Kryptologie und Protokollverifikation

Zusammenfassung

Art der Arbeit

im Fachgebiet Fachgebiet

vorgelegt von: Christopher Syben

Studienbereich: Studienbereich

Abstract

Das letzte Kapitel Protokollverifikation ist nicht mehr wirklich enthalten, nur noch ein bisschen pi-Kalkül. Ansonsten gilt: Kritisch Lesen und mit denken es könnten sich fehler eingeschlichen haben oder etwas ungenau sein :). Ich hab versucht, die Themen etwas zu gliedern. Daher befinden sich mathematische oder komplexitätstheoretische Grundlagen die er als Einschub in den Themen hatte im 7. Kapitel. Also wenn mal etwas unklar erscheint oder man das gefühl hat hier fehlt doch etwas, lohnt immer ein Blick in das Mathe-Kapitel.

Inhaltsverzeichnis

Abkürzungsverzeichnis

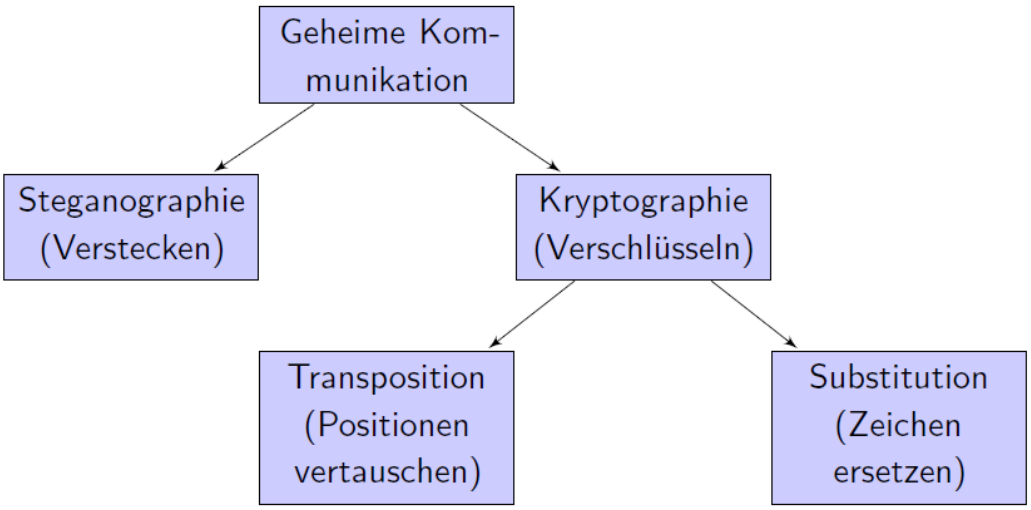
Abk. Abkürzung

Abbildungsverzeichnis

Tabellenverzeichnis

1. Übersicht

Die Verfahren mit denen Informationen "geheim"übertragen werden können lassen sich wie folgt gliedern (wobei Transposition und Substitution auch gemischt werden können):



1.1. Steganographie

Die Idee der Steganographie ist Nachrichten in scheinbar belanglosen Informationen zu verstecken, nach Vereinbarung eines Verfahrens zu ihrer Extraktion.
Als Beispiel: Ein Wort mit gerade Anzahl an Buchstaben sei 0 und ein Wort mit ungerader Anzahl an Buchstaben sei 1. Dann lässt sich ein Text mit sinnfreiem Inhalt schreiben dessen Wörter die eigentliche Nachricht binär-Codieren.
In der Modernen Steganographie werden zum verstecken von Informationen z.B. die verlustbehaftete Kompression von jpeg genutzt.

1.2. Klassische Kryptographie

1.2.1. Transposition

Die Transposition ist eine der beiden grundlegenden Verschlüsselungsklassen. Dabei werden die Zeichen der Nachricht umsortiert. Das heißt die Zeichen bleiben gleich ihre Position im Wort oder Text ändert sich jedoch. Da nicht die Zeichen geändert werden, kann mittels einer Häufigkeitsanalyse (wenn man die Sprache kennt) geprüft werden ob eine Transposition vorliegt[1].
Ein Beispiel wäre die Transposition mittels eines Zauns mit Tiefe $n = 3$

```
D   I   I   N   H   E   I
 I S S E N I F C E B I P E  ↦DIINHEIISSENIFCEBIPEETEASSL
  E   T   E   A   S   S   L
```

1.2.2. Substitution

Die Substitution ist die zweite grundlegende Verschlüsselungsklasse. Hierbei werden die Zeichen des Klartextes mit anderen Zeichen ausgetauscht (substituiert) um eine Verschlüsselung zu erreichen.

1.2.3. Einfache Substitution

Bei einer einfachen Substitution werden die Zeichen des Klartextes (Klartextalphabet) durch eine injektive Abbildung $f : A \mapsto B$ auf die Zeichen des Chiffretextalphabets abgebildet.
Kodiere das Alphabet $A = B = A, B, ..., Z$ durch die Zahlen von 0 bis 25: $A \mapsto 0, B \mapsto 1, ..., Z \mapsto 25$

- Verschiebechiffre (Caesar): $f(a) = (a + k \mod 26)$
- Multiplikationschiffre: $f(a) = (a \cdot k) \mod 26$ mit $ggT(k, 26) = 1$
- affine Transformation: $f(a) = (a \cdot k_1 + k_0) \mod 26$ mit $ggT(k_1, 26) = 1$

Da eine beliebige Permutation des Klartextalphabets verwendet werden kann Ergeben sich $26! \sim 403$ Quadrillionen viele Schlüssel. Da jedoch die Häufigkeiten der Zeichen nur Permutiert werden, kann die injektive Abbildung durch eine Häufigkeitsanalyse rekonstruiert werden.

[1] im deutschen müsste auch bei einem mittels Transposition verschlüsseltem Text das 'e' am häufigsten vorkommen.

1.2.4. Homophone Substitution

Um die permutierte Häufigkeit der einfachen Substitution zu vermeiden, werden bei der Homophonen Substitution einem Zeichen aus dem Klartextalphabet mehrere Zeichen aus Chiffretextalphabet zugeordnet.
Die Homophone Substitution ist durch eine surjektive Abbildung des Klartextalphabets A auf das Chiffretextalphabete C durch g: $C \mapsto A$ gegeben. Die g-Urbilder der Klartextbuchstaben sind wegen der Surjektivität nicht leer, und automatisch paarweise disjunkt. Die Abbildung sollte so gewählt werden, dass im Chiffretext alle Zeichen des Chiffretextalphabets gleich Häufig sind, sodass die Häufigkeitsanalyse ins leere läuft.
Beispiel: Homophone Substitution können durch sogenannte Known-Plaintext An-

Buchstabe	Häufigkeit	Homophone
E	17,74%	02 05 14 19 24 26 34 46 50 57 63 68 75 80 83 91 99
I	7,60%	01 15 22 43 52 86 87 98
K	1,40%	44
L	3,49%	04 28 82
N	10,01%	08 21 42 45 53 66 72 77 92 95
P	0,64%	03
R	6,98%	07 11 12 23 30 70 85
S	6,88%	10 18 36 39 61 83 93

K L E I N E S P I E L E R E I ↦ 44 04 24 01 66 57 36 03 98 05 82 99 30 75 52

griffe gebrochen werden.

1.2.5. Polyalphabetische Substitution

Idee:

- Einem Klartextzeichen werden mehrere verschiedene Chriffretextzeichen zugeordnet (Homophone Substitution)
- Die Substitution ist abhängig von der Position des Klartextzeichens im Text

Eine Periodische Substitution ist durch d injektive Funktionen $f_1, f_2, ..., f_d : A \mapsto B$ gegeben. Wobei die Funktion f_n in Abhängigkeit der Zeichenposition i: $n = i \mod d$ verwendet wird. Beispiel Vignère-Chiffre: Die Vignère-Chiffre kann mittelst

Beispiel (Schlüssel = BAND)

Klartext:	V O R L E S U N G E N
Schlüsselstrom:	B A N D B A N D B A N
Chiffretext:	W O E O F S H Q H E A

der Kasiski-Methode gebrochen werden:
Durch die periodische Verwendung des Schlüsselwortes treten im Chiffretext Wiederholungen auf aus denen die Periode (und damit Schlüsselwort länge) herausgefunden werden kann:

- 1. Finde alle Wiederholungen im Chiffretext
- 2. Bestimme die Abstände der Wiederholungen
- 3. Vermute d = häufigster Teiler aller Abstände
- 4. Danach für die d Teilchiffretexte Häufigkeitsanalyse durchführen (wie Caesarchiffre)

Die Sicherheit der Vignère-Chriffre hängt von der Länge des Schlüssels ab (Schlüssel genauso lang wie Klartext heißt Running Key Chipher).

1.2.6. One-Time-Pad

Das One-Time-Pad funktioniert ähnlich wie die Vignère-Chiffre nur das für den Schlüssel gilt:

- der (geheime) Schlüsselstrom ist eine unendliche zufällige Folge von Buchstaben
- Jeder Teil des Schlüsselstroms wird genau einmal zum Verschlüsseln verwendet.

Umsetzung: Klartext und Schlüsselstrom sind Bitfolgen die durch XOR verknüpft werden.
Das One-Time-Pad bietet als einziges perfekt sichere Verschlüsselung !
Der Nachteil ist, dass der Schlüssel genau soviel Speicher benötigt wie der Klartext.

1.2.7. Polygramm-Substitution

Idee: statt einzelnen Buchstaben werden ganze Blöcke von Buchstaben gemeinsam verschlüsselt. Daher auch Blockchiffre genannt. Moderne Chiffren sind ebenfalls Blockchiffren.

2. Grundbegriffe & Definitionen

- Klartext: die zu verschlüsselnde Nachricht/Daten
- Chiffretext: die verschlüsselten Daten
- Chiffrieren: Anwendung eines Verschlüsselungsverfahrens
- Dechiffrieren: Anwendung eines Entschlüsselungsverfahrens
- Schlüssel: Informationsträger für Ver-/Entschlüsselung | kontrolliert die Ver-/Entschlüsselung

2.1. Kryptographisches System

Ein **kryptographisches System** besteht aus 5 Komponenten

- 1. einem Klartextraum $\mathcal{M}$
- 2. einem Chiffretextraum $\mathcal{C}$
- 3. einem Schlüsselraum $\mathcal{K}$
- 4. einer Familie von Chiffrierfunktionen $E_K : \mathcal{M} \mapsto \mathcal{C}, K \in \mathcal{K}$
- 5. einer Familie von Dechiffrierfunktionen $D_K : \mathcal{C} \mapsto \mathcal{M}, K \in \mathcal{K}$

sodass für jedes $M\ in \mathcal{M}$ und jedes $K \in \mathcal{K}$ gilt:

$$D_K(E_K(M)) = M \tag{2.1}$$

jede der Funktionen E_K ist **injektiv** und jede der Funktionen D_K ist **surjektiv**.

> Die Sicherheit des Systems soll nur auf der Geheimhaltung des Schlüssels, und **nicht** auf der Geheimhaltung der Algorithmen beruhen.

2.2. Perfekte Geheimhaltung

Menge der Elementarereignisse: $\Omega = \mathcal{M} \times \mathcal{K}$
$P : \Omega \mapsto \mathbb{R}$ hängt vom konkreten Kryptosystem ab.
Zufallsvariablen:

- msg: $\Omega \mapsto \mathcal{M}$ mit $\text{msg}(M, K) = M$
- key: $\Omega \mapsto \mathcal{K}$ mit $\text{key}(M, K) = K$
- cph: $\Omega \mapsto \mathcal{C}$ mit $\text{cph}(M, K) = E_K(M)$

Definition: Perfekte Geheimhaltung
Ein kryptosystem unterliegt perfekter Geheimhaltung, falls für alle $M \in \mathcal{M}$ und $C \in \mathcal{C}$ gilt:

$$p(M|C) = p(M) \tag{2.2}$$

Heißt die Wahrscheinlichkeit den Klartext M unter der Voraussetzung das der Chiffretext C empfangen wurde ist gleich der Wahrscheinlichkeit den Klartext M zufällig zu wählen. Das heißt aus dem Chiffretext kann keinerlei Information über den Klartext gewonnen werden.
Oder: Kein Klartext wird wahrscheinlicher, wenn man einen Chiffretext kennt.
Notwendig für Perfekte Geheimhaltung:

$$\text{Anzahl Schlüssel} \geq \text{Anzahl möglicher Klartexte} \tag{2.3}$$

2.3. Konfusion & Diffusion

Konfusion & Diffusion sind zwei informelle Begriffe, deren stren mathematische Definition meist unterlbeibt:

- Konfusion: Die statistische Verteilung der Chiffretexte hängt so kompliziert von der Verteilung der Klartexte ab, dass sich hieraus keine Angriffsmöglichkeit ergibt. (Idealerweise sind die Chiffretexte fester Länge gleichverteilt)
- Diffusion: jedes Bit des Klartextes und jedes Bit des Schlüssels beeinflussen möglichst viele Bits des Chiffretextes

Für die Diffusion gibt es noch ein verschärftes Kriterium: Bei veränderung eines Klartext- oder Schlüssel-Bits ändert sich jedes Chiffretext-Bit mit einer W'keit von $\frac{1}{2}$

2.4. Strom- vs. Blockchiffrierung

- **Stromchiffrierung:** Es wird eine Bitfolge erzeugt (Schlüsselstrom), mit der der Nachrichten-Bit-Strom verschlüsselt wird. Optimal ist dies, wenn der Schlüsselstrom genauso lang wie der Nachrichtenstrom ist
- **Blockchiffrierung:** Es werden Gruppen von Bits zusammengefasst und gemeinsam verschlüsselt, oft jede Gruppe mit demselben Schlüssel (wie z.B. Caesar).

