{aligned} \tag{55}$$

   geradezu primitiv und entspricht dem logischen *Exklusiv-Oder*.[80] Wie man sofort sieht, ist das neutrale Element die 0 und wegen Beziehung 55 das additiv Inverse die 1. Addition und Subtraktion sind in diesem Sinne gleichwertig, denn es gilt $-1 \equiv 1 \pmod 2$.

2. Ebenfalls eine einfache Operation ist die Multiplikation, denn sie entspricht dem logischen *Und*.

$$\begin{aligned} 0 \cdot 0 &= 0 & 1 \cdot 1 &= 1 \\ 0 \cdot 1 &= 0 & 1 \cdot 0 &= 0 \end{aligned}$$

3. Die Division erklärt sich mit Hilfe des inversen Elements in $\mathbb{Z}_2^* = \mathbb{Z}_2 \setminus \{0\} = \{1\}$, welches ja die Bedingung $1 \cdot 1^{-1} = e_{(\cdot)} = 1$ erfüllen muß. Einzig mögliche Schlußfolgerung ist die, daß es sich bei dem inversen Element 1^{-1} um 1 selbst handelt.

Körper $\mathbb{Z}_5$ Für den Körper $\mathbb{Z}_5$, also dem Fall einer multiplikativen Gruppe der Ordnung $|\mathbb{Z}_5^*| = q - 1 = 4$, gilt für die Ordnung der Elemente:

$$\begin{array}{llllll} r_1 = 1 & \langle r_1 \rangle = \{1\} & k = 1 & \phi(k) = 1 & r_1^k = 1 & r_1^{p-1} = 1 \pmod 5 \\ r_2 = 2 & \langle r_2 \rangle = \{1,2,4,3\} & k = 4 & \phi(k) = 2 & r_2^k = 16 \equiv 1 & r_2^{p-1} = 16 \equiv 1 \pmod 5 \\ r_3 = 3 & \langle r_3 \rangle = \{1,3,4,2\} & k = 4 & \phi(k) = 2 & r_3^k = 81 \equiv 1 & r_3^{p-1} = 81 \equiv 1 \pmod 5 \\ r_4 = 4 & \langle r_4 \rangle = \{1,4\} & k = 2 & \phi(k) = 1 & r_4^k = 16 \equiv 1 & r_4^{p-1} = 256 \equiv 1 \pmod 5 \,. \end{array}$$

[80] Oftmals auch mit *XOR* oder dem Symbol $\oplus$ bezeichnet.

C.4. Erweiterungskörper

C.4.1. Vorbetrachtungen

Ein Erweiterungskörper M/K ist ein Körper $(M,+,\cdot)$, der einen anderen Körper $(K,+,\cdot)$ als Teilkörper enthält [PW72, 6.5]. Der Grad der Körpererweiterung von M über K ist die Dimension von M als (so genannter) K-Vektorraum und wird als $[M:K]$ bzw. $\dim_K M$ geschrieben. Jeder Vektor in M besteht entsprechend der Definition des Vektorraumes (vgl. Abschnitt A.4) aus jeweils $[M:K]$ Tupeln in K. Bekannte Beispiele für Körpererweiterungen sind:

$[\mathbb{C}:\mathbb{R}]=2,$ die Erweiterung der Dimension um eine imaginäre Komponente;

$[\mathbb{R}:\mathbb{Q}]=\infty,$ hier sind die rationalen Zahlen noch abzählbar.

C.4.2. Polynomringe

Ausgehend von den Vorbetrachtungen konstruieren wir jetzt einen endlichen Polynomring $(K[x],+,\cdot)$ auf dem Körper K.

$$K[x] := \left\{a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+\cdots+a_2x^2+a_1x+a_0\right\} = \{f(x)|\, n\in\mathbb{N},\ a_i\in K\} \tag{56}$$

$$f(x)=\sum_{\nu=0}^{n-1} a_\nu x^\nu \tag{57}$$

Darin seien die üblichen Polynomoperationen, wie Addition und Multiplikation gültig, weshalb man auch von einem Vektorraum der Polynome in der Unbestimmten x mit Koeffizienten aus dem Körper K spricht. Ist der Leitkoeffizient $a_{n-1}=1$, dann wird das Polynom als normiert (monisch) bezeichnet, sonst ist $a_{n-1}x^{n-1}$ das so genannte Leitmonom.

Wird anschließend eine Restklassendivision dieser Polynome $f(x)$ durch ein Polynom $m(x)$ mit Grad n definiert, also $r(x)=f(x) \bmod m(x)$ mit $r(x)\in K[x]/m(x)$, dann bildet die Menge der darin enthaltenen Restklassen wieder einen Restklassenring [PW72, 6.], [MvV92, 2.5.4]:

$\mathbb{Z}_p[x]/m(x)$ Polynom-Restklassenring auf dem Grundkörper $\mathbb{Z}_p$

1. Da bei einer Polynomaddition die einzelnen Koeffizienten unabhängig voneinander (und jeder für sich) addiert werden und außerdem nach Voraussetzung immer $\deg r(x) < \deg m(x) = n$ gilt, bildet $K[x]/m(x)$ eine additive Abel'sche Gruppe.

   a) Das Assoziativgesetz gilt: $r(x)+[g(x)+h(x)]=[r(x)+g(x)]+h(x)$ mit $r(x),g(x),h(x)\in K[x]/m(x)$.

   b) Das neutrale Element $e_{(+)}=0\in K[x]/m(x)$ mit der Beziehung $r(x)+e_{(+)}=r(x)$ ist das Nullpolynom.

c) Ein additiv inverses Element $-r(x) \in K[x]/m(x)$ mit $r(x) + [-r(x)] = 0$ ist vorhanden. Es ergibt sich aus den inversen Elementen der Koeffizienten $a_\nu \in K$ zu $-r(x) = \sum_{\nu=0}^{n-1} -a_\nu x^\nu = -\sum_{\nu=0}^{n-1} a_\nu x^\nu$.

2. $(K[x]/m(x), \cdot)$ ist eine multiplikative Halbgruppe, denn:

   a) Aufgrund der Modulo-Reduktion ist $(K[x]/m(x), \cdot)$ abgeschlossen, d. h. wenn $r(x), g(x) \in K[x]/m(x)$ angenommen wird, dann gilt für die Multiplikation $r(x)g(x) \equiv h(x) \pmod{m(x)}$ gleichfalls $h(x) \in K[x]/m(x)$.

   b) Auch das Assoziativgesetz $r(x)[g(x)h(x)] = [r(x)g(x)]h(x)$ ist in einem Restklassenring von Polynomen erfüllt.

3. Aus den bisherigen Erkenntnissen zu Restklassenringen ist im Zusammenhang mit Polynomoperationen zu schlußfolgern, daß das Distributivgesetz ebenfalls gilt: $r(x)[g(x) + h(x)] = r(x)g(x) + r(x)h(x)$.

C.4.3. Endlicher Erweiterungskörper

Der Übergang zu einem Körper wird möglich, wenn $m(x)$ ein Primelement in Bezug auf die Menge der Polynome $K[x]$ ist, es sich also um ein (so genanntes) irreduzibles Polynom handelt. Ein solches Polynom ist dadurch gekennzeichnet, daß es nicht weiter in Teilpolynome mit Koeffizienten aus K reduzierbar ist.[81]

Es sei nun $r(x)$ ein Restklassenpolynom mit Koeffizienten a aus dem endlichen Grundkörper $K := \mathbb{F}_p$ und $m(x)$ vom Grad n. Dann handelt es sich bei $\mathbb{F}_p[x]/m(x)$ um einen endlichen Körper mit $q = p^n$ Elementen [Bos96, 3.8]. Man spricht auch von einem Vektorraum V der Dimension n über $\mathbb{F}_p$, denn auf diese Weise wird (im Sinne von Abschnitt A.4) jedem Vektor $\mathbf{v} = (v_1, v_2, v_3, \ldots, v_n)$ ein Polynom $r(x)$ vom Grad $n-1$ zugeordnet. Es handelt sich folglich nur um eine andere Darstellung der n Tupel des Vektors $\mathbf{v}$ in der Art $a_0 = v_1, a_1 = v_2, \ldots, a_{n-1} = v_n$. Die Potenzen $x^0, x^1, x^2, \ldots, x^{n-2}, x^{n-1}$ bilden die (Polynom-) Basis des Vektorraumes V über $\mathbb{F}_p$. Entsprechend ist die Dimension des Erweiterungskörpers $\mathbb{F}_q$ über dem Grundkörper $\mathbb{F}_p$ genau $[\mathbb{F}_q : \mathbb{Z}_p] = n$. Als Notation für einen solchen Körper wird deshalb auch $\mathbb{F}_p^n$ verwendet, oftmals auch $\mathbb{F}_{p^n}$ oder $\mathrm{GF}(p^n)$.

Mit diesen Vorbemerkungen lassen sich alle Aussagen zu Restklassenkörpern, wie sie in Abschnitt C.3 allgemein formuliert wurden, auf den Erweiterungskörper $\mathbb{F}_{q=p^n}$ anwenden:

1. Das Modul m (Primelement) wird nun als das irreduzible Polynom $m(x)$ interpretiert.
2. Die Restklassendivision ist definiert als $r(x) \equiv f(x) \bmod m(x)$, was grundsätzlich immer zu einem Grad kleiner als n für $r(x)$ führt.[82] Die Kennzeichnung der zu $r(x)$ gehörenden Restklasse erfolgt wie gewohnt mit $[r(x)]_{m(x)}$, wird meistens jedoch weggelassen. Das

[81] Ausführliche Betrachtungen zu irreduziblen und primitiven Polynomen sind z. B. in [MvV92, 4.5], Tabellen irreduzibler Polynome in [PW72, Anhang C] zu finden.

[82] Die Menge der Elemente r umfaßt alle Polynome mit einem Grad kleiner als n.

Element $r(x)$ steht also auch hier wieder als Restklassenvertreter aller Polynome $f(x)$, welche die Bedingung $f(x) = s(x)m(x) + r(x)$ mit $\deg r(x) < \deg m(x)$ erfüllen.

3. Das Nullelement ist das Nullpolynom $r(x) = 0 \pmod{m(x)}$ bzw. dessen Restklasse $[0]_{m(x)}$, das Einselement das Einheitspolynom $e_{(\cdot)} = x^0 = 1$.

4. Addition und Multiplikation (von Polynomen) in $\mathbb{F}_q$ sind wohldefiniert und abgeschlossen, die entsprechenden Gruppen $\mathbb{F}_q^*$ und $\mathbb{F}_q^+$ also existent.

5. Die Anzahl der Elemente $r(x)$ im Restklassenkörper $\mathbb{F}_p[x]/m(x)$ ist aufgrund der Anzahl von möglichen Koeffizientenkombinationen p^n. Bei $\mathbb{F}_q$ handelt es sich folglich um einen Galois-Körper $\mathrm{GF}(p^n)$.[83] Wegen $|\mathbb{F}_q| = q = p^n$ hat dessen multiplikative Gruppe $\mathbb{F}_q^*$ die Ordnung $p^n - 1$.

6. Jedes Element $r(x) \in \mathbb{F}_q^*$ hat eine Ordnung bzw. Periode $k = |\langle r(x) \rangle|$, welche sich aus Gleichung 28 ableitet:

$$[r(x)]^k - 1 \equiv 0 \pmod{m(x)}\,.$$

7. Nach dem Satz von Lagrange teilt die Ordnung k des Elements die der multiplikativen Gruppe $\mathbb{F}_q^*$, d. h. $q - 1 = p^n - 1 = ik$ und demzufolge ist

$$[r(x)]^{q-1} - 1 \equiv 0 \pmod{m(x)}\,. \tag{58}$$

8. Jedes erzeugende (primitive) Element $g(x)$ von $\mathbb{F}_q^*$ erfüllt die Bedingung der maximalen Zykluslänge (Index $i = 1$), hat also die Ordnung $|\langle g(x) \rangle| = p^n - 1$. Es ist damit geeignet, als Basiselement für die Erzeugung aller anderen Elemente $r(x) \in \mathbb{F}_q^*$ verwendet zu werden. Nimmt man das Nullelement $0 = g^q$ sowie das Einselement $1 = g^{q-1}$ hinzu, so gilt für die Menge der Elemente $\mathbb{F}_q = \{0, 1, g^1, g^2, g^3, \ldots, g^{p^n-3}, g^{p^n-2}\}$.