2.5. Trusted Authorities - TA's

Trusted Authorities sind "vertrauenswürdige Instanzen"

- stellen einen **beweisbaren** Zusammenhang zwischen der Identität einer Person und ihrem Schlüssel her
- haben die Form von Schlüsselverteilzentren (key distribution centers) oder Zertifizierungsstellen (certification authorities)
- erteilen auf Wunsch **Zertifikate** für die Schlüssel der Benutzer
- solche Zertifikate können zeitlich beschränkt gültig sein
- der im Zertifikat enthaltene authentische Schlüssel wird zumeist nur für die Vereinbarung eines Sitzungsschlüssels verwendet

3. Moderne symmetrische Verfahren

Die klassischen Verfahren (bis hin zur Enigma) waren Zeichenorientier. Mit den Computern ist die bitweise Verschlüsselung aufgekommen. Diese bietet u.a. die Vorteile, dass eine häufigkeitsanalyse schwieriger ist und zum anderen, dass bspw. eine bitweise XOR-Operation einfach in Hardware implementierbar und leicht umkehrbar ist.

3.1. Feistel-Netzwerke

Feistel-Netzwerke bilden die zentrale Komponente gängiger Symmetrischer Kryptoverfahren, daher werden diese folgend erst einmal unabhängig von ihrem einsatz erläutert: Die Idee ist Blöcke gerader Länge in gleichlange Hälften zu zerlegen:

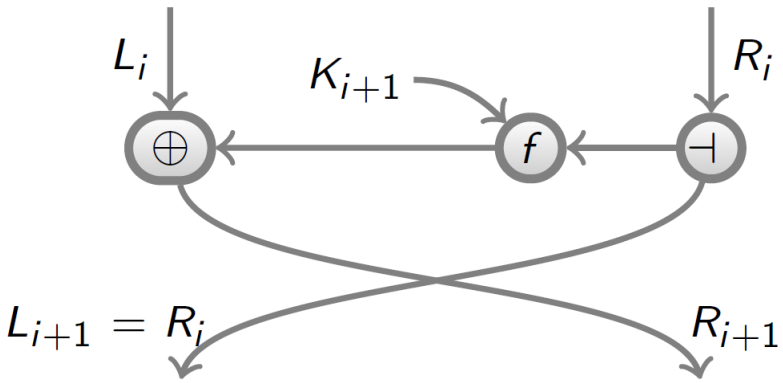


Abbildung 3.1.: Feistel-Netzwerk-Kern-Komponente. Es können mehrere hintereinander geschaltet werden, i bezeichnet die Runde

$B_i = L_i R_i$[2].
Bezeichnung:

- i Indiziert die Runde
- L_i ist die linke Hälfte des Blockes B_i aus der Runde i
- R_i ist die rechte Hälfte des Blockes B_i aus der Runde i

[2]Da man mehrere solcher Durchläufe hintereinander machen kann steht i für die Rundenanzahl

- K_{i+1} ist der Rundenschlüssel zur $i+1$-ten Runde
- f berechnet einen Funktionswert aus R_i und dem K_{i+1}-ten Rundenschlüssel
  - Einzige Bedingung an f: ohne Kenntnis des Schlüssels K_{i+1} soll $f(R_i, K_{i+1})$ praktisch nicht berechnbar sein. f muss nicht invertierbar sein !
  - f kann beliebig komplex werden, solang es effizient berechenbar bleibt
- $\dashv$ ist eine Aufteilung, R_i wird nach f und nach L_{i+1} weitergeleitet
- $\oplus$ XOR-Verknüpft L_i mit dem Funktionswert von $f(K_{i+1}, R_i)$

Chiffrierung:

$$L_{i+1} = R_i \tag{3.1}$$
$$R_{i+1} = L_i \oplus f(R_i, K_{i+1}) \tag{3.2}$$

Für die Dechiffrierung wird die Richtung des Netzwerkes umgedreht:
Dechiffrierung:

$$R_i = L_{i+1} \tag{3.3}$$
$$L_i = R_{i+1} \oplus f(R_i, K_{i+1}) \tag{3.4}$$
$$= L_i \quad \underbrace{\oplus f(R_i, K_{i+1}) \oplus f(R_i, K_{i+1})}_{\text{zwei XOR hintereinander heben sich auf!}} \tag{3.5}$$

Um L_i zu bestimmen setzt man für R_{i+1} die Gleichung aus dem Chiffrier-Vorgang ein. Die beiden XOR-Verknüpfungen mit f hintereinander heben sich auf und man erhält wieder L_i zurück $\rightarrow$ Hiermit kann gezeigt werden dass ein Feistel-Netzwerk im Umgekehrten Betrieb korrekt Dechiffriert.

3.2. Data Encyption Standard (DES)

Der Symmetrische **D**ata **E**ncryption **S**tandard basiert auf dem Feistel-Netzwerk und kombiniert Substitutions- und Transpositionschiffren in 16 Runden.
Blockchiffre:

- 64-Bit Klartextblöcke T
- 64-Bit Schlüssel K

  $\Rightarrow$ 48-Bit Rundenschlüssel $K_i, i < 16$

  - jedes Byte muss **ungerade Parität** besitzen. Wird durch ein Paritätsbit am Ende jedes Bytes erreicht. Daher ist die effektive Schlüssellänge 56 Bits

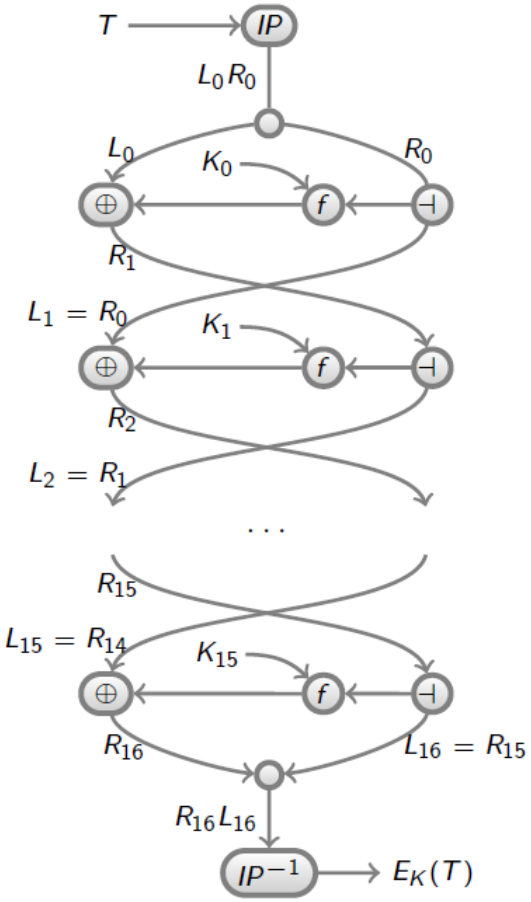
Was zu folgender Struktur des DES führt: Die Initial-Permutation löst lokale Information auf und verteilt diese auf den gesamten Klartext

- Klartext T
- Initiale Permutation $IP(T) = L_0R_0$
- Es werden 16 Runden durchgeführt:

$$L_i = R_{i-1} \quad (3.6)$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \quad (3.7)$$

- Funktion $f : 2^{32} \times 2^{48} \mapsto 2^{32}$
  - R_i wird auf 48 Bits erweitert
  - Aus dem Schlüssel K wird, nach einer festen Vorschrift, ein 48-Bit-Teilschlüssel K_i generiert
  - $R_i \oplus K_i$
  - das Ergebnis wird mit S-Boxen auf 32-Bit reduziert
- Am Schluss invers $IP^{-1}(R_{16}L_{16})$

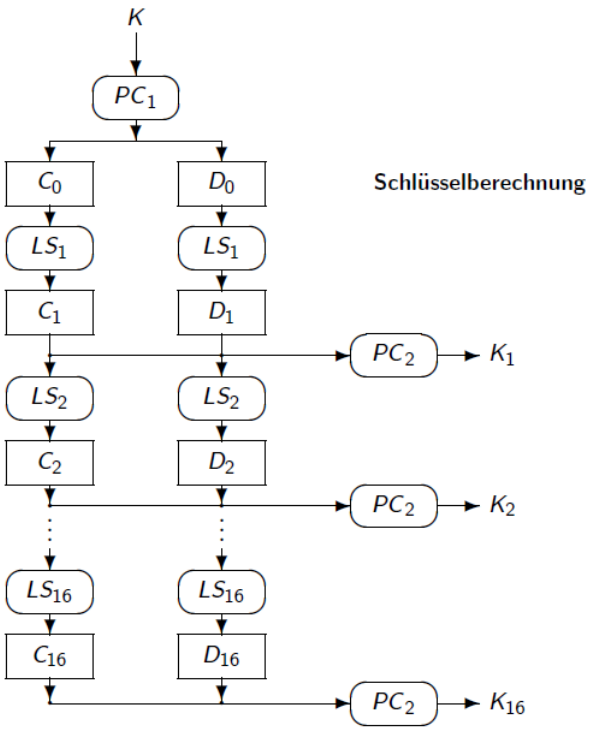


Erläuterung zu den einzelnen Schritten:
Permutation

- Die Eingangspermutation löst lokale Information auf

Funktion f

- R_{i-1} wird mit einer Bit-Auswahl-Tafel E auf 48 Bit erweitert. Geschieht durch zyklische Verschiebung nach Rechts und Verdopplung gewisser bits
- 48-Bit-Rundenschlüssel K_i wird nach fester Vorschrift aus dem 56-Bit Schlüssel K generiert
  - PC_1 reduziert K auf 56 Bit (entfernt Paritätsbits)
  - C_i, D_i sind 28-Bit Blöcke
  - LS_i sind zyklische Linksshifts um 1 oder 2 Positionen (fest codiert in einer Tabelle)
  - PC_2 entfernt 8 der 56 Bits und permutiert diese
  - jeder Rundenschlüssel K_i hat 48 Bits
- XOR-Verknüpfung von R_{i-1} und K_i
- Acht Substitutions-Boxen (S-Boxen) auf das Ergebnis anwenden. Jede, fest vorgegebene, S-Box reduziert 6-Bit auf 4-Bit, womit das Ergebnis der Funktion f wieder 32-Bit besitzt.
  - Jede S-Box ist durch eine 4×16-Matrix M gegeben
- 32-Bit Ergebnis der S-Boxen wird mit P[a] Permutiert. Das ist das Ergebnis von f



[a] ebenfalls eine fest Vorgegebene Matrix

Anmerkungen zum DES:

- DES beschränkt sich auf sehr einfache Bitoperationen: XOR, Bitauswahl, Permutation und ist daher sehr gut in Hardware umsetzbar

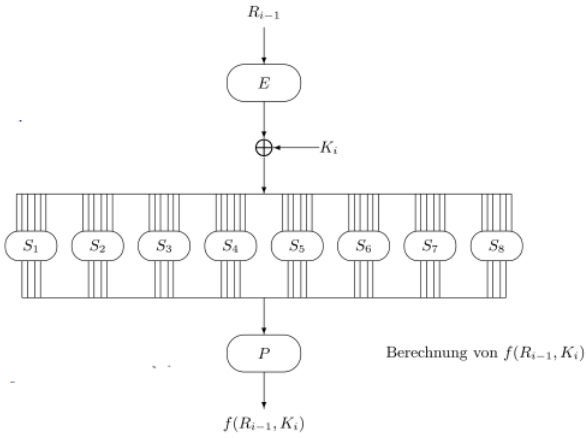


Abbildung 3.2.: Berechnung von f

- Auswahltabelle E: jede Änderung eines Klartextbits betrifft nach wenigen Runden alle Bits des erzeugten Chiffretextes
- Permutation P: Jedes Klartextbit wirkt sich bei jeder Runde auf eine andere S-Box aus

$\Rightarrow$ E und P sorgen für Diffusion

Sicherheit von DES

- Für einen festen Schlüssel K ist DES-Verschlüsselung ingesamt eine bijektive Funktion auf der Menge aller 64-Bit-Blöcke. Eine wiederholte Anwendung liefert irgendwann den Klartext !

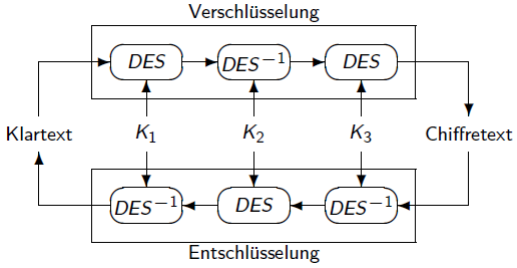
$\Rightarrow$ spätestens nach $2^{64}+1$ anwendungen von $E(k)$ erhält man wieder den Klartext (weil bijektiv)

$\Rightarrow$ Zyklen = kleinste Anzahl von Iterationen der Chiffrierung, bis sich der Klartext wiederholt. Es existieren 4 Schlüssel mit Zyklenlänge 2

Frößere Sicherheit für DES

- Verdopplung der Schlüssellänge auf 112 Bit

- **Triple-DES:** drei verschiedene Schlüssel K_1, K_2, K_3 nach folgendem Schema:



3.3. Advanced Encryption Standard (AES)

Der Advanced Encryption Standard benutzt keine Feistel-Struktur !

Grundkonzepte AES

- Ein- Ausgabeblöcke haben 128 Bit $\Rightarrow$ 16 Bytes
- der Algorithmus operiert auf (4×4)-**Byte**-Matrizen[3] sogenannten **Zuständen** die spaltenweise befüllt werden:

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

$\Longrightarrow$

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$\Longrightarrow$

out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}

- die 4 wesentlichen Operationen sind:
  - Substitute Bytes: substitution mittels S-Box
  - Shift Rows: einfache Permutation auf Zeilen der Zustandsmatrix
  - Mix Collumns: substituiert jedes Byte einer Spalte durch eine Funktion aller Bytes der Spalte
  - AddRoundKey: Bitweoses XOR des Zustands mit dem entsprechenden Rundenschlüssel

  $\Rightarrow$ nur hier wird der Schlüssel benutzt
- Shift Rows, Mix Columns, Sub Bytes sind unabhängig vom Schlüssel und erst im Zusammenspiel mit AddRoundKey erhält das Verfahren Sicherheit
- Jede der Operationen ist umkehrbar
- bei der Dechriffrierung werden die Rundenschlüssel in umgekehrter Reihenfolge benutzt. Die Dechiffrierung ist nicht identisch mit der Chiffrierung

[3] also 128 Bit :D

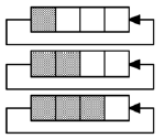
Details zu den einzelnen Schritten:

- **Substitution**: Die S-Box ist wie beim DES eine festgelegte Tabelle. Die S-Box jedoch kann im Gegensatz zum DES berechnet werden:
  - Zum zu substituierenden Byte b wird das multiplikative Inverse $b' = b^{-1}$ in GF(2^8) berechnet.
  - b' wird einer affinen $\mathbb{Z}_2$-Transformation unterzogen (Multiplikation mit einer Matrix und anschließender Addition eines Vektors)[4]. Die Matrix ist Invertierbar!
- **ShiftRows**:

$$\text{ShiftRows}(s) = s'$$

s

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$



s'

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

- **Mix Collumns:**
  - Operiert separat auf jeder Spalte eines Zustands
  - Anwendung einer invertierbaren Substitutionsfunktion f auf die Spalten des Zustandes:
    - Die Funktion f bildet 4 Bytes auf 4 Bytes ab
    - Die Funktion f ist eine Matrix-Multiplikation über GF(2^8). Es ist eine festgelegte Matrix, deren Einträge als Byte zu lesen sind und eine Zeile wird als Polynom wie in 7.2
  - Zusammen mit Shift-Rows erhält man nach einigen Runden einen Lawineneffekt (möglichst jedes Bits des Klartextes beeinflussen möglichst alle Bits des Chiffretextes)
- **AddRoundKey**
- bitweises XOR mit dem Zustand des Rundenschlüssels

[4] Hierfür werden die Operatoren in $\mathbb{Z}_2$ verwendet

- der Rundenschlüssel besteht aus 4 32-Bit Worten w die mit folgenden Key-Schedule erzeugt werden:

$$r_0 r_1 r_2 ... r_{15} := w[i...(i+3)] \begin{cases} i = 0 & \text{bei erstem Schritt} \\ i = 4 \cdot j & \text{in Runde j=1,...,10} \end{cases}$$

- bei der Dechiffrierung werden die Rundenschlüssel in umgekehrter Reihenfolge benutzt

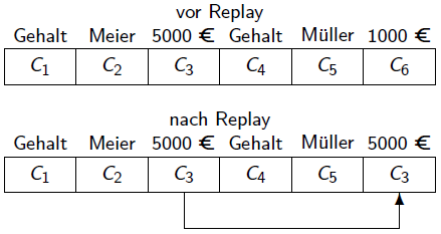
3.4. Betriebsmodi von Blockchiffren

Modus	Beschreibung	Anwendung
Electronic Code Book (ECB)	Unabhängige Kodierung einzelner blöcke mit demselben Schlüssel	Sichere Übertragung einzelner Werte
Cipher Block Chaining (CBC)	Kodierung von XOR-Verknüpfung mit letztem Chiffretextblock	Block-orientierte Übertragung; Authentifizierung
Cipher Feedback (CFB)	Verschlüsselung wirkt auf ein Feedbackregister der Länge n; zur Kodierung werden r Bits ($r < n$) des Registers mit den Klartextblöcken verknüpft	Block-orientierte Übertragung; Authentifizierung
Output Feedback (OFB)	Ähnlich CFB; aber Eingabe der Verschlüsselung ist vorherige Ausgabe	Strom orientierte Übertragung (z.B. Satellit)
Counter (CTR)	Jeder Klartextblock wird mit einem verschlüsselten Zähler XOR-verknüpft	Block orientierte Übertragung; hohe Geschwindigkeit

3.4.1. Electronic Code Book

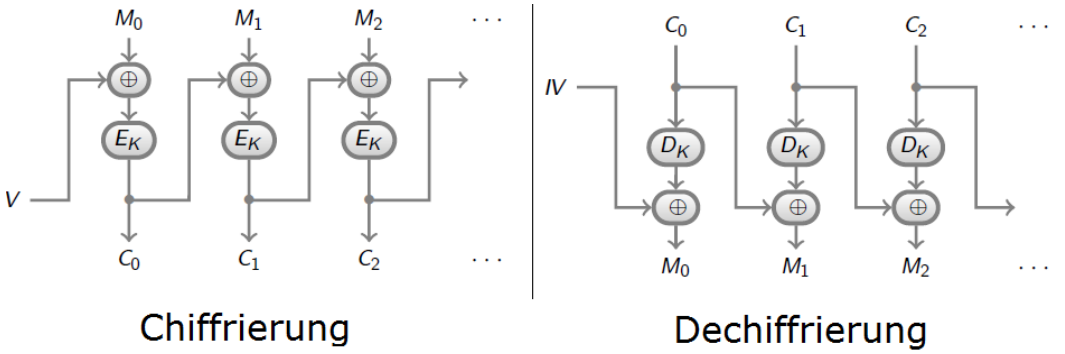
- Klartext M wird in n-Bit Blöcke zerlegt
- Jeder Block wird mit demselben Schlüssel K verschlüsselt

Probleme : Ermöglicht angriffe mit bekannten Klartexten. Eine Replay-Attacke ist in folgender Abbildung gezeigt, wenn man weiß, dass jemand mehr verdient, kopiert man den verschlüsselten Wert an eine weitere stelle:



3.4.2. Chipher Block Chaining (CBC)

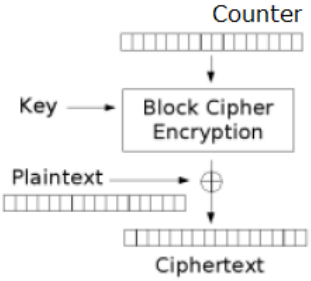
Idee: Jeder Chiffretextblock wird aus dem zugehörigen Klartextblock und dem vorhergehenden Chiffretextblock berechnet (XOR-Verknüpft).
⇒ Der Klartext wird mit dem vorangehenden Chiffretext XOR-Verknüpft und anschließen mit dem Schlüssel K verschlüsselt. Eigenschaften:



- Jeder Chiffretextblock ist abhängig von allen vorhergehenden Chiffretextblöcken. Statischte eigenschaften des Klartextes werden über den Chiffretext verteilt
- Reihenfolge der Chiffretextblöcke ist wichtig !
- Der CBC-Modus ist selbstsynchronisierend: Das heißt, ein Bitfehler (beim übertragen oder durch einen Angreifer) wirkt sich **nur** auf die **folgenden beiden** entschlüsselten Klartextblöcke aus.
- ein Angreifer kann hierdurch an bekannten stellen Bitfehler auslösen und beobachten

3.4.3. Counter (CTR)

Idee: n-Bit Zähler wird verschlüsselt und im Klartext M XOR-Verknüpft, der Zähler Z wird für jeden Block inkrementiert modulo 2^n: **Eigenschaften:**



- Chiffriervorgang: $C_i := M_i \oplus E_K(Z)$ ()
- Blöcke können parallel bearbeitet werden
- Preprocessing: Die Ausgabe der Chiffrierfunktion hängt nicht vom Klartext ab und kann vorberechnet werden
- Random Access: Zum Ver-/Entschlüsseln wird kein Zugriff auf andere Blöcke benötigt
- CTR ist mindestens so sicher wie die anderen Modi
- Es wird nur die Chiffrierfunktion benutzt !

4. Asymmetrische Verfahren

Das Problem bei den symmetrischen Verfahren ist das beide Kommunikationspartner den Schlüssel brauchen. Die Lösung die erstmal eingeführt wurde waren Schlüsselverteilerzentren mit denen verschlüsselt kommuniziert ein Sitzungsschlüssel für die Kommunikation zwischen **A** und **B** ausgehandelt werden konnte. Dafür muss jedoch 1. ebenfalls ein vertraulicher Schlüssel zwischen Server und dem Nutzer festgelegt werden und 2. muss der Server 100% Vertrauenswürdig sein.
Diffie und Hellman kamen mit der Idee eines asymmetrischen Verfahren:

- jeder besitzt zwei Schlüssel: einen **privaten** Schlüssel, der geheim gehalten werden muss und einen **öffentlichen** Schlüssel der jedem zur Verfügung steht
- Die Schlüssel sollen mathematisch voneinander abhängen, es soll jedoch **praktisch unmöglich** sein den einen Schlüssel aus dem anderen abzuleiten.
- Authentizität und Integrität müssen für den öffentlichen Schlüssel garantiert sein, Vertraulichkeit ist nicht erforderlich

Die Anforderungen an den Algorithmus sind:

- Es ist für jeden Teilnehmer X leicht ("computationally easy") die Transformationen E_x und D_x zu finden. D.h. ein Paar aus öffentlichem und privaten Schlüssel zu erzeugen
- Es ist für die Senderin Alice leicht, mit hilfe des öffentlichen Schlüssels von Bob und der Nachricht M den Geheimtext $C = E_B(M)$ zu erzeugen
- Für den Empfänger Bob ist es ebenfalls leicht, die erhaltene Nachricht zu entschlüsseln: $M = D_B(C) = D_B(E_B(M))$
- Es ist schwer ("computationally infeasible") aus dem öffentlichen Schlüssel den privaten abzuleiten. Aus E_B kann man D_B nicht bestimmen !
- Es ist schwer, aus dem öffentlichen Schlüssel und dem Chiffretext C den Klartext M abzuleiten
- Die Ver- und Entschlüsselungsalgorithmen können in beliebiger Reihenfolge angewendet werden: $M = E_B(D_B(M)) = D_B(E_B(M))$. (wird nur von RSA und elliptic Curve erfüllt, ist eher optional)

4.1. RSA

RSA erfüllt die Anforderungen die Diffi und Hellman an einen Algorithmus gestellt haben. RSA basiert auf modularer Arithmetik und Eigenschaften von Primzahlen: man benutzt, dass es berechnungsmäßig schwer ist das Produkt zweier großer Primzahlen zu faktorisieren. RSA ist eine Blockchiffre, Klar- und Geheimtextblöcke werden als große Zahlen aufgefasst, ebenso der Schlüssel.

4.1.1. Schlüsselerzeugung für RSA

Die Erzeugung eines Schlüsselpaares für asymmetrische Verschlüsselung beruhend auf RSA:

- Alice erzeugt zwei große Primzahlen p und q von ungefähr gleicher Länge (p und q müssen unterschiedlich viele Bits haben, aber nahe beinander liegen)
- Alice berechnet $n = p \cdot q$ und den Wert der Euler'schen Funktion: $\varphi(n) = (p-1)(q-1)$ folgt aus 7.3.4
- Alice wählt eine Zahl $e \in \mathbb{Z}^*_{\varphi(n)}$, d.h. $e < \varphi(n)$ und $\text{ggT}(e, \varphi(n)) = 1$
- Mit Hilfe des erweiterten euklidischen Algorithmus berechnet Alice: $d = e^{-1} \mod \varphi(n)$
- Der öffentliche Schlüssel von Alice ist (n, e), der private ist d

4.1.2. RSA-Verschlüsselung

Bob chiffriert eine Nachricht M für Alice, die dies dechiffriert:

- Zur Chiffrierung führt Bob die folgenden Schritte aus:
  - Bob besorgt sich den authentischen öffentlichen Schlüssel (n, e) von Alice
  - Bob stellt seinen Klartext M als Zahl in $\mathbb{Z}_n$ dar [5]
  - Bob berechneet $C = M^e \mod n$ mit Hilfe der schnellen Exponentiation. **Verschlüsselungs Schritt!**
  - Bob übermittelt $E_A(M) = C$ an Alice
- Zur Dechiffrierung führt Alice den folgenden Schritt aus:

[5] da die Berechnungen sehr aufwendig sind, wird der Klartext M meist mit symmetrischen Verfahren wie AES verschlüsselt und nur der Schlüssel mit dem RSA-Verfahren

– Mit ihrem privaten Schlüssel d berechnet Alice:
$M = D_A(C) = C^d \mod n$. **Entschlüsselungs Schritt!**

4.1.3. RSA-Signaturverfahren

Alice signiert die Nachricht M für Bob, der die Signatur verifiziert:

- Zur Signierung führt Alice die folgenden Schritte aus:
  - Alice stellt M als Zahl in $\mathbb{Z}_n$ dar [6]
  - Alice berechnet $C = M^d \mod n$ mit Hilfe der schnellen Exponentiation
  - Alice sendet (M und) die Signatur $D_A(M) = C$ an Bob
- zur Verifikation und zum Erhalt der Nachricht führt Bob die folgenden Schritte aus:
  - Bob besorgt sich den authentischen öffentlichen Schlüssel (n, e) von Alice
  - Bob berechnet $E_A(C) = C^e \mod n$. Wenn $C^e \mod n$ keinen vernünftigen Klartext ergibt, wird die Signatur abgelehnt, anderenfalls wird sie akzeptiert und $C^e \mod n$ als M anerkannt[7]

4.1.4. Korrektheit von RSA

Die Ver- und Entschlüsselung durch RSA lässt sich wie folgt darstellen:

> Für ungleiche Primzahlen p und q setzte $n = p \cdot q$. Erfüllen $e, d \in \mathbb{Z}_{\varphi(n)}$ die Bedingung $e \cdot d \mod \varphi(n) = 1$, so gilt für jede Nachricht $M \in \mathbb{Z}_n$:
>
> $$(M^e \mod n)^d \mod n = M \tag{4.1}$$

Beweis:

Um die Korrektheit von 4.1 Beweisen zu können wird noch ein Lemma (eine abgespeckte version vom Chinesischen Restesatz) benötigt (q, p sind die Primzahlen die

[6] Da die Berechnungen sehr aufwendig sind, ist M meist nicht der Klartext selber sondern nur der Fingerabdruck, berechnet durch eine Hashfunktion, vom Klartext

[7] Bob vergleicht M mit $E_A(C) = C^e \mod n$ und akzeptiert bei Übereinstimmung Alice als Sendern.