9. Aus Punkt 6 läßt sich (in Übereinstimmung mit Gleichung 52) schlußfolgern, daß für jedes Element $r(x) \in \mathbb{F}_q^*$ das Modul $m(x)$ ein Teiler von $[r(x)]^{q-1} - 1$ ist, also die Zerlegung $[r(x)]^{q-1} - 1 = h(x)\,m(x)$ hat. Setzt man insbesondere $r(x) = x$ als das kleinste Element (mit einem Grad größer als 0) aus $\mathbb{F}_q^*$, so erhält man die Beziehungen:

$$\begin{aligned} x^{q-1} - 1 &= h(x)\,m(x) \\ x^{q-1} - 1 &\equiv 0 \pmod{m(x)} \\ x^q - x &\equiv 0 \pmod{m(x)}, \end{aligned}$$

welche auch die Kongruenz $h(x)\,m(x) \equiv 0 \pmod{x^{q-1} - 1}$ rechtfertigen.

[83] Ein Erweiterungskörper schafft dadurch die Möglichkeit, daß auch Potenzen von Primelementen noch als Körperordnung zulässig sind.

C.4.4. Zerfällungskörper

Nach dem Hauptsatz der Zahlentheorie kann man für jede natürliche Zahl eine eindeutigen Primfaktorzerlegung der Form

$$m_1^{e_1} m_2^{e_2} m_3^{e_3} \cdots$$

finden. Gleiches trifft auch für Polynome in $\mathbb{F}_p[x]$ zu, nur daß es sich um irreduzible Polynome anstatt Primzahlen handelt.

$$f(x) = [m_1(x)]^{e_1} \, [m_2(x)]^{e_2} \, [m_3(x)]^{e_3} \cdots$$

Jedes der irreduziblen Polynome[84] $m_i(x)$ hat eine vom jeweiligen Grad n abhängige Anzahl von Nullstellen α, die entweder im Grundkörper $\mathbb{F}_p$ oder (sämtlich) in einem zugehörigen Erweiterungskörper $\mathbb{F}_{p^n}$ liegen. Nullstellen im Grundkörper, von denen $f(x)$ maximal $p = |\mathbb{F}_p|$ besitzen kann, lassen sich immer als einfache Faktoren der Art $m(x) = x - \alpha$ mit $\alpha \in \mathbb{F}_p$ darstellen (vgl. Beispiel-Faktorisierung von $x^3 - 1$ in Abschnitt C.4.5). Liegen dagegen alle n Wurzeln von $m(x)$ in einem Erweiterungskörper, dann muß es sich um ein irreduzibles Polynom (höheren Grades) handeln.[85] Ein solcher Körper besteht aus p^n Elementen, welche die $q = p^n$ Nullstellen der zugeordneten Funktion $\psi(x) = x^q - x$ darstellen (vgl. Abschnitt B.1.5). $\mathbb{F}_{p^n}$ nennt man deshalb auch den kleinsten Körper über den $\psi(x) \in \mathbb{F}_p[x]$ vollständig in Linearfaktoren zerfällt [Bos96, 4.5] bzw. kürzer: $\mathbb{F}_{p^n}$ sei der Zerfällungskörper von $x^q - x$.

$$\psi(x) = x^q - x = \prod_{\alpha \in \mathbb{F}_q} (x - \alpha), \qquad \psi(x) \in \mathbb{F}_p[x]$$

Ein weiteres Charakteristikum des Zerfällungskörpers $\mathbb{F}_{p^n}$ sind die konjugierten Nullstellen, d. h. bei Kenntnis einer Nullstelle $\alpha \neq 0$ des irreduziblen Polynoms $m(\alpha) = 0$ sind die restlichen $n - 1$ Nullstellen genau die Potenzen $\alpha^p, \alpha^{p^2}, \ldots, \alpha^{p^{n-2}}, \alpha^{p^{n-1}}$. Das irreduzible Polynom $m(x)$ zerfällt also bei Kenntnis nur *einer* Nullstelle α vollständig in seine n Linearfaktoren. Der induktive Beweis dieses Satzes geht von der Hilfsformel

$$(b + c)^p = \sum_{k=0}^{p} \binom{p}{k} b^{p-k} c^k = b^p + c^p + \sum_{k=1}^{p-1} \binom{p}{k} b^{p-k} c^k = b^p + c^p$$

[84] Die Indizierung mit i wird im weiteren wieder weggelassen, da wir uns auf die Betrachtung einer Funktion $m(x) := m_i(x)$ beschränken. Man sollte sich aber immer bewußt sein, daß die Variablen n, q sowie die abgeleiteten Größen bzw. Funktionen $\psi(x)$ abhängig von $m(x)$ variieren.

[85] Und umgekehrt, denn $m(x)$ ist ja im Grundkörper nicht weiter faktorisierbar.

aus, welche berücksichtigt, daß:

- mit $b, c \in \mathbb{F}_p$ der Binomialkoeffizient $\binom{p}{k} = \frac{p!}{k!(p-k)!}$ für $k \neq 0$, $p!$ immer durch p teilbar ist und folglich $\binom{p}{k} \bmod p = 0$ gilt;
- im Grundkörper $\mathbb{F}_p$ jedes Element a die Relation $a^p = a$ erfüllt (vgl. Abschnitt C.3.2).

$$m(\alpha^p) = \sum_{i=0}^{n} a_i \alpha^{ip} = \sum_{i=0}^{n} a_i^p \alpha^{ip} = \sum_{i=0}^{n} \left(a_i \alpha^i\right)^p = \left(\sum_{i=0}^{n} a_i \alpha^i\right)^p = [m(\alpha)]^p = 0$$

Aus diesem Grund läßt sich für jedes der irreduziblen Polynome $m(x)$ die folgende Linearfaktordarstellung angeben:[86]

$$m(x) = \prod_{\nu=0}^{n-1} \left(x - \alpha^{p^\nu}\right) .$$

Die Nullstellen α^{p^ν} kann man (wegen ihrer linearen Unabhängigkeit) verwenden, um statt einer Polynombasis eine so genannte Normalbasis des Vektorraumes (der Dimension n) über $\mathbb{F}_p$ zu definieren.

C.4.5. Erweiterungskörper $\mathbb{Z}_2^n$

Für eine Erweiterung des Primkörpers $\mathbb{Z}_2$ auf n Dimensionen ist ein irreduzibles Polynom $m(x)$ vom Grad n notwendig. Der dadurch entstehende Körper $\mathbb{Z}_2^n$ soll am Beispiel des Polynoms $m(x) = x^2 + x + 1$ mit Koeffizienten aus $\mathbb{Z}_2$ (zu den Rechenoperationen vgl. Seite 66) jetzt kurz betrachtet werden. Nach Darstellung 56 gehören genau $q = p^n = 4$ Polynome zum Erweiterungskörper ($p = 2$, $n = 2$), nämlich:

$$r_0(x) = 0 \qquad r_1(x) = 1$$
$$r_2(x) = x \qquad r_3(x) = x + 1 .$$

Wie sich leicht feststellen läßt, ist $r_3(x)$ das erzeugende Element der multiplikativen Gruppe $\{r_1(x), r_2(x), r_3(x)\}$, denn die anderen Elemente ergeben sich als Potenzen $r_3^\nu(x) \bmod m(x)$.

$$(x+1)^0 = 1$$
$$(x+1)^1 = x + 1$$
$$(x+1)^2 = x^2 + x + x + 1 = x^2 + 1 \equiv x \pmod{x^2 + x + 1}$$

[86] Das irreduzible Polynom $m(x)$ wird auch Minimalpolynom von α über $\mathbb{F}_p$ genannt. Bei einer Minimalfunktion handelt es sich allgemein um ein normiertes Polynom $m(x) \in \mathbb{F}_p[x]$ *kleinsten* Grades, welches beim Einsetzen eines Elementes $\alpha \in \mathbb{F}_q^*$ die Gleichung $m(\alpha) = 0$ erfüllt [PW72, 6.5].

Außerdem sind alle Elemente (abgesehen von $r_0(x)$) wirklich Nullstellen des Polynoms $x^{q-1} - 1 = x^3 - 1 = (x-1)m(x)$. Auch gut erkennen läßt sich, daß $m(x)$ für jedes Element $r(x)$ ein Teiler von $[r(x)]^{q-1} - 1$ ist.

$$[r_1(x)]^3 - 1 = 0$$
$$[r_2(x)]^3 - 1 = x^3 - 1 = (x-1)(x^2+x+1) \equiv 0 \pmod{x^2+x+1}$$
$$[r_3(x)]^3 - 1 = (x+1)^3 - 1 = x^3 + x^2 + x = x(x^2+x+1) \equiv 0 \pmod{x^2+x+1}$$

Äquivalent dazu ist das Produkt aller Elemente (konform zu Formel 42) genau das neutrale Element $e_{(\cdot)}$ der multiplikativen Gruppe.[87]

$$r_1(x) \cdot r_2(x) \cdot r_3(x) = 1 \cdot x \cdot (x+1) = x^2 + x \equiv 1 \pmod{x^2+x+1}$$

D. Algorithmen

D.1. GCD-Algorithmen

D.1.1. Euklidischer Algorithmus

Der Algorithmus von Euklid berechnet den größten gemeinsamen Teiler $d = \gcd(a,b)$ zweier natürlicher Zahlen $a,b \in \mathbb{N}$. Die grundlegende, iterativ angewendete Rechenoperation dabei ist modulare Division.[88] Algorithmus 2 beschreibt das klassische Verfahren (vgl. [Knu98, Sho05, CP05]). Es beginnt unter der Voraussetzung $a > b$ mit einer Modulo-Division $c_2 = a \bmod b$. Im nächsten Schritt wird c_2 als Modul verwendet und $c_3 = b \bmod c_2$ berechnet, wonach $c_4 = c_2 \bmod c_3$ folgt usw. . Diese (wegen $c_{k+1} < c_k$) absteigende Sequenz endet wenn $c_{k+1} = 0$ wird[89] – das Ergebnis d befindet sich dann in c_k.

Beweis[90] Jeder Iterationsschritt $c_{i+1} = c_{i-1} \bmod c_i$ kann in der Umkehrung (vgl. Restklassenbeziehung 48 in Abschnitt C) als

$$c_{i-1} = q_{i+1}c_i + c_{i+1}, \qquad 0 \leq c_{i+1} < c_i$$

[87] Welches in der additiven Gruppe $(\mathbb{Z}_2, \cdot)$ gleich dem Inversen von 1 ist, d. h. zur Erinnerung $-1 \equiv +1$.

[88] Die asymptotische Komplexität des klassischen Algorithmus wird in [MvV92, 2.4.2] mit $O(\lg^2 n)$ angegeben, was in starkem Maße durch die fortlaufenden Divisionen bestimmt wird. Zur Komplexität verschiedener Varianten des GCD-Algorithmus' siehe auch [Har06].

[89] Das diese stetig absteigende Sequenz mit dem Wert 0 endet, ist eine fundamentale Eigenschaft natürlicher Zahlen.