$n = p \cdot q$ aus 4.1 erzeugen):

Lemma:

$$x \mod n = m \mod n \iff \begin{matrix} x \mod p = m \mod p \\ x \mod q = m \mod q \end{matrix}$$

Beweis für „⇒“:

- Die modulare Gleichung $x \mod n = m \mod n$ kann auch gelesen werden als: *n teilt die Differenz von x und m* $\Rightarrow n|x - m$
- da $n = p \cdot q$ gilt: $q|pq|n$ und $p|pq|n$[8]
- daraus folgt das *p pq teilt und damit auch* $(x - m) \rightarrow p|pq|x - m$
- und damit auch: $q|pq|x - m$
- Die modulare Gleichung $x \mod p = m \mod p$ kann auch gelesen werden als: *p teilt die Differenz von x und m* $\Rightarrow p|x - m$[9]

Beweis für „⇐“:

In der selben Leseart wie oben:

- $p|x - m$ & $q|x - m$
- Betrachtet man die Primfaktorzerlegung von $x - m$ **muss** mindestens ein p und ein q vorkommen (da p und q verschiedene Primzahlen sind $p \neq q$)
- damit folgt $(x - m)$ wird auch von pq geteilt $\rightarrow pq|x - m$

□

[8] gelesen als: q teilt pq (und damit auch n) und p teilt pq (und damit auch n)

[9] Teilbeweis abgeschlossen. Gilt natürlich genauso für $q|x - m$

Korrektheits Beweis für RSA:
wir wissen:

$$e \cdot d \equiv_{\varphi(n)} 1 \quad \Leftrightarrow \quad e \cdot d = k \cdot \varphi(n) + 1^{10}$$

Beweise das (4.1) für p und für q gilt, dann kann aus dem vorangegangen Lemma gesagt werden das 4.1 auch für n gilt!:
a) Zeige, dass $(M^e)^d \mod p = M \mod p$:
Fall 1: $\text{ggT}(M, p) = 1$
Setzte für $(M^e)^d = M^{ed}$ die Gleichung $ed = k\varphi(n) + 1$ ein und Vereinfache diese mit Exponential-Regeln und verwende den Satz von Ferma(alle Gleichungen $\mod p$):

$$(M^e)^d = M^{k\varphi(n)+1} = M \cdot (\underbrace{M^{(p-1)}}_{\text{nach Satz von Euler (und Fermat)} = 1})^{(q-1)k} = M \qquad (p)$$

Fall 2: $\text{ggT}(M, p) \neq 1$ oder $\text{ggT}(M, p) = p$
damit $M = 0(p)$
also

$$(M^e)^d = (0^e)^d = 0 = M \qquad (p)$$

b) Analog (für q): $(M^e)^d = M \qquad (q)$

$\xrightarrow[\text{Lemma}]{}$ wegen Lemma gilt: $(M^e)^d = M \qquad (n)$,

das würde erst einmal bedeuten, es hat den selben rest !
Da aber $0 < M < n$ ist es wirklich der Klartext M (und kein kongruent mit selben Rest)

$$(M^e)^d \mod n = M \quad \text{,da} \quad 0 \leq M < n$$

□

Aus Fall 2 folgt, dass wenn M nicht teilerfremd zu n ist, dann erhält man für $\text{ggT}M(M, n)$ p oder q. Die Wahrscheinlichkeit ein M zufällig zu wählen sodass $\text{ggT}(M, n) \neq 1$ ist $\frac{1}{q} + \frac{1}{p}$ wobei p, q 300 stellige Zahlen sind.

[10] entsteht aus der darstellung des Restes 1 mit dem erweiterten Euklidischen Algorithmus:
$1 = k\varphi(n) + ed$

4.1.5. Geheimhaltung + Authenzität bei RSA

Der Vorgang, dass Alice eine Nachricht Signiert mit ihrem privaten Schlüssel und anschließend mit dem öffentlichen Schlüssel von Bob verschlüsselt:

$$C = E_B(D_A(M))$$

kann zu Dechiffrierfehlern führen wenn der Moduli von Alice $n_A > n_B$ ist. Werte von $D_A(M)$ werden auf $\mathbb{Z}_{n_A}$ abgebildet, wenn n_B jetzt kleiner als n_A ist werden alle Werte bei $E_B(M_{\text{signiert}})$ auf $\mathbb{Z}_{n_B}$ reduziert. Die Fehlerwahrscheinlichkeit für Dechiffrierfehler ist:

$$\frac{N_A - n_B}{n_A} \tag{4.2}$$

Den Vorgang umzudrehen also:

$$C = D_A(E_B(M))$$

ist ebenfalls keine Lösung denn die Signatur von Alice kann mit dem öffentlichen Schlüssel von Alice entfernt werden.
Eine Lösung wäre durch technische Infrastruktur dafür zu sorgen das Signaturmoduli $n_{X_0} < n_{X_1}$ Chriffriemoduli sind. Da jedoch meist nur der Fingerprint eines Klartextes M signiert wird, kommt es meist nicht zu diesem Problem.

4.1.6. Zeitbedarf von RSA

Im Vergleich zu syymetrischen Verfahren (DES,AES) ist RSA aufgrund der Arithmetik bei großen Zahlen langsamer (Datendurchsatz RSA: einige Mbps ; DES: einige Gbps(hardware)). Daher wird, wie schon erwähnt, meist eine Kombination aus symmetrischen Verfahren und RSA gewählt:

- Klartext M wird mit einem symmetrischen Verfahren verschlüsselt. Der Schlüssel wird mittels RSA übertragen
- Aus dem Klartext M wird mittels einer Hashfunktion ein Fingerprint berechnet und dieser mit dem privaten Schlüssel verschlüsselt. Übertragen wird der Fignerprint und der Klartext M, womit die Gegenseite durch Anwendung der Hashfunktion auf M und durch Benutzung des öffentlichen Schlüssel des Senders beide Fingerprints überprüft werden können (und damit auch Manipulationen erkannt werden können).

4.1.7. Faktorisierung und Sicherheit von RSA

Die Sicherheit von RSA beruht auf der Schwierigkeit des Faktorisierungsproblems, also die Primfaktoren einer natürlichen Zahl finden ($n = pq$, nur n ist öffentlich, die Zerlegung pq jedoch unbekannt). Ist die Faktorisierung von n bekannt, so ist der Private Schlüssel d leicht aus (n, e) bestimmbar.
Es gilt sogar:

> Die Bestimmung des privaten Schlüssel d aus dem öffentlichen Schlüssel $\langle n, e\rangle$ ist berechnungsmäßig äquivalent zur Faktorisierung von n.[11]

Beweis: Annahme: Ein Angreifer kennt d und e, n. **zu zeigen:** er kann n faktorisieren.
Es gilt:

$$ed - 1 = k + \varphi(n) \quad \underrightarrow{\text{wegen Euler}} \quad x^{ed-1} \mod n = 1 \quad \text{falls ggT}(x, n) = 1$$

Der Angreifer stellt $ed-1$ Binär dar und spaltet die Nullen am ende der Darstellung ab mit: $ed - 1 = 2^l \cdot m, m$ ungerade. **Algorithmisches Vorgehen:**

1. Der angreifer wählt $x \in \mathbb{Z}_n$ zufällig

2.1 Falls $ggT(x, n) \neq 1$ hat er p oder q gefunden.

2.2 Sonst: Versucht der Angreifer eine nicht-triviale Quadratwurzel zu finden. Da n keine Primzahl, kann es Zahlen geben mit $a \mod n = 1, a \notin \{1, -1\}$ ergeben. Dafür versucht er das $ed - 1$ nach und nach aufzubauen, indem er erst x^m nimmt und das immer weiter quadriert. Er sucht also ein $0 \leq i < l$ mit:

$$x^{2^i \cdot m} \neq \pm 1(n) \quad \& \quad x^{2^{i+1} \cdot m} = 1(n)$$

3.1 falls i gefunden (nicht sicher), dann setzte:

$$w = x^{2^i \cdot m} \mod n, \quad \text{es gilt:} 1 < w < n - 1$$

$1 < w < n - 1$ folgt aus der ersten Bedingung das $x^{2^i \cdot m}$ weder 1 noch -1 ist.

$$(w + 1)(w - 1) = w^2 - 1 = x^{2^{i+1} \cdot m} - 1 = 0(n)$$

[11] hierfür gibt es einen Probabilistischen Beweis: Video 09 ab Minute 46.

da das obige gilt, folgt

$$(w+1)(w-1) = r \cdot n$$

und wegen der $1 < w < n-1$ bedinung gilt:

$$(w+1) < n = pq$$
$$(w-1) < n = pq$$

Daher weiß man, dass sich die Primfaktoren p und q auf die beiden Faktoren $(w+1)$ und $(w-1)$ aufteilen, denn beide können nicht in einem stecken, da diese $< n$

$$\Rightarrow p|(w+1) \quad \& \quad q|(w-1) \quad \text{oder anderum}$$

3.2 $\text{ggT}(w+1, n) \in \{p, q\}$

Der Vorgang ist besser als das Raten in 2.1 denn die Wahrscheinlichkeit ein passendes x zu wählen ist $\frac{1}{2}$. Bei k-maliger Wiederholung ergibt das $1 - \frac{1}{2^k}$ (wahrscheinlich genug!).
Das heißt:

> Falls die RSA-Schlüssel (e, n) und (d, n) bekannt sind, gibt es einen probabilistischen Polynomzeit Algorithmus, der n faktorisiert.

4.1.8. RSA-Modulus Hinweise

Bei der Wahl des modulus sollte auf folgendes geachtet werden:

- Die Differenz $p - q$ sollte nicht zu klein sein. Weil ansonsten liegen p und q sehr nah an $\sqrt{n}$ und diese könnten mit einer Heuristik bestimmt werden. Also teste Teilbarkeit von n durch ungerade Zahlen in der Nähe von $\sqrt{n}$
- Günstig ist es p und q als starke Primzahlen mit einer kleinen Differenz in der Bitlänge wählen. Dann haben p und $p - q$ dieselbe Größenordnung

Eine Primzahl heißt stark, wenn:

- $p - 1$ hat einen großen Primfaktor r
- $p + 1$ hat einen großen Primfaktor

- $r-1$ hat einen großen Primfaktor

4.1.9. Erzeugung großer Primzahlen

Es gibt heuristische Methoden um Zahlen zu wählen und diese auf Prim zu testen:

- die Dichte der Primzahlen um die Zahl n ist: $\frac{1}{\ln(n)} \rightarrow$ unter $\ln(n)$ zufäligen Zahlen ist ca. eine Primzahl
- Erzeuge eine ungerade k-stellige Binärzahl

$$m = b_1 b_2 ... b_k, \qquad \text{mit} \quad b_1, b_k = 1$$

- Teste für $i = 0, 1, 2...$ ob $m + 2i$ eine Primzahl ist
- wenn innerhalb von $2k$ Versuchen kein Prim gefunden wurde, wähle ein neues m. $(2k = 2\log_2(n) > 2\ln(n))$

4.1.10. Primzahltests

Der Sieb des Eratosthenes ist exponentiell in der Bitlänge und daher nicht praktisch eingesetzt.

4.1.10.1. Primzahltest mit dem Satz von Fermat

Idee:

- wähle zufällig $1 < a < n$
- teste ob $a^{n-1} \mod n \neq 1$ (Satz von Fermat)
- falls "ja": n ist keine Primzahl; falls "nein": vermute n Primzahl

Es existieren jedoch Zahlen welche die Gleichung $a^{n-1} \mod n = 1$ erfüllen, aber zusammengesetzte Zahlen sind.
Definition: Eine Carmichael-Zahl ist eine zusammengesetzte Zahl n mit der Eigenschaft, dass für alle Zahlen $0 < a < n$ gilt: $a^{n-1} \mod n = 1$.
Es gilt zusätzlich:

> Ist n zusammengesetzt und keine Carmichael-Zahl, gilt für eine Mehrheit der Zahlen $0 < a < n : a^{n-1} \mod n \neq 1$

Damit der Primzahltest funktioniert wird noch ein Kriterium für Carmichael-Zahlen benötigt:

Verbesserung von Rabin-Miller

Miller-Zeuge:

> Wenn n eine Primzahl ist, dann hat die quadratische Gleichung $x^2 \mod n = 1$ nur die beiden Lösungen $x = 1$ und $x = -1$. Gibt es eine weitere Lösung für x, dann ist n keine Primzahl. Findet man also ein x ungleich 1 oder -1 mit $x^2 \mod n = 1$, so spricht man von einer nichttrivialen Quadratwurzel von $1 \mod n$

In der Kurzfassung:
Sei $n > 1$ und sei $b \in \{1, ..., n-1\}$ und erfüllt:

$$b \neq 1 \qquad b \neq n-1 \qquad b^2 \mod n = 1$$

Dann ist n eine zusammengesetzte Zahl. Die Zahl b heißt Miller-Zeuge für n oder nicht-triviale Quadratwurzel von 1.
Vorgehen:
Idee: suche Miller-Zeugen b durch fortgesetztes Quadrieren

- schreibe $n-1$ als $2^l \cdot m$ mit m ungerade
- für zufälliges $1 < a < n$ betrachte de Folge:

$$(a^m \mod n), (a^{2m} \mod n), (a^{4m} \mod n), ..., (a^{2^l \cdot m} \mod n)$$

Die Auswertung der Folge ergibt (? steht für eine Zahl ungleich 1 und ungleich $n-1$):

- $1, 1, ..., 1 \rightsquigarrow$ mögliche Primzahl
- $?, ..., ?, 1, ..., 1 \rightsquigarrow$ zusammengesetzte Zahl, folgt plötzlich auf ein ? eine 1, ohne dass vorher eine 1 oder -1 kam, hat man eine nicht triviale Wurzel gefunden.
- $?, ..., ?, n-1, 1, ..., 1 \rightsquigarrow$ mögliche Primzahl
- $?, ..., ?$ bzw. $?, ..., ?, n-1 \rightsquigarrow$ zusammengesetzte Zahl, weil das letzte Ergebnis $n-1$ ist der Fermattest $a^{n-1} \mod n$ und der müsste für eine Primzahl 1 sein!