[90] Siehe auch [MvV92, 2.4.2], [Ber84, 2.1], [Wel01, 10.1].

geschrieben werden. So gesehen wird durch den Algorithmus der folgende Abstieg vorgenommen:

$$\begin{aligned}
a = c_0 &= q_2 c_1 + c_2 && (c_2 < c_1)\\
b = c_1 &= q_3 c_2 + c_3 && (c_3 < c_2)\\
c_2 &= q_4 c_3 + c_4 && (c_4 < c_3)\\
&\vdots\\
c_{k-3} &= q_{k-1} c_{k-2} + c_{k-1} && (c_{k-1} < c_{k-2})\\
c_{k-2} &= q_k c_{k-1} + c_k && (c_k < c_{k-1})\\
c_{k-1} &= q_{k+1} c_k + 0 && (c_{k+1} = 0)
\end{aligned}$$

bis c_{k+1} verschwindet. Warum $d = c_k$ ein gemeinsamer Faktor von a und b ist, wird klar wenn man die Folge rückwärts betrachtet. In der letzten Zeile steht $c_{k-1} = q_{k+1}d$, also teilt d den Rest c_{k-1} (oder kürzer $d \mid c_{k-1}$). Wenn d aber als Faktor in c_{k-1} enthalten ist, dann kann man d auf der rechten Seite der vorletzten Gleichung ausklammern, weshalb es auch als Faktor in c_{k-2} vorkommen muß. Dies setzt sich bis in die erste Gleichung fort (sämtliche Divisionsreste $c_0, c_1, \ldots, c_k$ enthalten folglich d als Teiler), in der die Startbedingung $c_0 = a$ und $c_1 = b$ verankert ist. Deshalb ist der gemeinsame Teiler d sowohl in a als auch b enthalten.

Viel kürzer kann man auch damit argumentieren, daß die Modulo-Division $c_{i+1} = c_{i-1} \bmod c_i$ eine GCD erhaltende Operation ist. Denn mit der Faktorisierung $c_i = \overline{c}_i d$ gilt (ausgehend von $c_0 = a$, $c_1 = b$) folgende Beziehung:

$$c_{i+1} = c_{i-1} \bmod c_i = c_{i-1} - q_i c_i = \overline{c}_{i-1} d - q_i \overline{c}_i d = d(\overline{c}_{i-1} - q_i \overline{c}_i)$$

d. h. der gemeinsame Teiler d in c_{i-1} und c_i ist auch in c_{i+1} wieder enthalten.[91]

[91] So gilt allgemein: $\gcd(a,b) = \gcd(a \bmod b, b)$.

Algorithmus 2 Euklidischer Algorithmus $d = \gcd(a,b)$

Require: $a \geq b$
  $c_0 \Leftarrow a$, $c_1 \Leftarrow b$
  $k \Leftarrow 0$
  repeat
    $k \Leftarrow k+1$
    $c_{k+1} \Leftarrow c_{k-1} \bmod c_k$
  until $c_{k+1} = 0$
  $\gcd(a,b) \Leftarrow c_k$

$$\gcd(c_{i-1}, c_i) = \gcd(c_{i+1}, c_i) = \gcd(c_{i-1} \bmod c_i, c_i) \tag{59}$$

Damit d wirklich den größten gemeinsamen Teiler stellt, muß es überhaupt alle gemeinsamen Teiler enthalten. Mit dem Ziel dies nachzuweisen betrachten wir nochmals die Folge beginnend mit der vorletzten Gleichung, welche nach $c_k = d$ umgestellt wird. Ersetzt man darin c_{k-1} mit Hilfe der vorvorletzten Gleichung und fährt aufsteigend fort, so erhält man eine lineare Darstellung für $d = \gcd(a, b)$,

$$\begin{aligned} d &= c_{k-2} - q_k c_{k-1} \\ &= c_{k-2} - q_k(c_{k-3} - q_{k-1}c_{k-2}) = c_{k-2}(1 + q_k q_{k-1}) - c_{k-3}q_k \\ &= (c_{k-4} - q_{k-2}c_{k-3})(1 + q_k q_{k-1}) - c_{k-3}q_k = c_{k-4}(1 + q_k q_{k-1}) - c_{k-3}\left[q_{k-2}(1 + q_k q_{k-1}) + q_k\right] \\ &\vdots \\ &= \alpha c_0 + \beta c_1 \end{aligned}$$

welche auf ganzen Zahlen $\alpha, \beta \in \mathbb{Z}$ sowie $c_0 = a$ und $c_1 = b$ beruht.

$$d = \gcd(a, b) = \alpha a + \beta b \tag{60}$$

Mit Hilfe von Formel 60, welche auch Satz von Bézout genannt wird,[92] kann jetzt relativ einfach bewiesen werden, daß d wirklich der größte gemeinsame Teiler ist. Nehmen wir dazu an, es gäbe einen weiteren gemeinsamen Teiler d'. Dann würde dieser als Faktor auf der rechten Seite von Gleichung 60 auszuklammern sein und deshalb (wenn man die linke Seite betrachtet) als Teiler von d auftreten. Anders formuliert, stecken alle weiteren Teiler (schon) in d, weshalb nur dieser der größte gemeinsame Teiler sein kann.

Hinweis Es existieren unendlich viele lineare Darstellungen für d als Funktion von a und b (vgl. auch die kurze Betrachtung zu diophantischen Gleichungen auf Seite 86). Jede Substitution der Art $\alpha := \alpha + n\overline{b}$ und $\beta := \beta - n\overline{a}$, mit

$$a = d\overline{a} \qquad b = d\overline{b} \qquad 1 = \gcd(\overline{a}, \overline{b}) = \alpha\overline{a} + \beta\overline{b} \tag{61}$$

erfüllt Bézout's Identität 60.

[92] Die Bézout-Identität nach Formel 60 kann man für $\alpha, \beta \in \mathbb{N}$ auch als $d = \pm\alpha a \mp \beta b$ schreiben.

$$\begin{aligned} d &= (\alpha + n\overline{b})a + (\beta - n\overline{a})b \\ &= \alpha a + \beta b + n(\overline{b}a - \overline{a}b) \\ &= \alpha a + \beta b + n(\overline{b}\overline{a}d - \overline{a}\overline{b}d) \\ &= \alpha a + \beta b \end{aligned}$$

Die kleinsten Werte für $|\alpha|$ und $|\beta|$ zeichnen sich folglich durch die Relationen $|\alpha| < \overline{b}$ und $|\beta| < \overline{a}$ aus. Um sie zu ermitteln kann man entweder $\overline{a}$ und $\overline{b}$ sukzessive von α und β subtrahieren/addieren oder man reduziert die Kofaktoren mit Hilfe von $q = \left\lfloor |\alpha|/\overline{b} \right\rfloor$ bzw. $q = \lfloor |\beta|/\overline{a} \rfloor$ und folgender Formeln:[93]

$$\alpha := \alpha \mp q\overline{b} \qquad\qquad \beta := \beta \pm q\overline{a}\,.$$

D.1.2. Erweiterter euklidischer Algorithmus

Der erweiterte euklidische Algorithmus erlaubt eine effiziente Berechnung der Bézout-Kofaktoren α und β zusammen (und gleichzeitig) mit dem größten gemeinsamen Teiler (siehe auch [Knu98, Sho05, CP05, Ber84]). Dazu berücksichtigt er einige Erkenntnisse aus dem vorigen Abschnitt, insbesondere daß:

- der größte gemeinsame Teiler $d = \gcd(a,b)$ für $\alpha,\beta \in \mathbb{Z}$ als Linearkombination $\alpha a + \beta b$ darstellbar ist;
- d für $0 \le i \le k$ jeden Rest c_i teilt und damit die Darstellung $c_i = d\overline{c}_i = \alpha_i a + \beta_i b$ rechtfertigt;
- die Berechnung von c_{i+1} mit Hilfe von $q_{i+1} = \lfloor c_{i-1}/c_i \rfloor$ und $c_{i+1} = c_{i-1} - q_{i+1}c_i$ erfolgen kann.

Durch Induktion ergibt sich aus

$$c_i = c_{i-2} - q_i c_{i-1} = d\overline{c}_i = \alpha_i a + \beta_i b \tag{62}$$

durch Einsetzen von $c_{i-1} = \alpha_{i-1}a + \beta_{i-1}b$ und $c_i = \alpha_i a + \beta_i b$:

$$\begin{aligned} c_{i+1} &= c_{i-1} - q_{i+1}c_i = \alpha_{i+1}a + \beta_{i+1}b \\ &= \alpha_{i-1}a + \beta_{i-1}b - q_{i+1}(\alpha_i a + \beta_i b) \\ &= (\alpha_{i-1} - q_{i+1}\alpha_i)a + (\beta_{i-1} - q_{i+1}\beta_i)b\,. \end{aligned}$$

[93] Sollte man nur an einem Kofaktor (beispielhaft α) interessiert sein, dann kann auf die Berechnung von q ganz verzichtet und statt dessen $\alpha := \alpha \bmod \overline{b}$ gerechnet werden.

Vergleich mit Ausgangsformel 62 erlaubt die Bestimmung von α_{i+1} und β_{i+1}.

$$\alpha_{i+1} = \alpha_{i-1} - q_{i+1}\alpha_i \qquad \beta_{i+1} = \beta_{i-1} - q_{i+1}\beta_i$$

Mit den Startwerten $\alpha_0 = 1$ und $\beta_0 = 0$ (gewährleistet $c_0 = a$) bzw. $\alpha_1 = 0$ und $\beta_1 = 1$ (ebenso für $c_1 = b$) kann man nun den erweiterten euklidischen Algorithmus 3 formulieren.

Algorithmus 3 Erweiterter euklidischer Algorithmus

Require: $a \geq b$
  $c_0 \Leftarrow a,\ c_1 \Leftarrow b,\ \alpha_0 \Leftarrow 1,\ \alpha_1 \Leftarrow 0,\ \beta_0 \Leftarrow 0,\ \beta_1 \Leftarrow 1$
  $k \Leftarrow 0$
  repeat
    $k \Leftarrow k+1$
    $q \Leftarrow \lfloor c_{k-1}/c_k \rfloor$ {Integer-Division}
    $c_{k+1} \Leftarrow c_{k-1} - qc_k$ {Modulo-Division $c_{k+1} = c_{k-1} \bmod c_k$}
    $\alpha_{k+1} \Leftarrow \alpha_{k-1} - q\alpha_k$
    $\beta_{k+1} \Leftarrow \beta_{k-1} - q\beta_k$
  until $c_{k+1} = 0$
  $d \Leftarrow c_k,\ \alpha \Leftarrow \alpha_k,\ \beta \Leftarrow \beta_k$

D.1.3. Binärer GCD-Algorithmus

Der binäre GCD-Algorithmus kommt im Gegensatz zum klassischen euklidischen Algorithmus ohne Divisionen aus [Ste67, Knu98]. Statt dessen wird (beginnend mit $a_0 = a$ und $b_0 = b$) durch Subtraktion und Halbierung eine stetige Reduktion der Argumente a und b vorgenommen, welche letztlich zum größten gemeinsamen Teiler $d = \gcd(a,b)$ führt.[94]

[94] Im Wesen unterscheidet er sich nicht vom euklidischen Algorithmus, der in ähnlicher Art und Weise die Argumente stetig reduziert und dabei den größten gemeinsamen Teiler d bewahrt. Die Halbierung eines der Argumente ist allerdings (durch Verschiebung um ein Bit) effizient zu realisieren.