Beispiel:

- Sei $n = 561$ (Carmichael-Zahl).
- wähle zufällig $a = 5$ und mache den Fermat-Test: $5^{560} \mod 561 = 1$
- Es gilt $560 = 2^4 \cdot 35$. Berechne:

- $5^{35} \mod 561 = 23$
- $5^{2 \cdot 35} \mod 561 = 23^2 \mod 561 = 529$
- $5^{2^2 \cdot 35} \mod 561 = 529^2 \mod 561 = 463$
- $5^{2^3 \cdot 35} \mod 561 = 463^2 \mod 561 = 67$
- $5^{2^4 \cdot 35} \mod 561 = 67^2 \mod 561 = 1$ (wenn n eine Primzahl sein soll, darf $x^2 \mod n = 1$ nur für $x = 1$ und $x = -1$ gelten! hier ist 67 eine nicht-triviale Wurzel)

Also ist $b = 67$ ein Miller-Zeuge und 561 damit eine zusammengesetzte Zahl.
Eine Zahl welche den Fermat-Test erfüllt und keinen Miller-zeugen zur Basis a besitzt aber auch keine Primzahl ist, heißt starke Pseudoprimzahl.
Die Wahrscheinlichkeit das der Rabin-Miller Primzahltest einem eine falsche Antwort (n ist Prim zur Basis a) gibt ist $\frac{1}{4}$.
Man kann den Test für k unterschiedliche Basen a testen womit die Wahrscheinlichkeit einer falschen Antwort zu $\leq \frac{1}{4^k}$ wird.

5. Sicherheit von Kryptosystemen

Anstatt Sicherheit durch geheimhaltung der Verschlüsselungsverfahren zu erreichen, soll die Sicherheit darauf beruhen, dass es bei bekannten Algorithmen ßchwerïst aus einem gegebenen Chiffretext $C = E_K(M)$ den Klartext M ohne Kenntnis des Schlüssels K zu berechnen. „schwer zu berechnen“soll mit Mitteln der Komplexitätstheorie beschrieben werden.

5.1. Angriff Arte

Ciphertext-only attack: Nur ein oder mehrere Chiffretexte sind bekannt.

Known-plaintext attack: Einige Klartext-Chiffretext-Paare $(M, E_K(M))$ sind bekannt. Ziel: ermitteln des Schlüssels

Chosen-plaintext attack: Der Angreife hat die Möglichkeit, dem Verschlüsseler eine bestimme Klartextnachricht unterzuschieben, die verschlüsselt wird

Chosen-ciphertext attack: Der Angreifer hat die Möglichkeit eine unter einem unbekannten Schlüssel verschlüsselte Nachricht zu wählen und die dazugehörige entschlüsselte Nachricht zu erhalten

Side channel attack: Wird hier nicht betrachtet, aber: Angriff auf eine bestimme physikalische Implementierung eines Kryptosystems, indem das kryptographische Gerät während der Verschlüsselung beobachtet wird

5.2. Kryptographie und die Komplexitätsklasse NP

Bisher wurden Begriffe wie “schwer berechenbar“ oder “praktisch unmöglich“ gesprochen wenn Forderungen an die Sicherheit eines Kryptosystems gestellt wurden. Diese Begriffe sollen mithilfe der Komplexitätstheorie im folgenden Definiert werden.
Zuallererst sollen die **Forderungen** an die Sicherheit eines Kryptosystems definiert werden:
Betrachtung: asymmetrisches Kryptosystem:

Der Schlüsselraum sei: $\mathcal{K} = \mathcal{K}_E \cup \mathcal{K}_D$ (mit $\mathcal{K}_E$ als öffentlichem und $\mathcal{K}_D$ als privatem)

- Die Verschlüsselungsfunktion $E : \mathcal{M} \times \mathcal{K}_E \mapsto \mathcal{C}$ und die entschlüsselungsfunktion $D : \mathcal{C} \times \mathcal{K}_D \mapsto \mathcal{M}$ liegen in FP
- Jede Funktion $D' : \mathcal{C} \times \mathcal{K}_E \mapsto M$ mit $D'_K(E_K(M)) = M$ für alle $M \in \mathcal{M}, K \in K_E$ liegen nicht in FP ! (Ein entschlüsseln ohne privatem Schlüssel soll nicht durch eine polynomielle Funktion möglich sein)

Da die Forderung das D' in FP liegt zu schwach erscheint und Begriffe wie "NP-vollständig" nur für Sprachen definiert ist, soll das Entschlüsselungsproblem als Sprache definiert werden:

- $\mathcal{M}, \mathcal{C} \subseteq \Sigma^*$ für ein geeginetes Alphaet Σ
- $E : \mathcal{M} \times \mathcal{K}_E \mapsto \mathcal{C}$ liegt in FP
- Falls die Verschlüsselung mit dem Public-Key $E(m, e) = c$ gilt, so ist m höchsten polynomiell größer als c: $p(|c|) \geq |m|$ für ein Polynom p. Das heißt die Funktion E verkürzt den Klartext nicht zu stark (polynomiell beschränkt) da sonst nur die Abbildung von $\mathcal{C} \mapsto \mathcal{M}$ beim Entschlüsseln nicht mehr polynomiell wäre (meist ist das chiffrat länger als der Klartext).
- mit diesen Forderungen liegt folgende Sprache in NP:

  $$L_E = \{(m', c, e) | \exists u \in \Sigma^* : E(m'u, e) = c\}$$

  heißt: Es existiert eine Sprache L_E mit dem Triple (m', c, e) (m' :=prefix vom Klartext;c :=chiffretext) mit: es existiert ein u sodass das prefix m' verlängert unter dem Schlüssel e den Chiffretext ergibt.
- die Sprache frägt ob es möglich ist, wenn man den Chiffretext c, den Schlüssel e und ein prefix des KLartextes m' besitzt, den Rest des Chiffretextes brechen kann.
- Beweis das $L_E \in$NP liegt: rate u nichtdeterministisch und überprüfe ob $E(m'u, e) = c$. E ist in polynomieller Zeit berechenbar und aufgrund der polynomiellen Schranke $p(|c)$ von oben arbeitet die nichtdeterministische Turingmaschine in polynomieller Zeit!

Würde L_E in P liegen, dann könnte D' einfach berechnet werden (induktiver beweis mit $m' = \epsilon$ als Anfang (ϵ :=leeres Wort)). Dies gilt ebenfalls umgekehrt: falls D' in FP liegt, dann liegt L_E in P. Folglich ist L_E gleich schwierig wie D', da sich Begriffe wie NP-vollständig nur auf Sprachen beziehen, ist es günstiger mit dem Entscheidungsproblem von L_E weiterzuarbeiten.

Die **Forderung** L_E soll **NP-vollständig** sein, um D' so schwierig wie möglich zu machen, ist **nicht ausreichend**!
Denn:

- der Angreifer kann auch randomisierte Berechnungen durchführen
- Die Funktion D' muss für **jedes Element** schwer zu berechnen sein. Denn die NP-Vollständigkeit von L_E garantiert nur, dass es Instanzen gibt, für die D' schwer zu berechnet ist (worst-case vs. average-case). (wie bei SAT, ist NP-Vollständig, aber nur für relativ wenige Formeln ist SAT wirklich schwer !)

Derzeitiger Kenntnisstand: Als grundlage für Kryptosysteme sind Probleme besser geeignet, von denen nicht bekannt ist, ob sie in P liegen und auch nicht als NP-vollständig bekannt sind: Faktorisierungsproblem, diskreter Logarithmus und Quadratwurzel mod n.
Diese Funktionen werden Falltürfunktionen genannt, wobei nicht bekannt ist ob Falltürfunktionen existieren. Die RSA-Funktion könnte eine Falltürfunktion sein. Das heißt die Verschlüsselungsfunktion ist schwer zu berechnen außer mit dem privaten Schlüssel, dieser stellt eine "Falltür" dar mit deren Hilfe die Verschlüsselungsfunktion doch einfach zu invertieren ist.

Um also die Sicherheit von Kryptosystemen vollständig betrachten zu können muss das potential eines randomisierten Angreifer analysiert und definiert werden um die 'richtigen" Forderungen stellen zu können(siehe Kaptel Mathematische Definitionen - Komplexitätstheorie).

Die 5 Welten: Es gibt eine Betrachtung, in welchen möglichen Welten "wir" leben, je nachdem wie die Beziehung twischen **P** und **NP** sind:

- Algorithmica: Es gilt P = NP
  - Programme müssten nur noch die Eigenschaften der Ausgabe beschreiben, das Berechnugnsverfahren kann automatisch generiert werden (aufgrund von Reduktion)
  - Es gibt **keine** Einweg- oder Falltürfunktionen
  - Alle Kryptosysteme, die mit **beschränkten** Schlüssellängen arbeiten, sind einfach zu brechen. Es gibt nur noch OneTimePad
  - es gibt keine Authentifizierverfahren

- Heuristica: NP-Probleme sind im wort-case schwierig zu lösen, aber fast alle zufällig gewählten Instanzen sind einfach zu Lösen. Schwere Instanzen sind schwer aufzufinden
  - Da schwere Probleme sehr selten sind, können in der Praxis fast alle Aufgabenstellungen (aus NP) effizient gelöst werden
  - Auch in Heuristicia ist Kryptographie im weentlichen nicht möglich, da schwere Instanzen - d.h. schwer zu brechende Chiffretexte- nicht verläßlich produziert (gefunden) werden können
- Pessiland: Es ist einfach, schwere Instanzen von NP-Problemen zu finden, aber es ist nicht möglich, verläßlich schwere und gelöste Instanzen zu erzeugen
  - Es gibt keine Einwegfunktionen, denn Einwegfunktionen sind gerade ein Verfahren, um effizient schwere und gelöste Instanzen zu generieren
  - Ebensowenig ist Authentifizierung möglich
- Minicrypt: schwere und gelöste Instanzen von NP-Problemen können generiert werden
  - Es gibt Einwegfunktionen, aber keine Falltürfunktionen
  - Durch Einwegfunktioen kann man Pseudo-Zufallsgeneratoren erzeugen
  - Symmetrische Verschlüsselung und Authentifizierung sind möglich
  - Asymmetrische Verschlüsselungsverfahren sind nicht möglich
- Cryptomania: zusätzlich zu Minicrypt exisitieren Falltürfunktionen und damit assymetrische Verschlüsselung, Signaturen etc.

6. Kryptographische Protokolle

Im Folgenden geht man davon aus, dass man in Cryptomania lebt. Es existieren Einwegfunktionen also P $\neq$ NP, und es existieren Falltürfunktionen. Somit gibt es Asymmetrische Kryptographie.

6.1. Digitale Unterschriften - Digital Signature Algorithm

Der DSA-Algorithmus bietet nur digitale Signaturen - keine Verschlüsselung.
Es werden Hash-funktionen(genauer aus der sha-familie) benutzt und Signiert und nicht der direkte Klartext. Für den DSA-Algorithmus wird die Definition für die Ordnung eines Elementes $a \in \mathbb{Z}_n^*$ benötigt (siehe 7.4)

6.1.1. Schlüsselerzeugung

- Wähle Längenpaar: $\langle L, N\rangle \in \{\langle 1024, 160\rangle, \langle 2048, 224\rangle, \langle 2048, 256\rangle, \langle 3072, 256\rangle\}$
- Wähle eine N-Bit Primzahl $2^{N-1} < q < 2^N$
- Wähle eine L-Bit Primzahl $2^{L-1} < p < 2^L$ mit $q|(p-1)$
- Wähle ein Element g der Ordnung q von $\mathbb{Z}_p^*$

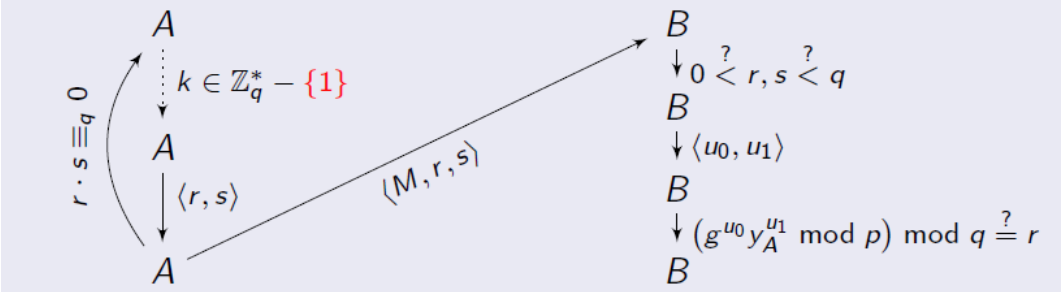
Privater Schlüssel: ein zufallsexponent $x_A \in \{2, ..., q-1\}$

Öffentlicher Schlüssel: $\langle p, q, g, y_A = g^{x_A} \mod p\rangle$

6.1.2. Signatur

Gegeben: Eine Hashfunktion h mit Werten der bitlänge $\geq N$

Zusammenfassung: Alice versieht Nachricht M für Bob mit einer Signatur $\langle r, s\rangle \in G \times \mathbb{Z}_q$, die dieser verifiziert:



$r := (g^k \bmod p) \bmod q$ $\quad$ $u_0 := s^{-1}\rho_N(h(M)) \bmod q$

$s := (\rho_N(h(M)) + x_A r)\, k^{-1} \bmod q$ $\quad$ $u_1 := s^{-1} r \bmod q$

- Dabei extrahiert ρ_N die signifikantesten N Bits aus dem Argument.
- y_A ist der öffentliche Schlüssel
- k darf nicht 1 sein, da sonst x_A bestimmt werden kann!

6.1.3. Korrektheit

Beweis: Löse s nach k auf:

$$k = (\rho_N(h(M)) + x_A \cdot r)s^{-1} \mod q$$

s^{-1} in die Klammer multiplizieren

$$k = \underbrace{\rho_N(h(M)) \cdot s^{-1}}_{=u_0} + x_A \cdot \underbrace{r \cdot s^{-1}}_{=u_1} \mod q$$

$$k = u_0 + x_A \cdot u_1 \mod q$$

da $\mathrm{ord}(g) = q$ gilt (der exponent ist $\mod q$!):

$$g^k \mod p = g^{u_0 + x_A \cdot u_1} \mod p$$

$$g^k \mod p = g^{u_0} \cdot g^{x_A \cdot u_1} \mod p$$

der Öffentliche Schlüssel ist $y_A = g^{x_A} \mod p$:

$$g^k \mod p = g^{u_0} \cdot y_A^{u_1} \mod p$$

die Gleichung $\mod q$ nehmen:

$$\underbrace{g^k \mod p \mod q}_{=r} = g^{u_0} \cdot y_A^{u_1} \mod p \mod q$$

Das ist genau das was im letzten Schritt überprüft wird □

6.1.4. Sicherheitsüberlegung

Die Bereichsprüfung $0 \overset{?}{<} r, s \overset{?}{<} q$ ist nötig, da sonst ein Angreifer beliebige Signaturen fälschen kann.
Erhält ein Angreifer einen Signierten Text $\langle M, r, s\rangle$ und $\rho_N(h(M)) \in \mathbb{Z}_q^*$ dann gilt für die Fälle:

$r \equiv_q 0$: Der Angreifer möchte nun den eigenen Text M' mit $\rho_N(h(M')) := m'$ signieren. Sei $\rho_N(h(M)) := m$. Dann erzeugt der Angreifer ein neues s' mit:

$$s' = m' \cdot m^{-1} \cdot s \mod q$$

Dann ist (M', r, s') eine gültige Signatur. Betrachte die Werte die in die letzte Prüfung einfließen:

$$u_0' + x_A \cdot u_1' \equiv_q s'^{-1} m' + x_A \cdot s'^{-1} \cdot r$$

da $r \mod q = 0$ (weil Bereichsprüfung vergessen) ist der hintere Termin = 0. Daher ist der Term gleich egal ob s'^{-1} oder s^{-1} enthalten ist.

$$\equiv_q s'^{-1} m' + \underbrace{x_A \cdot s^{-1} \cdot r}_{\equiv_q 0}$$

Ersetzte im vorderen Term s' für s'^{-1} ein:

$$\equiv_q \underbrace{m'^{-1} \cdot m \cdot s^{-1}}_{s'^{-1}} \cdot m' + x_A \cdot s^{-1} \cdot r$$

m'^{-1} und m' heben sich auf:

$$\equiv_q s^{-1} \cdot m + x_A s^{-1} \cdot r \Rightarrow u_0 + x_A u_1$$

Also gilt: $g^{u'_0} y_A^{u'_1} \equiv_q = r$ □.
$\Longrightarrow$ der Angreifer kann also einen Text in Alice namen Signieren. Bob sollte einer Signatur mit $r = 0$ nicht trauen !

$s \equiv_q = 0$: Dann kann der Angreifer den privaten Schlüssel errechnen (sei m wieder: $\rho_N(h(M)) := m$):

Für $s \equiv_q = 0$ gilt:

$$s = (m + x_A \cdot r)k^{-1} \mod q \equiv_q 0$$

Also einer der beiden Faktoren muss 0 sein (da q Prim + mod q). $k \neq 0$ weil es aus $\mathbb{Z}_q^*$

$$\Leftrightarrow m + x_A \cdot r \equiv_q = 0$$

umstellen nach x_A um den geheimen exponenten zu erhalten:

$$x_A \equiv_q -m \cdot r^{-1}$$

und damit erhält der Angreifer den geheimen Schlüssel

Mögliche Angriffe:
Gegebenen Klartext M signieren ohne geheimen Exponenten zu kennen:
die letzte Überprüfung umgeformt schaut wie folgt aus:

$$g^m \cdot y_A^r \equiv_q r^s \qquad \text{wobei:} m = \rho_N(h(M))$$

r vorgegeben: passendes s berechnen mit:

$$\log_r(g^m \cdot y^r) \equiv_q s$$

s vorgegeben: passendes r berechnen mit:

$$g^{ms^{-1}} \cdot y_A^{s^{-1}r} \equiv_q = r$$

hierfür ist kein effizienter Algorithmus bekannt

- Ob r, s gemeinsam bestimmt werden können ist ebenfalls unklar

Schwächerer Angriff: Nachricht M zu gegebener Signatur $\langle r, s \rangle$ finden:

- $m = \log_g(r^s \cdot y_A^{-r})$
- eine Klartextnachricht M' mit $m := \rho_N(h(M)) \stackrel{!}{=} \rho_N(h(M'))$. Also eine kollision in der Hashfunktion erzeugen.

6.2. Schlüsselaustausch Protokoll

Wird der bisher betrachtete Fall des Schlüsseltausch betrachtet: also es wird ein Schlüssel K für ein symmetrisches Verfahren gewählt und mittels public-key-Verfahren über eine unsichere Leitung übertragen, fallen einem zwei Nachteile auf:

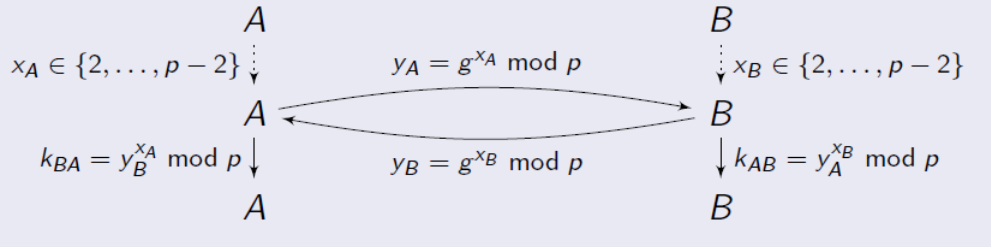
- die Wahl des symmetrischen Sitzungsschlüssels obliegt nur einem Teilnehmer
- der Schlüssel wird über das Netz geschickt (wenn auch verschlüsselt)

Daher im Folgenden Protkolle die diese beiden Nachteile adressieren:

6.2.1. Diffie-Hellman-Verfahren

Das Diffie-Hellman-Verfahren ist ein Verfahren zum direkten Schlüsselaustausch. Das Verfahren kann mittels Man-In-The-Middle-Angriff korrumpiert werden, das Station-to-Station Protokoll, behebt diesen Fehler, baut jedoch maßgeblich auf dem Diffi-Hellman-Verfahren auf, daher wird es im folgenden behandelt:

Gegeben: Primzahl p und Primitivwurzel $g \mod p$, bekannt
Zusammenfassung: Alice und Bob vereinbaren einen gemeinsamen Schlüssel aus $\mathbb{Z}_p*$:



1. jeder wählt einen geheimen Exponenten x_A und x_B zufällig
2. jeder berechnet für sich Erzeuger g hoch Exponent, bsp für Alice: $y_A = g^{x_A} \mod p$
3. der gemeinsame Schlüssel ist nun: $k_{BA} = k_{AB} = g^{x_A \cdot x_B} \mod p$

$\Rightarrow$ der Schlüssel wird **nicht** über die Leitung übertragen, nur ein teil-Schlüssel

- Zwecks Geheimhaltung des Schlüssels sind die Werte 0, 1 und p-1 als Exponenten auszuschließen
- Das Verfahren beruht auf dem Diskreten Logarithmus, ist dieser eine Einwegfunktion, so können die geheimen Exponenten nicht bestimmt werden durch abhören der $y_{A/B}$

Schwachstelle: Die Authentizität der Teilnehmer ist nicht gewährleistet, wodurch ein Man-In-The-Middel-Angriff möglich wird: Der Angreifer unterbricht die Verbindung und vereinbart mit beiden Parteien je einen gemeinsamen Schlüssel, wobei er sich jeweils als der andere Partner ausgibt. Dann kann er die geheime Kommunikation belauschen.

Abhilfe:

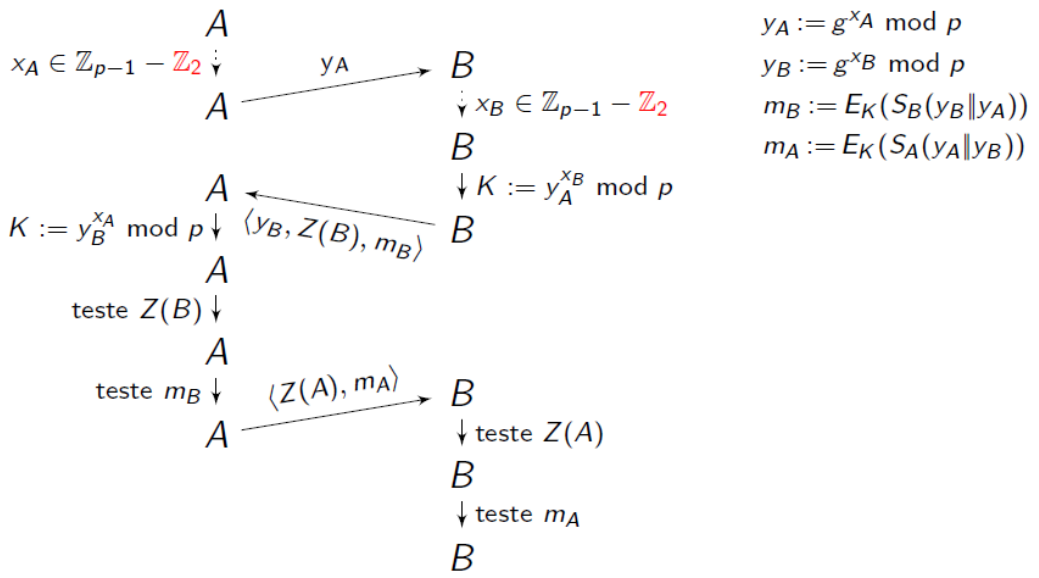
- Alice und Bob müssen zusätzlich auch sicher sein, dass beide mit demselben Schlüssel arbeiten (Key confirmation)

6.2.2. Station-To-Station Protokoll

Das Ziel des Station-To-Station Protokolls ist es das Diffie-Hellman Verfahren um eine Authentifizierung der Teilnehmer und eine key confirmation zu erweitern. Dafür verliert das Verfahren seine Parallelität und wird Seriell.
Voraussetzungen:

- ein symmetrisches Tiel-Kryptosystem $\langle E, D\rangle$
- öffentliche und private RSA-Schlüssel $\langle n_x, e_x\rangle$ bzw. d_x pro Teilnehmer X
- eine Hashfunktion h mit Werten $< n_x$
- ein Public-Key Kryptosystem $\langle E^{TA}, D^{DA}\rangle$ der TA
- Zertifikat $Z(X)$ für alle Nutzer
- RSA-Signaturen der Form $S_X(M) := (h(M))^{d_x} \mod n_x$
- eine Primzahl p und eine primitive Wurzel $g \mod p$

1. Alice wählt geheimen Exponenten und übertragt y_A
2. Bob wählt geheimen Exponenten und erzeugt den Schlüssel K
3. Bob sendet Alice y_B und seine Signatur $Z(B)$. Zusätzlich verschlüsselt Bob den signierten Wert der Hashfunktion mit dem schlüssel k (für die key confirmation)



4. Alice berechnet den Schlüssel k und testet $Z(B)$ um sicher zu stellen, dass sie mit Bob kommuniziert. Alice testet m_B um sicher zu stellen, dass beide den selben Schlüssel haben.
5. Alice sendet an Bob $Z(A)$ und ebenfalls den signierten Wert der Hashfunktion verschlüsselt mit k
6. Bob testet das Zertifikat von Alice und überprüft ob beide den selben Schlüssel benutzten

6.2.3. Selbst-zertifizierende Schlüssel

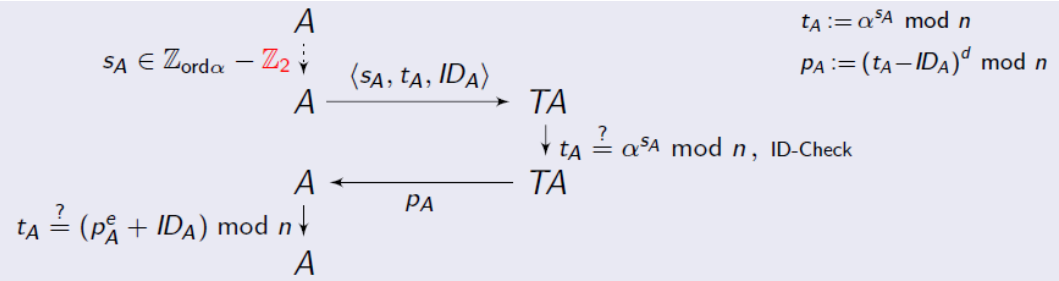
Statt mittels eines Zertifikats $Z(X)$ lässt sich die Schlüsselkomponente t_x auch anders authentifizieren.