Tabelle 2: Reduktionsschema des binären GCD-Algorithmus

a_i	b_i	$\gcd(a_{i+1}, b_{i+1}) =$	Begründung
gerade		$2\gcd\left(\frac{a_i}{2}, \frac{b_i}{2}\right)$	gemeinsamer Teiler ist 2
gerade	ungerade	$\gcd\left(\frac{a_i}{2}, b_i\right)$	2 ist kein gemeinsamer Teiler
ungerade	gerade	$\gcd\left(a_i, \frac{b_i}{2}\right)$	2 ist kein gemeinsamer Teiler
ungerade, $a_i \geq b_i$		$\gcd\left(\frac{a_i - b_i}{2}, b_i\right)$	$a_i - b_i$, wie auch $a_i + b_i$, enthalten den Teiler $\gcd(a_i, b_i) = d$, denn $a_i = \overline{a}_i d$, $b_i = \overline{b}_i d$ führt zu $\gcd(a_i - b_i, b_i) = \gcd[d(\overline{a}_i - \overline{b}_i), \overline{b}_i d]$. Außerdem sind sowohl Summe als auch Differenz gerade Zahlen ($a_i = 2\nu + 1$, $b_i = 2\mu + 1$ ergibt $a_i - b_i = 2(\nu - \mu)$), weshalb 2 als gemeinsamer Teiler wieder ausgeschlossen werden kann.
ungerade, $b_i \geq a_i$		$\gcd(a_i, \frac{b_i - a_i}{2})$	siehe Fall $a_i > b_i$
$a_i > 0$	0	a_i	a_i ist größter gemeinsamer Teiler mit 0
0	$b_i > 0$	b_i	b_i ist größter gemeinsamer Teiler mit 0

$$
\begin{aligned}
d &= \gcd(a_0, b_0) \\
&= \gcd(a_1, b_1) \\
&\vdots \\
&= \gcd(a_i, b_i) \\
&\vdots \\
&= \gcd(a_k, 0) = a_k
\end{aligned}
$$

Der Algorithmus endet spätestens dann, wenn im Schritt $i = k$ eines der beiden Argumente a_i oder b_i verschwindet (im obigen Fall beispielhaft für $b_k = 0$). Das Reduktionsschema zeigt Tabelle 2.

Anmerkungen

1. Bezüglich a_{k-1} gibt es genau eine Situation, die das Verschwinden von a_k hervorruft (genauso bezüglich b_{k-1} und b_k) . Betrachtet man dazu Tabelle 2, so wird a_{k-1} in folgenden Fällen reduziert:

a) a_{k-1} und b_{k-1} sind gerade: Halbierung von $a_{k-1} = 1$ ergibt 0, bedeutet aber ungerades a_{k-1} (Widerspruch).

b) a_{k-1} gerade, b_{k-1} ungerade: Halbierung von $a_{k-1} = 1$ ergibt 0, was ebenfalls ungerades a_{k-1} bedeutet (Widerspruch).

c) a_{k-1} und b_{k-1} sind ungerade: Halbierung von $b_{k-1} - a_{k-1} = 1$ ergibt 0, aber unter der Voraussetzung $b_{k-1} = a_{k-1} + 1$ können niemals beide ungerade sein (Widerspruch).

d) a_{k-1} und b_{k-1} sind ungerade: Halbierung von $b_{k-1} = a_{k-1}$ führt zu $a_k = 0$ und stellt damit den einzig möglichen Fall im Schritt $k-1$ dar.

2. In jedem Iterationszyklus wird entweder a_i oder b_i um mindestens ein Bit reduziert.[95] Aus diesem Umstand läßt sich (im Fall $a_0 > b_0$) für die Anzahl der Iterationsschritte $\lceil \log_2 a_0 \rceil \leq k \leq 2\lceil \log_2 a_0 \rceil$ schlußfolgern (vgl. auch [Kal95, Theorem 2]).

3. Die Halbierung des Arguments für den Fall, daß a_i und b_i ungerade sind, muß man nicht unbedingt im Iterationsschritt i ausführen. Es ist durchaus legitim, falls beispielsweise $a_i \geq b_i$ gilt, einfach nur $a_{i+1} = a_i - b_i$ zu berechnen. Die Differenz ergibt ja bekanntlich wieder eine gerade Zahl (dann ist a_{i+1} gerade, b_{i+1} ungerade) und es kommt im nächsten Iterationsschritt zu der gewünschten Halbierung.

4. Den Fall, daß sowohl a_i als auch b_i gerade sind, kann man grundsätzlich aus der Haupt-Iterationsschleife herausziehen. Denn wird einmal eines der beiden Argumente ungerade, dann kann dieser Fall niemals wieder eintreten (genau das ungerade Argument ist in allen anderen Reduktionsfällen unveränderlich). Solange beide Argumente gerade sind, kann man sie also kontinuierlich reduzieren, bis nach n Schritten entweder a_i oder b_i ungerade geworden ist. Danach kann auf $a_n = a/2^n$ und $b_n = b/2^n$ irgendein binärer (erweiterter) GCD-Algorithmus angewendet werden, der $\gcd(a_n, b_n) = \alpha_n a_n + \beta_n b_n$ ermittelt.

$$\begin{aligned} d &= 2^n \gcd\left(\frac{a}{2^n}, \frac{b}{2^n}\right) \\ &= 2^n(\alpha_n a_n + \beta_n b_n) = \alpha_n a + \beta_n b \end{aligned}$$

Der so ermittelte größte gemeinsame Teiler muß am Schluß nur noch mit 2^n multipliziert werden (also um n Bit verschoben), was auch Algorithmus 4 entsprechend wiedergibt.[96] Wir können deshalb in den Betrachtungen zum erweiterten binären GCD-Algorithmus (siehe nächste Abschnitte) immer voraussetzen, daß a oder b ungerade ist.

[95] Wodurch zwar mehr Schritte als beim euklidischen Algorithmus nötig sind, sich die Ausführungszeit (durch Vermeidung von Langzahldivisionen) aber typischerweise verringert. Die Komplexität des Algorithmus' bleibt unverändert quadratisch, kann aber durch eine systolische Implementierung nach [BK83, Jeb93, BB87] sogar bis auf $O(n)$ reduziert werden.

[96] Die Bézout-Koeffizienten α und β müssen übrigens nicht korrigiert werden.

Algorithmus 4 Reduktion gerader Argumente

$n \Leftarrow 0$
while a gerade $\wedge$ b gerade **do**
  $a \Leftarrow a/2$
  $b \Leftarrow b/2$
  $n \Leftarrow n+1$
end while
$d \Leftarrow 2^n \gcd(a,b)$ {GCD-Algorithmus mit Voraussetzung: a oder b ungerade}

5. Für den Fall, daß entweder a oder b ungerade ist (oder beide), kann man ausgehend von $a = d\overline{a}$ und $b = d\overline{b}$ feststellen:[97]

   - Der größte gemeinsame Teiler d ist ungerade.
   - Sollte a ungerade sein, dann ist es dessen teilerfremde Anteil $\overline{a}$ ebenfalls.
   - Sollte b ungerade sein, dann ist es dessen teilerfremde Anteil $\overline{b}$ auch.

D.1.4. Erweiterter binärer GCD-Algorithmus

Algorithmus nach Kaliski[98] [Kal95] Betrachtet man das Reduktionsschema des binären GCD-Algorithmus, so kann man verschiedenste lineare Transformationen der Art

$$a_i = u'_{i+1}a_{i+1} + v'_{i+1}b_{i+1} \qquad a_{i+1} = u''_i a_i + v''_i b_i$$
$$b_i = s'_{i+1}a_{i+1} + t'_{i+1}b_{i+1} \qquad b_{i+1} = s''_i a_i + t''_i b_i,$$

definieren, wobei sich die einzelnen Betrachtungsweisen durch unterschiedliche Werte in den Koeffizienten unterscheiden. Bei jedem Schritt sind so a_i und b_i als lineare Funktion von a_{i+1} und b_{i+1} darstellbar, die Ausgangsgrößen a_0 und b_0 deshalb als lineare Funktionen von a_i und b_i.

$$a_0 = u_i a_i + v_i b_i \qquad (u_0 = 1, v_0 = 0) \tag{63}$$
$$b_0 = s_i a_i + t_i b_i \qquad (s_0 = 0,\ t_0 = 1) \tag{64}$$

Durch Einsetzen von a_i und b_i in folgende Gleichung,

[97] Grundlage bildet die Tatsache, daß das Produkt zweier ganzer Zahlen nur dann ungerade ist, wenn beide Faktoren ungerade sind.
[98] Auch *Right Shift Delayed Halving* (RSDH) oder *Almost Montgomery Inverse* Algorithmus genannt [Har06].

Tabelle 3: Linearfaktoren beim binären GCD-Algorithmus nach Kaliski

a_i	b_i	$a_{i+1} =$	$b_{i+1} =$	$u_{i+1} =$	$v_{i+1} =$	$s_{i+1} =$	$t_{i+1} =$	$\mathbf{D_{i+1}} =$
gerade		$\frac{a_i}{2}$	$\frac{b_i}{2}$	$2u_i$	$2v_i$	$2s_i$	$2t_i$	$4\mathbf{D_i}$
gerade	ungerade	$\frac{a_i}{2}$	b_i	$2u_i$	v_i	$2s_i$	t_i	$2\mathbf{D_i}$
ungerade	gerade	a_i	$\frac{b_i}{2}$	u_i	$2v_i$	s_i	$2t_i$	$2\mathbf{D_i}$
ungerade, $a_i \geq b_i$		$\frac{a_i - b_i}{2}$	b_i	$2u_i$	$u_i + v_i$	$2s_i$	$s_i + t_i$	$2\mathbf{D_i}$
ungerade, $b_i \geq a_i$		a_i	$\frac{b_i - a_i}{2}$	$u_i + v_i$	$2v_i$	$s_i + t_i$	$2t_i$	$2\mathbf{D_i}$

$$a_0 = u_i a_i + v_i b_i = u_{i+1} a_{i+1} + v_{i+1} b_{i+1}$$
$$b_0 = s_i a_i + t_i b_i \ = s_{i+1} a_{i+1} + t_{i+1} b_{i+1}$$

gefolgt von einem Koeffizientenvergleich, kann man für die konkreten Reduktionsfälle die jeweilige Transformationen der Linearfaktoren ableiten (siehe Tabelle 3). Am Beispiel $a_{i+1} = (a_i - b_i)/2$, $b_{i+1} = b_i$ soll das Vorgehen exemplarisch verdeutlicht werden.

$$\begin{aligned} s_i a_i + t_i b_i &= s_{i+1} \frac{a_i - b_i}{2} + t_{i+1} b_i \\ &= \frac{s_{i+1}}{2} a_i + \left(t_{i+1} - \frac{s_{i+1}}{2} \right) b_i \\ &= \underbrace{\frac{s_{i+1}}{2}}_{s_i} a_i + \underbrace{(t_{i+1} - s_i)}_{t_i} b_i \end{aligned}$$

Aus den Transformationen nach Tabelle 3 kann man außerdem die Determinante

$$\mathbf{D_i} = u_i t_i - s_i v_i = \begin{vmatrix} u_i & v_i \\ s_i & t_i \end{vmatrix}, \text{ mit } \mathbf{D_0} = 1$$

bestimmen.