Erhalt des selbst-zertifizierenden Schlüssels

Gegeben:

- öffentlicher und privater RSA-Schlüssel $\langle n, e \rangle$ und d der TA (mit $n := p \cdot q, p, q \in$ prim)
- ein öffentliche bekanntes Element $\alpha \in \mathbb{Z}_n^*$ der maximalen Ordnung kgV(q-1,p-1)

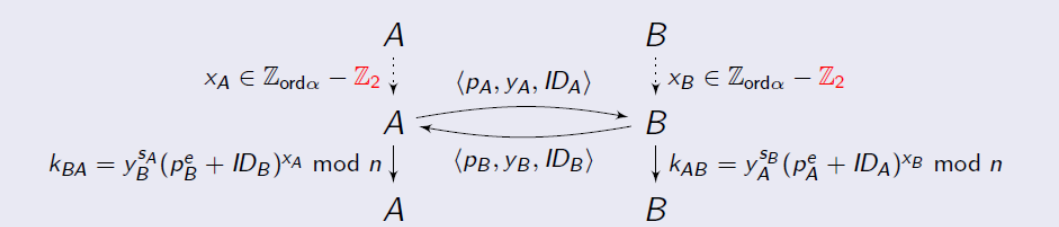
Alice erhält von der TA einen selbst-zertifizierenden Schlüssel p_A (ID_A ist die ID von Alice (bspw. Personalausweisnummer o.ä.) wodurch die TA die Identität von Alice überprüfen kann):



1. Alice wählt ein $s_A \in \mathbb{Z}_{\text{ord}(\alpha)}$ und schickt, s_A, den wert t_A und ihre ID_A an die TA.
2. Die TA überprüft die ID von Alice und ob das t_A auch durch die exponentation α^{s_A} entanden ist.
3. die TA berechnet den selbst-zertifizierenden Schlüssel p_A und schickt ihn (über einen sicheren Übertragungsweg!) an Alice.
4. Alice überprüft den erhaltenen Schlüssel p_A

Schlüsselaustausch nach Girault - mit selbst-zertifizierendem Schlüssel

Das Protokoll ist ähnlich dem Diffie-Hellman Verfahren:



wobei $y_A := \alpha^{x_A} \bmod n$ und $y_B := \alpha^{x_B} \bmod n$.

1. Alice und Bob wählen gleichzeitig ihren geheimen Exponenten x_A bzw. $x_B \in \mathbb{Z}_{\text{ord}(\alpha)}$
2. Beide übertragen sich gegenseitig, ihren selbst-zertifizierenden Schlüssel $p_{A/B}$,ihre berechneten $y_{A/B}$ und ihre ID $ID_{A/B}$
3. Beide können nur den gemeinsamen Schlüssel k berechnet. Dieser ist nun: $k = k_{AB} = k_{BA} = \alpha^{s_a x_B + s_b x_a} \bmod n$

- Korrektheit kann einfach durch einsetzen in k_{BA} und k_{AB} gezeigt werden!

Bemerkung:

- Weder p_A noch t_A oder ID_A sind von der TA signiert !

- Die Authentizität des selbst-zertifizierenden Schlüssels p_A ist nicht direkt verifizierbar. Der selbst-zertifizierende Schlüssel ist **nur** in diesem Protokoll verwendbar und "selbst-zertifizierend" ein Angreifer kann nicht im Namen von Alice einen gemeinsamen Schlüssel vereinbaren. Dies würde im Rahmen dieses Protokolls fehlschlagen. Bei der Korrektheit p_A gegen ein anderes $p_\square$ austauschen und es ergibt sich, dass der Schlüssel nicht mehr stimmt!
- die Übertragung von s_A beim erhalten des selbst-zertifizierenden Schlüssels an die TA ist **zwingend** erforderlich, da ansonsten ein Angreifer sich einen selbst-zertifizierenden Schlüssel für Alice besorgen kann ! Die TA **muss** überprüfen können ob t_A nur durch exponentation mit s_A entstanden ist (sonst kann der Angreifer dort "mehr Informationen" reinpacken, sodass sie am ende einen sz-Schlüssel für Alice erhält.)
  - Angreiferin Eve wählt s'_A zufällig (wie Alice auch)
  - Eve sendet im Namen von Alice $t'_U := (\alpha^{s'_A} - ID_A + ID_U) \mod n$ an die TA
  - die TA berechnet ein p'_U für Eve wie vorgesehen (ohne Test zu Testen ob t'_U korrekt erzeugt wurde):

    $$p'_U = (\underbrace{\alpha^{s'_U} - ID_A + ID_U}_{t'_U} - ID_U)^d \mod n = (\alpha^{s'_U} - ID_A)^d \mod n = p'_A$$

    zurück.

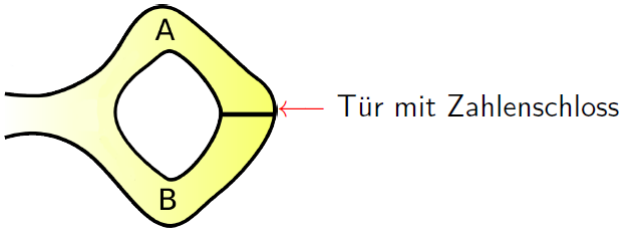
  $\Longrightarrow$ p'_A ist ein korrekt konstruierter selbst-zertifizerender Schlüssel für Alice und im Besitzt von Eve !

6.3. Zero Knowledge Protokoll

Identifizierungs- und Authentifizierungsverfahren dienen dazu, dass Teilnehmer in einem Netzwerk gegenüber anderen eindeutig ihre Identität nachweisen können.
Dies geschieht zumeist dadurch, dass sie nachweisen, ein Geheimnis zu kennen, das nur diesem bestimmten Teilnehmer bekannt sein sollte.
Das Identifizierungsverfahren hat zwei Kommunikationspartner: einen **Verifier**("Vector") und einen **Prover** ("Peggy").
Es soll garantiert sein, dass Peggy außer ihrer Identität Victor gegenüber keine weiteren Informationen verrät.
Informelle eigenschaft von Zero-Knowledge-Protkollen:

Peggy überzeugt Victor, ein Geheimnis zu kennen, ohne dieses Geheimnis selbst du verraten

Ein Gedankenexperiment skizziert die geforderte Eigenschaft: Der Prover kennt den



Zahlencode der Tür in der Höhle. Der Prover will den Verifier überzeugen, dass er den Zahlencode kennt, ohne diesen zu verraten. Geht der Prover nun an der Weggabelung in Gang A, während der Verifier an der Weggabelung steht, und kommt am Gang B wieder herraus, so hat der Prover dem Verifier gezeigt, das er den Code kennt ohne diesen zu verraten.
Ein Zero-Knowledge-Beweis muss drei Kriterien erfüllen:

Vollständigkeit: falls der Prover das Geheimnis kennt und nicht betrügt, ist am Ende des Protokolls der ehrliche Verifizierer von der Kenntnis des Provers überzeugt.

Korretheit: falls der Prover das Geheimnis nicht kennt, dann ist die Wahrscheinlichkeit vernachlässigbar, dass der Verifier den Beweis akzeptiert (z.B. nach 30 Durchläufen: $\frac{1}{2^{30}}$)

Zero-Knowledge-Eigenschaft: falls der Prover das Geheimnis kennt, lernt ein betrügender Verifizierer nichts darüber. Warum lernt der Verifier nichts über das Geheimnis?:

- Angenommen ein Kamerateam filmt die 30 Durchläufe des Protokolls $\rightsquigarrow$ ein **Transcript** des Protokolls
- der Verifier zeigt einem dritten (Charles) das Transcript, um ihn zu überzeugen das der Prover in dem Transcript das Geheimnis kennt. Ist charles davon überzeugt, dass der Prover das Geheimnis kennt?

  Nein!: denn der Verifier und der Prover könnten das Transcript gefälscht haben, sie verabreden sich (Drehbuch) oder "schneiden" den Film.

– ein echtes Transcript ist von einem gefälschten Transcript ununterscheidbar

⇒ der Verifier gewinnt keine Information aus dem Protokolllauf

Beispiel Protokoll - Challenge-Response-Schema

1. Victor wählt eine Zufallszahl (einen Nonce) r, berechnet den Hashwert $x = h(r)$ und den Challenge $c = E_P(r)$ und sendet $(ID(V), c, x)$ an Peggy.
2. Peggy beschließt, sich Victor gegenüber zu identifizieren: Sie berechnet $r = D_P(c)$ und prüft ob $x \stackrel{?}{=} h(r)$. Wenn die Prüfung erfolgreich, schickt sie r an Victor
3. Victor akzeptiert den Beweis von Peggys Identität, falls er den korrekten Wert r zurückerhält.

Bemerkungen:

- Peggy entschlüsselt den Challenge c und weist damit nach, dass sie ihren privaten Schlüssel kennt.
- Eine Zufallszahl r (statt einer festen Zahl) wird verwendet, damit kein Replay-Angriff durchgeführt werden kann (ansonsten könnten Informationen gewonnen werden)
- Durch die Prüfung des Hashwertes x weiß Peggy, dass Victor den Wert r bereits kennt, sie liefert ihm keine zusätzlichen Informationen und keine bisher unbekannte Signatur (ohne die Prüfung könnte r entweder unverschlüsselt und somit danach Signiert sein oder r war verschlüsselt und Victor lässt peggy r entschlüsseln um den Inhalt zu erfahren)

Vorteile gegenüber Passwort-basierter Identifizierung: es muss kein Passwort ausgetauscht werden und ein Angreifer, der die Kommunikation abhört, kann sich anschließend **nicht** als Peggy ausgeben

6.4. Interaktive Beweissysteme

Um die oben Informell aufgeführten Kriterien zu formalisieren wird ein Interaktives Beweissystem benötigt, erst einmal ohne die Zero-Knowledge-Eigenschaft:
Idee:

- Peggy und Victor starten mit Eingabe $x \in \Sigma^*$

- Peggy und Victor rechnen abwechselnd (Victor beginnt)
- am Ende eines Berechnungs-"Zuges" senden Peggy/Victor dem Kommunikationspartner eine Nachricht
- Victor entscheidet am Ende ob x akzeptiert wird

Bemerkungen: Die Nachrichten folge ist $m_1, m_2, ..., m_{2l}$. m_1 bezieht sich nur auf die Eingabe $V(x)$. Im Allgemeinen dürfen beide die Eingabe und alle vorherigen Nachrichten benutzten! **Beschränkungen:**

- Victor ist eine polynomzeitbeschränkte probabilistische TM
- Peggy ist eine TM mit **unbeschränkten Resourcen**, unsbesondere darf sie nicht-deterministisch arbeiten, beliebig viel Zeit verbrauchen und Zufallsbits verwenden
- Eingabe steht auf gemeinsamen **read-only Band**
- Die TMs Peggy und Victor haben zwei gemeinsame Bänder zum Nachrichtenaustausch:

P $\rightarrow$ V Band: **write-only** für Peggy; **read-only** für Victor

V $\rightarrow$ P Band: **write-only** für Victor; **read-only** für Peggy

- es werden aber nur polynomial viele Nachrichten ausgetauscht, die auch nur polynomiale Länge haben (jeweils polynomial in $|x|$)

6.4.1. Definition (Interaktives Beweissystem für eine Sprache)

Eine Sprache L hat ein interaktives Beweissystem, wenn es ein - wie oben beschriebenes - interaktives Beweissystem mit Verifier V und Prover P, wobei gilt:

Vollständigkeit: $x \in L \Rightarrow P(\text{“}(V,P)\text{akzeptiert}x\text{“}) \geq \frac{3}{4}$

Korrektheit: $x \notin L \Rightarrow P(\text{“}(V,P')\text{akzeptiert x“}) \leq \frac{1}{4}$
für jeden (potentiell bösartigen) Prover P'

Bemerkungen:

- Intuitiv ist P der gutartige Prover, der in der Lage sein muss, den Verifier V mit hoher Wahrscheinlichkeit zu überzeugen
- ein bösartiger Prover P' - der V evtl. tuschen will - hat nur eine niedrige Wahrscheinlichkeit, mit der er V überzeugen kann

- Wie bei den probabilistischen Komplexitätsklassen kann man die Wahrscheinlichkeit für Akzeptanz bzw. fehlerhafte Akzeptanz durch mehrfach Ausführung des Protokolls erhöhen bzw. vermindern
- **IP** ist die Menge aller Sprachen, die ein interaktives Beweissystem haben. Es gilt NP $\subseteq$ IP und BPP $\subseteq$ IP.
- es gilt weiterhin IP = PSPACE[12] . Obwohl eine Partei im interaktiven Beweissystem unbeschränkte Ressourcen hat, befindet es sich im Deterministischenraum. Das schränkt die Mächtigkeit von Pegy ein ! Informell kommt das daher, dass Peggy zwar alles berechnet, aber kann durch die Einschränkung auf polynomiell viele Nachrichten, die polynomiell in der Länge beschränkt sind, nur beschränkt viel Victor mitteilen, und dieser muss am Ende überzeugt sein / Entscheiden !