Um den Zusammenhang mit den Bézout-Koeffizienten $\alpha, \beta \in \mathbb{Z}$ in $\gcd(a,b) = \alpha a_0 + \beta b_0$ herzustellen, stellen wir die Formeln 63 und 64 nach a_i und b_i um. Dazu werden beide Gleichungen mit den Linearfaktoren der jeweils anderen multipliziert und dann wechselweise voneinander subtrahiert.

$$\begin{aligned} s_i a_0 - u_i b_0 &= s_i(u_i a_i + v_i b_i) - u_i(s_i a_i + t_i b_i) \\ &= (s_i v_i - u_i t_i) b_i \\ &= -\mathbf{D_i}\, b_i \end{aligned} \qquad \begin{aligned} t_i a_0 - v_i b_0 &= t_i(u_i a_i + v_i b_i) - v_i(s_i a_i + t_i b_i) \\ &= (u_i t_i - s_i v_i) a_i \\ &= \mathbf{D_i}\, a_i \end{aligned}$$

Schließt man den Fall aus, daß a_0 *und* b_0 gerade sind ($\mathbf{D_i} = 2^i$, vgl. Anmerkungen auf Seite 77), dann kann für den letzten Iterationsschritt $i = k$, in Abhängigkeit davon ob a_k oder b_k zuerst verschwindet, folgendermaßen konkretisiert werden:

$$\begin{aligned} &\text{Fall: } b_k = 0 \\ &s_k a_0 - u_k b_0 = 0 \\ &t_k a_0 - v_k b_0 = 2^k a_k \end{aligned} \qquad \begin{aligned} &\text{Fall: } a_k = 0 \\ &s_k a_0 - u_k b_0 = -2^k b_k \\ &t_k a_0 - v_k b_0 = 0\,. \end{aligned}$$

An diesem Punkt stellen wir jedoch fest, daß es sich bei den Größen u_k, v_k, s_k und t_k nicht um die Kofaktoren von $\gcd(a,b)$, sondern um eine Linearfaktordarstellung von $2^k \gcd(a,b)$ handelt.[99]

$$\begin{aligned} &\text{Fall: } b_k = 0 \\ &a_k = \gcd(a,b) = \alpha a_0 + \beta b_0 \\ &2^k \gcd(a,b) = t_k a_0 - v_k b_0 \end{aligned} \qquad \begin{aligned} &\text{Fall: } a_k = 0 \\ &b_k = \gcd(a,b) = \alpha a_0 + \beta b_0 \\ &2^k \gcd(a,b) = -s_k a_0 + u_k b_0 \end{aligned}$$

Um aus t_k, v_k und s_k, u_k die Bézout-Koeffizienten α und β zu bestimmen, sind zusätzliche Korrekturschritte nötig, welche in [Kal95] auch Korrekturphase (oder Phase II) genannt werden. Ziel ist es dabei, die rechte Seite der letzten Gleichung durch 2^k zu dividieren (oder in k Schritten wiederholt durch 2). Da die linke Seite immer eine gerade Zahl ist, können folgende Schlußfolgerungen im Falle $b_k = 0$ gezogen werden (falls $a_k = 0$ war, äquivalent für $t_k \to s_k$ und $v_k \to u_k$):[100]

- Sind v und t gerade, dann ist eine ganzzahlige Halbierung $v := v^{(g)}/2$, $t := t^{(g)}/2$ möglich (d. h. führt wieder zu einer ganzen Zahl).

[99] Für Anwendungen, die im weiteren eine Montgomery-Reduktion vornehmen, kann dies sogar von Vorteil sein [Kal95].

[100] Der Übersichtlichkeit halber wird auf die Indizierung jetzt verzichtet und statt dessen gerade Variablen/Terme mit (g) und ungerade mit (u) gekennzeichnet.

- In allen anderen Fällen (t bzw. v ungerade) kann man wegen

$$2^i \gcd(a,b) = ta - vb = (t+b)a - (v+a)b$$

die Reduktion $t := (t+b)/2$ bzw. $v := (v+a)/2$ vornehmen, ohne daß sich die Gleichung ändert. Man wandelt jedoch die ungerade Zahl t bzw. v in eine gerade Zahl, die dann (wie gewünscht) durch 2 teilbar ist. Die Ursache liegt einerseits darin begründet, daß die linke (und demzufolge auch rechte) Seite der Gleichung

$$2^i \gcd(a,b) = ta - vb$$

immer eine gerade Zahl repräsentiert und andererseits, entweder a oder/und b als ungerade vorausgesetzt wurden. Berücksichtigt man die folgenden Gesetzmäßigkeiten:

1. die Summe zweier ungerader oder zweier gerader Zahlen ist eine gerade Zahl;
2. die Summe einer ungeraden und einer geraden Zahl ist eine ungerade Zahl;
3. das Produkt zweier ungerader Zahlen ist eine ungerade Zahl;
4. alle anderen Kombinationen ergeben bei der Multiplikation eine gerade Zahl;

dann sind genau drei Fälle konstruierbar, in denen t oder v ungerade ist.

$$\begin{aligned}
\left[2^i \gcd(a,b)\right]^{(g)} &= \left[t^{(u)}a^{(u)}\right]^{(u)} - \left[v^{(u)}b^{(u)}\right]^{(u)} \\
\left[2^i \gcd(a,b)\right]^{(g)} &= \left[t^{(u)}a^{(g)}\right]^{(g)} - \left[v^{(g)}b^{(u)}\right]^{(g)} \\
\left[2^i \gcd(a,b)\right]^{(g)} &= \left[t^{(g)}a^{(u)}\right]^{(g)} - \left[v^{(u)}b^{(g)}\right]^{(g)}
\end{aligned}$$

In allen diesen Varianten führt aber die Addition $t+b$ bzw. $v+a$ zu einer geraden Zahl, womit sich das Korekturverfahren bestätigt.

Als Ergebnis kann man Algorithmus 5 formulieren, wobei außerdem folgende Anmerkungen berücksichtigt wurden:

1. Aus den Gleichungen 63 und 64 läßt sich im letzten Iterationsschritt (von Phase I)

$$\begin{aligned}
&\text{Fall: } b_k = 0 && \text{Fall: } a_k = 0 \\
&\quad a_0 = u_k a_k = u_k d && \quad a_0 = v_k b_k = v_k d \\
&\quad b_0 = s_k a_k = s_k d && \quad b_0 = t_k b_k = t_k d
\end{aligned}$$

schlußfolgern, d. h. bei u_k und s_k handelt es sich im Fall $b_k = 0$ (gleichermaßen für v_k, t_k im Fall $a_k = 0$) um die teilerfremden Faktoren

Algorithmus 5 Algorithmus $d = \gcd(a,b) = \alpha a + \beta b$ nach KALISKI

Require: a ungerade $\vee$ b ungerade

{Phase I}

$(a_0, u_0, s_0) \Leftarrow (a, 1, 0)$
$(b_0, v_0, t_0) \Leftarrow (b, 0, 1)$
$f \Leftarrow 0$ {Flag, daß anzeigt, ob schlußendlich α oder β negativ ist}
$k \Leftarrow 0$
while $a_k > 0$ **do**
  if $b_k > a_k$ **then**
    $(b_k, v_k, t_k) \Longleftrightarrow (a_k, u_k, s_k)$ {gewährleistet $a_k \geq b_k$}
    $f \Leftarrow \overline{f}$ {invertiere Flag}
  end if
  if a_k ungerade $\wedge$ b_k ungerade **then**
    $(a_k, v_k, t_k) \Longleftarrow (a_k - b_k, v_k + u_k, t_k + s_k)$ {a_k ist jetzt gerade, b_k weiterhin ungerade}
  end if
  if a_k gerade **then**
    $(a_{k+1}, u_{k+1}, s_{k+1}) \Longleftarrow (a_k/2, 2u_k, 2s_k)$ {a_k gerade, b_k ungerade}
  else
    $(b_{k+1}, v_{k+1}, t_{k+1}) \Longleftarrow (b_k/2, 2v_k, 2t_k)$ {a_k ungerade, b_k gerade}
  end if
  $k \Leftarrow k + 1$
end while
$\gcd(a,b) \Leftarrow b_k$ {Teilergebnis $d = \gcd(a,b)$}

{Phase II}

$i \Leftarrow k$
while $i > 0$ **do**
  if s_k ungerade $\vee$ u_k ungerade **then**
    $s_k \Longleftarrow s_k + t_k$ {Addition von $t_k = \overline{b}$ (vgl. Bemerkung 3)}
    $u_k \Longleftarrow u_k + v_k$ {Addition von $v_k = \overline{a}$ (vgl. Bemerkung 3)}
  end if
  $s_k \Longleftarrow s_k/2$ {Korrektur α}
  $u_k \Longleftarrow u_k/2$ {Korrektur β}
  $i \Leftarrow i - 1$
end while
$(\alpha, \beta) \Longleftarrow (s_k, u_k)$ {Wenn $f = 0$, dann $\alpha < 0$, sonst β}

Fall: $b_k = 0$	Fall: $a_k = 0$
$\overline{a}_0 = u_k \leq \gcd(a,b) \leq a_0$	$\overline{a}_0 = v_k \leq \gcd(a,b) \leq a_0$
$\overline{b}_0 = s_k \leq \gcd(a,b) \leq b_0$	$\overline{b}_0 = t_k \leq \gcd(a,b) \leq b_0$.

2. Wegen der stetigen Reduktion von a_i oder b_i müssen die Linearfaktoren u_i, s_i, v_i und t_i in Phase I schrittweise anwachsen, denn nur so können a_0 und b_0 nach Gleichung 63 und 64 konstant bleiben. In [Kal95, Theorem 1] wird bewiesen, daß keine dieser Größen während der Ausführung des Algorithmus' den Maximalwert $2a_0 - 1$ überschreitet (für $a_0 > b_0$).[101]

3. Die in Phase II vorzunehmende Addition von a bzw. b kann (entsprechend Anmerkung 5 auf Seite 79) ersetzt werden durch eine Addition von $\overline{a}_0$ bzw. $\overline{b}_0$.[102]

$$\begin{aligned} 2^i \gcd(a,b) &= (t+\overline{b})a - (v+\overline{a})b \\ &= ta + \overline{b}a - vb + \overline{a}b \\ &= ta + \overline{b}\overline{a}d - vb - \overline{a}\overline{b}d \\ &= ta - vb \end{aligned}$$

Algorithmus nach Penk [Knu98, Exercise 4.5.2.39], [MvV92, 14.4.3] Dieser Algorithmus stellt a_{i+1} und b_{i+1} als lineare Funktion von a_i und b_i dar, letztlich wird also von (a_0, b_0) auf (a_i, b_i) geschlossen.

$$a_i = u_i a_0 + v_i b_0 \qquad (u_0 = 1, v_0 = 0) \tag{65}$$

$$b_i = s_i a_0 + t_i b_0 \qquad (s_0 = 0,\ t_0 = 1) \tag{66}$$

Vorteilhaft wirkt sich aus, daß im letzten Reduktionsschritt (wenn a_k oder b_k verschwindet) u_k und v_k bzw. s_k und t_k direkt die gesuchten Kofaktoren α, β darstellen.

[101] Im Hinweis auf 74 wurde zwar für die kleinsten Werte der Bézout-Koeffizienten $|\alpha| < \overline{b}$ und $|\beta| < \overline{a}$ geschlußfolgert, was jedoch nicht für die Zwischenwerte eines bestimmten Algorithmus' gelten muß (hier anwendbar auf das Ende von Phase II). Theorem 1 in [Kal95] läßt sich aber auch nachvollziehen, wenn man mit Blick Algorithmus 5 die Anmerkung 2 (auf Seite 78) zur Anzahl der Iterationsschritte berücksichtigt.

[102] Und das nicht nur für den speziellen Fall $\gcd(a,b) = 1$, für welchen $a_0 = \overline{a}_0$, $b_0 = \overline{b}_0$ gilt.

Fall: $b_k = 0$	Fall: $a_k = 0$
$a_k = \gcd(a,b) = \alpha a_0 + \beta b_0$	$b_k = \gcd(a,b) = \alpha a_0 + \beta b_0$
$= u_k a_0 + v_k b_0$	$= s_k a_0 + t_k b_0$

Durch Einsetzen von a_{i+1} und b_{i+1} (entsprechend Tabelle 2) in die Gleichungen

$$a_{i+1} = u_{i+1} a_0 + v_{i+1} b_0$$
$$b_{i+1} = s_{i+1} a_0 + t_{i+1} b_0$$

kann man für den jeweiligen Reduktionsfall die zugehörige Transformationen der Linearfaktoren ableiten. Wieder am Beispiel $a_{i+1} = (a_i - b_i)/2$, $b_{i+1} = b_i$ soll das Vorgehen veranschaulicht werden (a_i und b_i sind ungerade, $a_i \geq b_i$).

$$a_{i+1} = \frac{a_i - b_i}{2} = u_{i+1} a_0 + v_{i+1} b_0 \qquad b_{i+1} = b_i = s_{i+1} a_0 + t_{i+1} b_0$$
$$(u_i - s_i) a_0 + (v_i - t_i) b_0 = 2u_{i+1} a_0 + 2v_{i+1} b_0 \qquad s_i a_0 + t_i b_0 = s_{i+1} a_0 + t_{i+1} b_0$$

Vergleich von linker und rechter Seite läßt den Schluß zu:

$$u_{i+1} = \frac{u_i - s_i}{2} \qquad s_{i+1} = s_i$$
$$v_{i+1} = \frac{v_i - t_i}{2} \qquad t_{i+1} = t_i \; .$$

Die anderen Kombinationen können genau nach demselben Schema abgeleitet werden. Tabelle 4 faßt die Ergebnisse in übersichtlicher Form zusammen.[103]

Dabei tritt allerdings wieder das bekannte Problem auf: Wie kann man die Ganzzahligkeit des jeweiligen Linearfaktoren bei der Halbierung wahren? Die Antwort haben wir schon beim vorangegangenen Algorithmus geliefert – indem die Ausgangsgleichungen 65 und 66 folgendermaßen erweitert:

[103] Der Fall, daß a_i und b_i gerade sind, kann ausgeschlossen werden, wenn dies auch für a_0 und b_0 vorausgesetzt wird (was man ohne Einschränkung kann, vgl. Anmerkungen zum binären GCD-Algorithmus auf Seite 77).

Tabelle 4: Linearfaktoren beim Algorithmus nach Penk

a_i	b_i	$a_{i+1} =$	$b_{i+1} =$	$u_{i+1} =$	$v_{i+1} =$	$s_{i+1} =$	$t_{i+1} =$
gerade	ungerade	$\frac{a_i}{2}$	b_i	$\frac{u_i}{2}$	$\frac{v_i}{2}$	s_i	t_i
ungerade	gerade	a_i	$\frac{b_i}{2}$	u_i	v_i	$\frac{s_i}{2}$	$\frac{t_i}{2}$
ungerade, $a_i \geq b_i$		$\frac{a_i - b_i}{2}$	b_i	$\frac{u_i - s_i}{2}$	$\frac{v_i - t_i}{2}$	s_i	t_i
ungerade, $b_i \geq a_i$		a_i	$\frac{b_i - a_i}{2}$	u_i	v_i	$\frac{s_i - u_i}{2}$	$\frac{t_i - v_i}{2}$

$$a_i = (u_i \pm b_0)a_0 + (v_i \mp a_0)b_0$$
$$b_i = (s_i \pm b_0)a_0 + (t_i \mp a_0)b_0$$

und dadurch ungerade Zahlen u_i, v_i bzw. s_i, t_i in gerade umwandelt.

Beschränken wir uns auf die Kombinationen nach Tabelle 4, in denen u_i und v_i verändert werden (äquivalent für $u_i \to s_i$ und $v_i \to t_i$). Für den Fall, daß a_i gerade und b_i ungerade ist (zweite Zeile in 4), können wir auf die Argumentation von Seite 81 zurückgreifen (Phase II der Methode nach Kaliski). Sie erlaubt uns die Subtraktion/Addition von a_0 und b_0 für den Fall, daß u_i oder v_i ungerade ist. Die Situation, daß a_i und b_i ungerade sind (vorletzte Zeile in 4), kann man durch gedankliche Verzögerung der Halbierung in den nächsten Iterationsschritt erklären. Wählt man als modifizierten Einzelschritt $a_{i+1} = a_i - b_i$ (und entsprechend $u_{i+1} = u_i - s_i$, $v_{i+1} = v_i - t_i$), so reduziert sich die Fragestellung wieder auf eine gerade Zahl $a_{i+1} = u_{i+1}a_0 + v_{i+1}b_0$, also auf den vorangegangenen Fall.

Als Ergebnis der Ausführungen kann man Algorithmus 6 formulieren. Der Vorteil des Algorithmus' (gegenüber Kaliski's) liegt vor allem darin, daß keine Korrekturphase nötig ist. Nachteilig für eine praktische Umsetzung ist die notwendige Vorzeichen-Arithmetik.

D.2. Lineare diophantische Gleichungen

Gleichungen mit ausschließlich ganzzahligen Lösungen $x, y, \ldots \in \mathbb{Z}$ nennt man diophantisch, wobei sie im linearen Fall (für zwei Variablen) die folgende Form haben:

$$ax + by = c, \qquad \text{mit } a, b, c \in \mathbb{N}\,. \tag{67}$$

Solche Gleichungen haben genau dann eine Lösung, wenn der größte gemeinsame Teiler $d = \gcd(a, b)$ den Wert c teilt. Im Zusammenhang mit Euklid's Algorithmus wurde dies praktisch

Algorithmus 6 Algorithmus $d = \gcd(a,b) = \alpha a + \beta b$ nach PENK

Require: b ungerade
  $(a_0, u_0, v_0) \Leftarrow (a, 1, 0)$
  $(b_0, s_0, t_0) \Leftarrow (b, 0, 1)$
  $i \Leftarrow 0$
  while $a_i > 0$ **do**
    if $b_i > a_i$ **then**
      $(b_i, s_i, t_i) \Longleftrightarrow (a_i, u_i, v_i)$ {gewährleistet $a_i \geq b_i$}
    end if
    if a_i ungerade $\wedge$ b_i ungerade **then**
      $(a_i, u_i, v_i) \Longleftarrow (a_i - b_i, u_i - s_i, v_i - t_i)$ {a_i ist jetzt gerade, b_i weiterhin ungerade}
    end if
    if a_i gerade **then**
      $a_{i+1} \Longleftarrow a_i/2$ {a_i gerade, b_i ungerade}
      if u_i gerade $\wedge$ v_i gerade **then**
        $u_{i+1} \Longleftarrow u_i/2$
        $v_{i+1} \Longleftarrow v_i/2$
      else
        $u_{i+1} \Longleftarrow (u_i + b)/2$
        $v_{i+1} \Longleftarrow (v_i - a)/2$
      end if
    else
      $b_{i+1} \Longleftarrow b_i/2$ {a_i ungerade, b_i gerade}
      if s_i gerade $\wedge$ t_i gerade **then**
        $s_{i+1} \Longleftarrow s_i/2$
        $t_{i+1} \Longleftarrow t_i/2$
      else
        $s_{i+1} \Longleftarrow (s_i + b)/2$
        $t_{i+1} \Longleftarrow (t_i - a)/2$
      end if
    end if
    $i \Leftarrow i + 1$
  end while
  $(d, \alpha, \beta) \Leftarrow (b_i, s_i, t_i)$ {$d = \gcd(a,b) = \alpha a + \beta b$}

schon nachgewiesen – auch daß es unendlich viele solcher Lösungen gibt, kam in Abschnitt D.1.1 zur Sprache.

Zuerst bemerken wir, daß die Differenz zweier Lösungen (x_1,y_1) und (x_2,y_2) die homogene Gleichung $ax+by=0$ erfüllt.

$$(ax_1+by_1)-(ax_2+by_2)=a\underbrace{(x_1-x_2)}_{x}+b\underbrace{(y_1-y_2)}_{y}=0$$

Mit $a=d\overline{a}$ und $b=d\overline{b}$ läßt sich sogar schreiben

$$\overline{a}x+\overline{b}y=0$$

und es liegen die Lösungen $(x,y)=(n\overline{b},-n\overline{a})$, $n\in\mathbb{Z}$ auf der Hand (Einsetzen ergibt $\overline{a}x+\overline{b}y=\overline{a}n\overline{b}-\overline{b}n\overline{a}=0$).[104] Findet man jetzt noch eine partikuläre Lösung (x_2,y_2), dann ergeben sich alle weiteren zu:

$$x_1=x_2+x=x_2+n\overline{b}\qquad\qquad y_1=y_2+y=y_2-n\overline{a}\,.$$

Partikuläre Lösungen (x_2,y_2) für $ax_2+by_2=c$ haben wir aber schon mittels der erweiterten GCD-Algorithmen zur Verfügung, denn mit der Bézout's Identität (den Index 2 jetzt weggelassen)

$$\begin{aligned}\gcd(a,b)=d&=\alpha a+\beta b\\ d\overline{c}&=\alpha\overline{c}a+\beta\overline{c}b\end{aligned}$$

gilt:

$$ax+by=c=d\overline{c}=\alpha\overline{c}a+\beta\overline{c}b\,.$$

Eine partikuläre Lösung ($x_2\mathrel{\hat{=}}x$, $y_2\mathrel{\hat{=}}y$) kann deshalb mit Hilfe der Kofaktoren α,β gegeben werden.

$$x_2=\alpha\overline{c}=\alpha\frac{c}{d}\qquad\qquad y_2=\beta\overline{c}=\beta\frac{c}{d}$$

Die Vielfalt aller Lösungen stellt sich dadurch wiefolgt dar:

[104] Eine Untermenge der Lösungen stellt für $n=ld$, $l\in\mathbb{Z}$ natürlich $(x,y)=(lb,-la)$.

$$x_n = \alpha\frac{c}{d} + n\frac{b}{d} \qquad\qquad y_n = \beta\frac{c}{d} - n\frac{a}{d}\,. \tag{68}$$

Für den Spezialfall $d = \gcd(a,b) = 1$ entartet die Formel zu:

$$x_n = \alpha c + nb \qquad\qquad y_n = \beta c - na\,. \tag{69}$$

D.3. Chinesischer Restsatz

D.3.1. Hilfssatz für zwei Kongruenzen

Um sich dem Chinesischen Restsatz[105] zu nähern, betrachten wir zunächst einen etwas einfacheren Fall, der die grundsätzliche Fragestellung jedoch beinhaltet: Welche natürliche Zahl z erfüllt die folgenden beiden Kongruenzen:

$$z \equiv a \pmod{n} \qquad\qquad z \equiv b \pmod{m},$$

wenn vorausgesetzt wird, daß n und m relativ prim zueinander sind?

Um sie zu beantworten formulieren wir den Ausgangspunkt zuersteinmal entsprechend Restklassenbeziehung 48:

$$z = a + xn = b + ym, \qquad x,y \in \mathbb{N}\,. \tag{70}$$

Deren Darstellung als

$$c = b - a = xn - ym \tag{71}$$

zeigt mit Verweis auf die Form von 67, daß es sich um eine lineare diophantische Gleichung handelt. Wegen $\gcd(n,m) = 1$ könnte die zugehörige Lösungsformel 69 zwar sofort zur Anwendung kommen – naheliegend (da kurz) ist jedoch auch die Anwendung von Bézout's Identität. Denn multipliziert man $\gcd(n,m) = \alpha n + \beta m = 1$ mit c, dann kann aus

$$c = c(\alpha n + \beta m) = \alpha nc + \beta mc$$

[105] CRT – Chinese Remainder Theorem

durch Vergleich mit 71 sofort abgelesen werden, daß $x = \alpha c$ und $y = -\beta c$ gelten muß.[106] Mit der Erkenntnis aus Formel 69, daß es sich bei den Lösungen einer linearen diophantischen Gleichungen immer um eine ganze Lösungsmenge handelt, resultiert:[107]

$$x_k = \alpha c + km \qquad\qquad y_k = -\beta c - kn, \qquad k \in \mathbb{Z}\,.$$

Durch Einsetzen in Formel 70 erhält man

$$\begin{aligned} z_k &= a + x_k n & z_k &= b + y_k m \\ &= a + (\alpha c - km)n & &= b - (\beta c + kn)m \\ &= a \underbrace{(1 - \alpha n)}_{\beta m} + \alpha bn - kmn & &= b \underbrace{(1 - \beta m)}_{\alpha n} + \beta am - kmn \end{aligned} \tag{72}$$

und so eine geschlossene Darstellung für die Lösungsmenge.

$$z_k = \alpha bn + \beta am + kmn \tag{73}$$

Mit der von den Restklassen bekannten Kongruenz 49 für teilerfremde Zahlen:[108]

$$1 \equiv \beta m \pmod{n} \qquad\qquad 1 \equiv \alpha n \pmod{m}$$

können wir die Probe machen.

$$\begin{aligned} z_k \bmod n &= \underbrace{(\alpha b + km)n}_{0} + \beta am \bmod n & z_k \bmod m &= \alpha bn + \underbrace{(\beta a + kn)m}_{0} \bmod m \\ &= \beta am \bmod n = a & &= \alpha bn \bmod m = b \end{aligned}$$

Die Lösungsmenge z_k bildet demzufolge eine Restklasse $[z]_{mn}$.

[106] H. L. Garner hat genau diesen Lösungsansatz für eine unbeschränkte Anzahl von Kongruenzen ausgebaut [Gar59].
[107] Die eigentliche Ursache für die Vielfalt von Lösungen ist im Hinweis zum euklidischen Algorithmus auf Seite 74 begründet.
[108] Hierbei wird ganz deutlich, daß es sich bei α und β um Inverse handelt: $\alpha \equiv n^{-1} \pmod{m}$, $\beta \equiv m^{-1} \pmod{n}$.

$$z \equiv \alpha bn + \beta am \pmod{mn} \tag{74}$$

D.3.2. Ein System von Kongruenzen

Nehmen wir jetzt ein ganzes System von Kongruenzen an:

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_n \pmod{m_n}, \end{aligned}$$

wobei $\gcd(m_i, m_k) = 1$ für $i \neq k$ gelten soll. Wie schon im Fall von zwei Kongruenzen stellt man die Frage, welche Lösung x kongruent zu allen a_i modulo m_i ist ($i = 1, \ldots, n$). Ohne einen exakten mathematischen Beweis anzutreten, scheint als Schlußfolgerung aus Abschnitt D.3.1 einleuchtend, daß die Lösung(en) x eine Restklasse modulo $m = \prod_n m_i$ darstellen [Knu98, CP05].[109]

Bezeichnen wir mit $\overline{m}_i$ das Produkt aller Moduli ausgeschlossen m_i, also

$$\overline{m}_i = \frac{m}{m_i} = \frac{\prod_{j=1}^{n} m_j}{m_i} = \prod_{\substack{j=1 \\ j \neq i}}^{n} m_j,$$

dann gilt wegen der Teilerfremdheit der einzelnen Moduli $\gcd(\overline{m}_i, m_i) = 1$ bzw. mit dem Satz von Bézout:[110]

$$\begin{aligned} \alpha_i \overline{m}_i + \beta_i m_i &= 1 \\ \alpha_i \overline{m}_i \bmod m_i &= 1\,. \end{aligned}$$

Im Gegensatz dazu verschwindet $\alpha_k \overline{m}_k \bmod m_i$ für $i \neq k$, denn m_i ist als Faktor in $\overline{m}_k$ enthalten. Die Orthogonalität beider Fälle kann mit Hilfe des Kronecker-Symbols δ_{ik} ausgedrückt werden:

[109] Da jedes $x - a_i$ ein Vielfaches des zugehörigen m_i sein muß, nach Voraussetzung aber alle m_i relativ prim zueinander sind, ist das kleinste gemeinsame Vielfache von $m_0, m_1, \ldots, m_n$ genau das Produkt $m = \prod_n m_i$.

[110] Der Bézout-Koeffizient α_i kann wieder als Inverses aufgefasst werden: $\alpha_i \equiv \overline{m}_i^{-1} \pmod{m_i}$.

$$\alpha_k \overline{m}_k \bmod m_i = \delta_{ik} = \begin{cases} 0 & (i \neq k) \\ 1 & (i = k) \end{cases}.$$

Die endgültige Lösungsidee besteht nun darin, für jede Kongruenz i den folgenden Ausdruck zu bilden:

$$\sum_{k=1}^{n} \alpha_k a_k \overline{m}_k \bmod m_i = \sum_{k=1}^{n} a_k \delta_{ik} \bmod m_i = a_i \bmod m_i \equiv x \pmod{m_i}$$

Wie zu sehen, ist die Summe auf der linken Seite kongruent zu a_i modulo m_i, was

$$x \equiv \sum_{k=1}^{n} \alpha_k a_k \overline{m}_k \pmod{m} \tag{75}$$

als finales Ergebnis rechtfertigt.

Für den einfachen Fall $n = 2$ von Abschnitt D.3.1 kann man mit $\overline{m}_1 = m_2$, $\overline{m}_2 = m_1$ sowie

$$\begin{aligned} 1 &= \alpha_1 \overline{m}_1 + \beta_1 m_1 \\ &= \alpha_1 m_2 + \beta_1 m_1 \end{aligned} \qquad \begin{aligned} 1 &= \alpha_2 \overline{m}_2 + \beta_2 m_2 \\ &= \alpha_2 m_1 + \beta_2 m_2 \end{aligned}$$

$$\alpha_1 = \beta_2 \qquad \alpha_2 = \beta_1$$

relativ einfach Kongruenz 74 verifizieren.[111]

$$x \equiv \alpha_1 a_1 \overline{m}_1 + \alpha_2 a_2 \overline{m}_2 = \alpha_1 a_1 m_2 + \alpha_2 a_2 m_1 = \alpha_1 a_1 m_2 + \beta_1 a_2 m_1 \pmod{m_1 m_2}$$

D.4. Montgomery-Potenzierung

Bei der Methode nach Montgomery handelt es sich eigentlich um eine Multiplikationsmethode, bei der die Modulo-Reduktion des (Zwischen-) Ergebnisses ohne echte Langzahldivision erfolgen kann [Mon85]. Da allerdings zuerst eine Transformation beider Faktoren in den Montgomery-"Raum" vorgenommen werden muß (welche zwei Modulo-Division erfordert) und außerdem

[111] Ein echtes Rechenbeispiel für $n = 3$ kann man z. B. in [CP05, 2.1.3] finden.

einmal der erweiterte GCD-Algorithmus bemüht werden muß, kommen die Geschwindigkeitsvorteile nur bei der Potenzierung wirklich zum tragen.

Um einen leicht verständlichen Zugang zu finden, konzentrieren wir uns auf eine einzelne Multiplikation $c = ab \pmod m$. Dazu soll eine Zahl r vorausgesetzt werden, die keinen gemeinsamen Teiler mit dem Modul m hat und für die $r > m$ gewährleistet ist. Die MONTGOMERY-Multiplikation kann man nun, unter der Voraussetzung $\gcd(r,m) = 1$ (bzw. mit dem Satz von BÉZOUT $\alpha r - \beta m = 1$), in folgenden Einzelschritten darstellen:

1. Berechnung der BÉZOUT-Koeffizienten α, β mit Hilfe des erweiterten euklidischen Algorithmus;
2. Eingangstransformation der Faktoren a und b zu $\hat{a} = ar \bmod m$ und $\hat{b} = br \bmod m$ (eine normale Modulo-Division);
3. (Wiederholte) Multiplikation im MONTGOMERY-Bereich:
   a) Berechnung des Produkts $x = \hat{a} \cdot \hat{b}$ (man beachte, daß hierbei keine Modulo-Reduktion vorgenommen wird);
   b) MONTGOMERY-Reduktion von $x = (ar \bmod m) \cdot (br \bmod m)$ zu $\hat{c} = abr \bmod m = cr \bmod m$;

      Die nötige Korrektur[112] $\hat{c} = \mathrm{M}(x,r,m) = xr^{-1} \bmod m$ führt zu einer MONTGOMERY-Darstellung für c (als Voraussetzung für den nächsten Teilschritt).[113]
   c) Eine weitere (optionale) Multiplikationen im MONTGOMERY-"Raum", die $\hat{c}$ als einen der Faktoren verwendet;
4. Rücktransformation des Ergebnisses $\hat{c}$ zu $c = \hat{c}r^{-1} \bmod m$, ebenfalls eine MONTGOMERY-Reduktion: $c = \mathrm{M}(\hat{c},r,m)$.

MONTGOMERY-Reduktion Zur Herleitung von MONTGOMERY's effizienter Berechnungsmethode für $\mathrm{M}(x,r,m) = xr^{-1} \bmod m$ wählen wir als Ausgangspunkt:

$$\begin{aligned} m(\beta x \bmod r) &= m(\beta x - ur), \quad \text{mit } u = \lfloor \beta x / r \rfloor \\ r(\alpha x \bmod m) &= r(\alpha x - vm), \quad \text{mit } v = \lfloor \alpha x / m \rfloor . \end{aligned}$$

Subtraktion beider Gleichungen ergibt

$$\begin{aligned} m(\beta x \bmod r) - r(\alpha x \bmod m) &= m(\beta x - ur) - r(\alpha x - vm) \\ &= rm(v-u) - x(\alpha r - \beta m), \end{aligned}$$

[112] Der Ausdruck $xr^{-1} \bmod m = \hat{a}\hat{b}r^{-1} \bmod m$ wird auch als MONTGOMERY-Produkt von $\hat{a}$ und $\hat{b}$ bezeichnet.

[113] Im Gegensatz zur Multiplikation, für welche $\hat{a} \cdot \hat{b} \neq \hat{c} \pmod m$ gilt, verhält sich die Addition regulär: $\hat{a} + \hat{b} = ar + br = (a+b)r$.

was mit $\gcd(r,m) = \alpha r - \beta m = 1$ zu

$$m(\beta x \bmod r) + x = (v-u)rm + r(\alpha x \bmod m)$$

führt.[114]

Mit dem Wissen, daß es sich bei α um das multiplikativ inverse Element von r im Restklassensystem modulo m handelt ($\alpha = r^{-1} \pmod m$, $\beta = -m^{-1} \pmod r$), stellen wir noch um:

$$\alpha x \bmod m = \frac{m(\beta x \bmod r) - (v-u)rm + x}{r}$$

und gewinnen letztlich Montgomery's Reduktionsformel.

$$\mathrm{M}(x,r,m) = xr^{-1} \bmod m = \frac{m(\beta x \bmod r) + x}{r} - (v-u)m \tag{76}$$

Praktisch benötigt man u und v nicht, sondern geht meist nach folgendem Algorithmus vor:

Algorithmus 7 Montgomery-Reduktion $y = xr^{-1} \bmod m$

$t \Leftarrow \beta x \bmod r$
$y \Leftarrow \dfrac{mt + x}{r}$
if $y \geq m$ **then**
  $y \Leftarrow y - m$
end if

Bei geschickter Wahl von r zum Beispiel als $r = 2^s$ sind für alle (Modulo-) Divisionen in Formel 76 nur logische oder Schiebeoperationen nötig.[115] Um $\gcd(2^s, m) = 1$ zu garantieren ist die einfachste Bedingung die, m als ungerade vorauszusetzen.[116]

[114] Wegen des Vorkommens von r in allen Summanden der rechten Seite muß die linke Seite durch r teilbar sein, d. h. $m(\beta x \bmod r) + x$ ist ein Vielfaches von r.
[115] Effiziente numerische Algorithmen zur Montgomery-Reduktion findet man z. B. in [Koç94b, KAK96, DK91].
[116] Oftmals kann eine solche Einschränkung hingenommen werden, falls nicht siehe z. B. [Koç94a].

Literatur

[AF99] *ATM Security Specification Version 1.0*. af-sec-0100.000, ATM Forum Technical Committee, Februar 1999.

[ANS98] *Triple Data Encryption Algorithm Modes of Operation*. American National Standard for Financial Services X9.52, ANSI, 1998.

[ANS05] *Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*. American National Standard for Financial Services X9.62, ANSI, September 2005.

[AR78] Adleman, Leonard M. und Ronald L. Rivest: *The use of public key cryptography in communication system design*. IEEE Commun. Mag., 16(6):20–23, November 1978.

[BB87] Bojanczyk, A. und R. P. Brent: *A systolic algorithm for extended GCD computation*. Computers & Mathematics with Applications, 14(4):233–238, 1987.

[Ber84] Berlekamp, Elwyn R.: *Algebraic Coding Theory, Revised 1984 Edition*. Aegean Park Press, Laguna Hills, CA, 1984.

[BK83] Brent, R. P. und H. T. Kung: *Systolic VLSI arrays for linear time GCD computation*. In: Anceau, F. und E. J. Aas (Herausgeber): *Proceedings of International Conference on Very Large Scale Integration (VLSI 83)*, Seiten 145–154. International Federation of Information Processing, Elsevier Science Publishers B.V., 1983.

[Ble98] Bleichenbacher, Daniel: *Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1*. In: *Advances in Cryptology — CRYPTO '98*, Band 1462 der Reihe *Lecture Notes in Computer Science*, Seiten 629–660. Springer, August 1998.

[Bos96] Bosch, Siegfried: *Algebra*. Springer, Berlin, 1996.

[BR95] Bellare, Mihir und Phillip Rogaway: *Optimal Asymmetric Encryption – How to Encrypt with RSA*. In: De Santis, Alfredo (Herausgeber): *Advances in Cryptology — EUROCRYPT '94: Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9 - 12, 1994. Proceedings*, Band 950 der Reihe *Lecture Notes in Computer Science*, Seiten 92–111, Berlin, August 1995. Springer-Verlag.

[BR96] Bellare, Mihir und Phillip Rogaway: *The Exact Security of Digital Signatures – How to Sign with RSA and Rabin*. In: Maurer, U. (Herausgeber): *Advances in Cryptology — EUROCRYPT '96*, Band 1070 der Reihe *Lecture Notes in Computer Science*, Seiten 399–416, Berlin Heidelberg, 1996. Springer-Verlag.

[BR03] Black, John und Phillip Rogaway: *A Suggestion for Handling Arbitrary-Length Messages with the CBC MAC*. Comments to NIST concerning AES Modes of Operations, Mai 2003.

[Bun08] Bundschuh, Peter: *Einführung in die Zahlentheorie*. Springer, Berlin Heidelberg, 6. Auflage, 2008.

[CP05] Crandall, Richard und Carl Pomerance: *Prime Numbers. A Computational Perspective*. Springer, 2. Auflage, 2005.

[Dam90] Damgård, Ivan Bjerre: *A Design Principle for Hash Functions*. In: Brassard, G. (Herausgeber): *Advances in Cryptology — CRYPTO '89*, Band 435 der Reihe *Lecture Notes in Computer Science*, Seiten 416–427, Berlin / Heidelberg, 1990. Springer-Verlag.

[DK91] Dussé, Stephen R. und Burton S. Kaliski, Jr.: *A Cryptographic Library for the Motorola DSP56000*. In: *Advances in Cryptology — EUROCRYPT '90*, Band 473 der Reihe *Lecture Notes in Computer Sciences*, Seiten 230–244, New York, 1991. Springer.

[Dwo01] Dworkin, Morris: *Recommendation for Block Cipher Modes of Operation*. NIST Special Publication 800-38A, NIST, 2001.

[Dwo07] Dworkin, Morris: *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. NIST Special Publication 800-38D, NIST, 2007. DRAFT (June, 2007).

[EBS09] Elaine Barker, Lily Chen, Andrew Regenscheid und Miles Smid: *Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography*. NIST Special Publication 800-56B, NIST, August 2009.

[FH03] Frankel, S. und H. Herbert: *The AES-XCBC-MAC-96 Algorithm and Its Use with IPsec*. RFC 3566, IETF, September 2003.

[FIP01] *Advanced Encryption Standard (AES)*. FIPS Publication 197, National Institute of Standards and Technology, 2001.

[Gar59] Garner, Harvey L.: *The Residue Number System*. IRE Transactions on Electronic Computers, 8(2):140–147, Juni 1959.

[Gro00] Grossschädl, Johann: *The Chinese Remainder Theorem and its application in a high-speed RSA crypto chip*. In: *ACSAC '00: Proceedings of the 16th Annual Computer Security Applications Conference*, Seiten 384–393, Washington, DC, Dezember 2000. IEEE Computer Society.

[Har06] Hars, Laszlo: *Modular Inverse Algorithms Without Multiplications for Cryptographic Applications*. EURASIP Journal on Embedded Systems, 2006.

[HFS75] Horst Feistel, William A. Notz und J. Lynn Smith: *Some cryptographic techniques for machine-to-machine data communications*. Proc. IEEE, 63(11):1545–1554, November 1975.

[HK97] H. Krawczyk, M. Bellare, R. Canetti: *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104, IETF, Februar 1997.

[IEE00] *Standard Specifications For Public-Key Cryptography*. Std 1363-2000, IEEE, 2000.

[ISO99] *Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher*. International Standard ISO/IEC 9797-1, ISO/IEC, 1999.

[ISO02a] *Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Integer factorization based mechanisms*. International Standard ISO/IEC 9796-2, ISO/IEC, 2002.

[ISO02b] *Information technology – Security techniques – Message Authentication Codes (MACs) – Part 2: Mechanisms using a dedicated hash-function*. International Standard ISO/IEC 9797-2, ISO/IEC, 2002.

[ISO04] *Information technology – Security techniques – Hash functions – Part 3: Dedicated hash-functions*. International Standard ISO/IEC 10118-3, ISO/IEC, Februar 2004.

[ISO05] *Information technology – Security techniques – Encryption algorithms – Part 3: Block ciphers*. International Standard ISO/IEC 18033-3, ISO/IEC, 2005.

[ISO06a] *Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers*. International Standard ISO/IEC 18033-2, ISO/IEC, 2006.

[ISO06b] *Information technology – Security techniques – Modes of operation for an n-bit block cipher algorithm*. International Standard ISO/IEC 10116, ISO/IEC, 2006.

[ITU97] *Information technology - Open Systems Interconnection - The Directory: Authentication framework*. Recommendation X.509, ITU-T, 1997.

[Jeb93] Jebelean, Tudor: *Systolic Normalization of Rational Numbers*. In: Dadda, L. und B. Wah (Herausgeber): *ASAP'93 — International Conference on Application Specific Array Processors*, Seiten 502–513, Venice, Italy, Oktober 1993. IEEE Computer Society Press. Also: Technical Report 93-45, RISC-Linz, Johannes Kepler University, Linz, Austria, August 1993.

[JK03] Jonsson, J. und B. Kaliski: *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*. RFC 3447, IETF, Februar 2003.

[KAK96] Koç, Çetin Kaya, Tolga Acar und Burton S. Kaliski, Jr.: *Analyzing and Comparing Montgomery Multiplication Algorithms*. IEEE Micro, 16(3):26–33, Juni 1996.

[Kal95] Kaliski, Jr., Burton S.: *The Montgomery Inverse and Its Applications*. IEEE Transactions on Computers, 44(8):1064–1065, August 1995.

[Ken93] Kent, S.: *Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers*. RFC 1423, IETF, Februar 1993.

[Knu98] Knuth, Donald E.: *Seminumerical Algorithms*, Band 2 der Reihe *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, 3. Auflage, 1998.

[Koç94a] Koç, Ç. K.: *Montgomery Reduction with Even Modulus*. IEE Proceedings on Computers and Digital Techniques, 141(5):314–316, September 1994.

[Koç94b] Koç, Çetin Kaya: *High-Speed RSA Implementation*. Technischer Bericht TR-201, RSA Laboratories, 1994. Version 2.0.

[Kör88] Körner, T.W.: *Fourier Analysis*. Cambridge University Press, 1988.

[LRW00] Lipmaa, Helger, Phillip Rogaway und David Wagner: *CTR-Mode Encryption*. Comments to NIST concerning AES Modes of Operations, Mai 2000.

[Mer90] MERKLE, RALPH C.: *One Way Hash Functions and DES*. In: BRASSARD, G. (Herausgeber): *Advances in Cryptology — CRYPTO '89*, Band 435 der Reihe *Lecture Notes in Computer Science*, Seiten 428–446, Berlin / Heidelberg, 1990. Springer-Verlag.

[Mon85] MONTGOMERY, P.: *Modular multiplication without trial division*. Mathematics of Computation, 44(170):519–521, 1985.

[MvV92] MENEZES, ALFRED J., P. C. VAN OORSCHOT und SCOTT A. VANSTONE: *Handbook of applied cryptography*. CRC Press Series on Discrete Mathematics and Its Application. CRC Press, 2. Auflage, 1992.

[NBS80] *DES Modes of Operation*. FIPS Publication 81, National Bureau of Standards, 1980.

[NIS93] *Secure Hash Standard*. FIPS Publication 180-1, National Institute of Standards and Technology, April 1993. (Supersedes FIPS PUB 180 - 1993 May 11).

[NIS94] *Digitale Signature Standard*. FIPS Publication 186, National Institute of Standards and Technology, 1994.

[NIS07] *The Keyed-Hash Message Authentication Code (HMAC)*. FIPS Publication 198-1, National Institute of Standards and Technology, 2007.

[NIS08] *Secure Hash Standard (SHS)*. FIPS Publication 180-3, National Institute of Standards and Technology, Oktober 2008.

[NIS09] *Digitale Signature Standard (DSS)*. FIPS Publication 186-3, National Institute of Standards and Technology, 2009.

[Oka90] OKAMOTO, T.: *A Fast Signature Scheme Based on Congruential Polynomial Operations*. IEEE Trans. Inform. Theory, IT-36(1):47–53, Januar 1990.

[PD75] POLLARD, HARRY und HAROLD G. DIAMOND: *The Theory of Algebraic Numbers*. The Mathmatical Association of America, 2. Auflage, 1975.

[PKC93] *RSA Encryption Standard*. PKCS #1 v1.5, RSA Laboratories, November 1993.

[PKC02] *RSA Cryptography Standard*. PKCS #1 v2.1, RSA Laboratories, Juni 2002.

[PW72] PETERSON, W. WESLEY und E. J. WELDON, JR.: *Error-Correcting Codes*. The MIT Press, Cambridge Massachusetts and London, England, 2. Auflage, 1972.

[Riv92] RIVEST, R.: *The MD5 Message-Digest Algorithm*. RFC 1321, IETF, April 1992.

[Ros99] ROSING, MICHAEL: *Implementing Elliptic Curve Cryptography*. Manning Publications, 1999.

[RSA78] RIVEST, R. L., A. SHAMIR und L. M. ADLEMAN: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Communications of the ACM, 21(2):120–126, Februar 1978.

[Rul97] RULAND, C.: *Breitbandverschlüsselung (155 Mbit/s) mit Selbstsynchronisation*. In: *Tagungsband 5. Deutscher IT-Sicherheitskongreß des BSI*, Seiten 289–301. Bundesamt für Sicherheit in der Informationstechnik - BSI, 1997.

[Sch96] Schneier, Bruce: *Applied Cryptography*. John Wiley & Sons, 2. Auflage, 1996.

[Sha49] Shannon, C. E.: *Communication Theory of Secrecy Systems*. Bell Systems Technical Journal, 28:656–715, 1949.

[Sho05] Shoup, Victor: *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005.

[Ste67] Stein, J.: *Computational problems associated with Racah algebra*. Journal of Computational Physics, 1(3):397–405, 1967.

[Wel01] Welschenbach, Michael: *Kryptographie in C und C++*. Springer, Berlin Heidelberg New York, 2. Auflage, 2001.

Index