6.4.2. Definition - Interaktives Beweissystem mit perfekter Zero-Knowledge-Eigenschaft

Zusätzlich zur Vollständigkeit und Korrektheit wird zusätzlich gefordert:

Das interaktive Beweissystem (P, V) heißt ein perfektes Zero-Knowledge-Beweissystem für die Sprache L, wenn es für jeden - möglichweise bösartigen - Verifier V^* eine probabilistische polynomzeitbeschränkte TM M^* (Simulator genannt) gibt, sodass für jede Eingabe $x \in L$ die folgenden zwei Zufallsvariablen identische Wahrscheinlichkeitsverteilung haben:

- die Transkripte von (P, V^*) auf x und
- die Ausgabe von M^* auf x

Das heißt ein Beobachter kann die Menge der echten Transkripte von (P, V^*) nicht von gefälschten Transkripten von M^* unterscheiden ! Das ist die Modellierung der Forderung, dass keine Informationen übertragen werden

- Die Zero-knowledge-eigenschaft wird für jeden Verifier V^* gefordert und nicht nur für den ehrlichen V. Denn nicht nur der gutartige Verifer V, der sich an das Protokoll hält, sondern auch ein bösartiger Verifier kann nicht in dem Besitz von zusätzlichem Wissen kommen

[12] Also die Probleme die zwei Maschinen durch diese Kommunikation-art und weise lösen können, sind genau die Probleme die determinstische Turingmaschinen im Polynomiellenraum entschieden werden können

Variante: Computational Zero-Knowledge: Die Forderung, dass die Wahrscheinlichkeitsverteilungen gar nicht voneinander unterschieden werden können, ist sehr stark ist. Daher gibt es eine Variante welche sagt:

> die Wahrscheinlichkeitsverteilungen von $(P, V*)$ und M^* auf x können von keiner probabilistischen Polyzeit-TM(das was wir einem Angreifer zugestehen) voneinander unterschieden werden.

6.4.3. Definition - Perfekte Zero-Knowledge Eigenschaft für Verifier

Es gibt noch eine Einschränkung, da einige Protokolle nur die folgende definition und nicht die obige erfüllen:

> Das interaktive Beweissystem (P, V) für L heißt ein perfektes Zero-Knowledge Beweissystem für V, wenn es eine probabilistische polynomzeitbeschränkte TM M gibt, sodass für jede Eingabe $x \in L$ die folgenden zwei Zufallsvariablen identische Wahrscheinlichkeitsverteilung habe:
>
> - die Transkripte von (P, V) auf x und
> - die Ausgabe von M auf x
>
> Die Forderung dieselbe wie bei der obigen Definition nur dass hier der bösartige Verifier V^* entfernt wurde ! Es gilt folglich nur für einen **ehrlichen Verifier**!

6.4.4. Beispiel Interaktives Beweissystem auf NON-/GRAPHISO

Ein Interaktives Beweissystem für Nicht-Graphisomorphie(unbekannt ob in NP bzw. in BPP):
NON-GRAPHISO:

Eingabe: ein Paar (G_1, G_2) ungerichteter Graphen $G_i, i = \{1, 2\}$

Ausgabe: Antwort auf Frage: Sind die Graphen G_1, G_2 nicht isomorph?

Interaktives Beweissystem:

Gegeben: G_1, G_2 Graphen, beide mit Knotenmenge $\{1, ..., n\}$

1. Victor wählt eine Zufallszahl $i \in \{1, 2\}$ und eine zufällige Permutation der Kanten: $\pi : \{1, ..., n\} \mapsto \{1, ..., n\}$
2. Victor benennt die Knoten G_i mit Hilfe von π um und erhält dadurch $H = \pi(G_i)$. Er schickt H an Peggy und fragt damit implizit nach Index i mit $G_i \cong H$

3. Peggy antwortet mit einem Index j. Peggy kann nicht-deterministisch die isomorphie testen
4. Victor akzeptiert, falls $i = j$

Bemerkungen:

- Falls G_1 und G_2 nicht isomorph sind: Peggy berechnet mit ihren unbeschränkten Resourcen, ob H isomorph zu G_1 oder zu G_2 und gibt eine korrekte Antwort. Victor akzeptiert mit Wahrscheinlichkeit $= 1$.
  $\rightsquigarrow$ **Vollständigkeit**
- Falls G_1 und G_2 isomorph sind: Peggy kann nur raten (da alle drei isomorph sind, kann Peggy nicht wissen welchen Index Victor gewählt hat) und gibt nur in der Hälfte der Fälle eine korrekte Antwort. Peggy hat nur eine Wahrscheinlichkeit von $\frac{1}{2}$ Victor zur Akzeptanz zu bringen/zwingen (wenn böser profer).
  $\rightsquigarrow$ **Korrektheit**
- Nach zwei Durchläufen des Protokolls hat Peggy nur noch eine Wahrscheinlichkeit von $\frac{1}{4}$ Victor zur Akzeptanz zu bringen/zwingen(wenn böser prover). Somit ist dies ein interaktives Beweissystem für NON-GRAPHISO
- Das interaktive Beweissystem für NON-GRAPHISO besitzt **keine** Zero-Knowledge-Eigenschaft da:
  - Wenn $G_1 \ncong G_2$

  Annahme: V^* kennt $\pi(G_1)$ und $\pi'(G_2)$, aber **nicht** π und **nicht** welcher Graph isomorph zu G_1 bzw. G_2 ist. (Also er hat 4 Graphen und weiß nicht wie diese isomorph zueinander sind)

  $\rightarrow$ dann erhält V^* diese Information indem er einen der Graphen $\pi(G_1$ oder $\pi'(G_2)$ an P sendet und von P als Antwort $i = 1$ oder 2 erhält. Somit weiß er wie diese 4 Graphen zueinander stehen und somit hat er mehr Informationen als vor dem Protokoll.

$\Rightarrow$ somit gilt die Zero-Knowledge-Eigenschaft für ein allgemeines V^* **nicht !**

GRAPHISO: Ein Interaktives Beweissystem für Graphisomorphie(in NP, unbekannt of NP-Vollständig). Das Komplement von NON-GRAPHISO:
Interaktives Beweissystem:

Gegeben: G_1, G_2 Graphen, beide mit Knotenmenge $\{1, ..., n\}$

1. Peggy wählt eine zufällige Permutation: $\pi : \{1, ..., n\} \mapsto \{1, ..., n\}$.
   Sie wendet π auf G_1 an und erhält $H = \pi(G_1)$ und schickt H an Victor
2. Victor wählt eine Zufallszahl $i \in \{1, 2\}$ und schickt diese an Peggy

3. Peggy berechnet die Permutation σ mit $H = \sigma(G_i)$ und schickt σ an Victor
4. Victor akzeptiert, falls $\sigma(G_i) = H$

Bemerkungen:

- **Vollständigkeit:** Falls $G_1 \cong G_2$ gilt, akzeptiert Victor und Peggy gemeinsam mit Wahrscheinlichkeit 1, denn Peggy findet immer eine entsprechende Permunation σ
- **Korrektheit:** Falls $G_1 \ncong G_2$, dann mus Peggy mit Wahrschienlichkeit $\frac{1}{2}$ mit einer falschen Permutation σ antworten. Wird das Protokoll zweimal ausgeführt, sinkt diese Wahrschienlichkeit wieder.

$\Rightarrow$ somit ist das gegebene System ein Interaktives Beweissystem für GRAPHISO !

- **Zero-Knowledge-Eigenschaft für Victor:** Beweis: Sei $G_1 \cong G_2$
  - Protokolltranskripte: die Transkripte sind folgende Triple: (H, i, σ) mit der Eigenschaften, dass $H = \pi(G_1)$ für ein π ; $i \in \{1, 2\}$; und außerdem $H = \sigma(G_i)$.
  - Jetzt brauchen wir einen Algorithmus der diese Transkripte fälscht mit den selben W'keitverteilungen

  Simulator M (für Victor:) der Simulator muss sich nicht an die Reihenfolge halten, deshalb

  1. wählt er zuerst das Bit $i \in \{1, 2\}$ zufällig und danach σ (unabhängig voneinander)
  2. danach berechne $H = \sigma(G_i)$
  3. schreibe Triple (H, i, σ)

  - Korrektheit: $H \cong G_i \cong G_1$ also existiert π wie oben. Die anderen bedingungen sind erfüllt, denn i und σ sind passend gewählt
  - Laufzeit: eine Permutation zu wählen, zu berechnen und ein bit zu wählen gescheit in polynomieller Laufzeit $\rightarrow$ ok
  - Verteilung: Die Wahrscheinlichkeitsverteilung von gefälschten und echten Transkripten ist gleich verteilt, denn die Zufalls Wahl von i und σ im Protokoll und im Fälschungsalgorithmus sind gleich.
  - der Simulator muss 2 Transkripe erzeugen,da das Protokoll zweimal durchlaufen muss, damit es ein interaktive Beweissystem ist

- $\rightsquigarrow$: Damit liefert das Protokoll ein perfektes Zero-Knowledge-Beweissystem für Victor
- Fazit: Falls GRAPHSIO $\notin$ BPP, dann gewinnt der Verifier aus dem Beweissystem keine Informationen über den Isomorphismus zwischen zwei gegebenen Graphen
- um ein perfektes Zero-Knowledge-Beweissystem zu erhalten, muss der Beweis noch für jeden V^* durchgeführt werden.
- Auch im Falle ines bösartigen Verifiers kann jeder Ablauf durch eine probabilistische Polynomzeit-TM simuliert werden
- Insbesondere folgt aus der Tatsache, dass es sich um ein Zero-Knowledge-Protokoll handelt, dass der Verifier nicht in den Besitzt von Informationen kommen kann, die er nicht selbst in probabilistischer Polynomzeit berechnen könnte !
- Jede Spache $L \in$ BPP hat ein Zero-Knowledge-Beweissystem

6.4.5. Interaktives Beweissytem für NP

Falls bijektive Einwegfunktionen existieren, dann besitzt jedes $L \in$NP ein Zero-Knowledge-Beweissystem (im Sinne von Computational Zero-Knowledge)
Dafür werden Hilfsprotkolle benötigt:

6.4.5.1. Bit-Commitment

Bit-Commitment soll folgende Problemstellung lösen:

> Bob will, dass Alice sich auf den Wert eines Bits $b \in \{0, 1\}$ festlegt. Alice will aber, dass Bob erst **später** von ihrer Wahl erfährt. Wie kann Bob erreichen, dass sich Alice jetzt auf ein Bit festlegt und später - wenn sie zusätzliche Informationen besitzt - nicht bezüglich ihrer Festlegung lügen kann?

Voraussetzung:

- Eine bijektive Einwegfunktion f
- ein Hardcore-Prädikat B $\in$ FP zu f

Definition - Hardcore-Prädikat zu f

Ein Prädikat $b : \{0,1\}^* \mapsto \{0,1\}$[13] heißt hard-core zur Funktion $f\{0,1\}^* \mapsto \{0,1\}^*$ wenn:

- $b \in$FP gilt und
- für jede probabilistische TM M, jedes Polynom $p(n)$ und jedes hinreichend große n gilt für ein zufällig gewähltes $x \in \{0,1\}^n$:

  $$\mathrm{P}(M \text{ berechnet } b(x) \text{ auf Eingabe } f(x)) \leq \frac{1}{2} + \frac{1}{p(n)}$$

  heißt das M $b(x)$ auf Eingabe von $f(x)$ nur zu einer Wahrscheinlichkeit von $\frac{1}{2}$ (rest vernachlässigbar) berechnen kann.

$\rightsquigarrow$ Es ist einfach (in FP) $b(x)$ zu berechnen wenn man x hat und schwer wenn man nur $f(x)$ hat(eine probabilistische TM hat eine W'keit von unter $\frac{1}{2}$).

Kandidate für Hardcore-Prädikate:

- Bei der diskreten Exponentialfunktion: höchstwertiges Bit ($most(x)$)
- bei der RSA-Falltürfunktion: niedrigstwertiges Bit

Beobachtung: Eine injektive Funktion mit Hardcore-Prädikat ist eine Einwegfunktion. Außerdem existieren Hardcore-Prädikate, falls Einwegfunktionen existieren !

Bit Commitment - Protokoll

Gegeben Einwegfunktion f und ein dazugehöriges Hardcore-Prädikat B:

1. Alice wählt ihr Bit $b \in \{0,1\}$. Sie sucht ein $x \in \Sigma^*$ mit $B(x) = b$ und schickt $C = f(x)$ an Bob. (wählt man als Grundlage den diskreten logarithmus und als Hardcore-Prädikat most(x), so müsste Bob jetzt ein diskretes Logarithmusproblem Lösen um x herauszubekommen !)
2. Bob empfängt C, kann aus C allerdings keine Informationen über b erhalten.
3. Später (nachdem das Protokoll in dem Bit Commitment verwendet wird, weitergelaufen ist) kann Alice Bob gegenüber nachweisen, dass sie sich auf b festgelagt hat, indem sie x an Bob schickt.
4. Bob überprüft dann $C \stackrel{?}{=} f(x)$ und berechnet $B(x)$

Bemerkungen:

[13] b bildet also einen Bit-string auf ein Bit ab

- Es ist wichtig, dass f eine bijektive Einwegfunktion (sog. Einweg-Permutation) ist. Ansonsten hat $f(x)$ evtl. mehrere Urbilder, von denen sich Alice später eines aussuchen kann.
- Die Festlegung C werden in Zukunft mit $C(b)$ notiert
- Wie lang soll x sein? Eigentlich sollte die Sicherheit eines Protokolls in der Länge der Eingabe gemessen werden, hier gibt es jedoch keine Eingabe. Daher nimmt man als Ersatz einen Sicherheitsparameter k als zusätzliche Eingabe, k bestimmt die Bitlänge von x. In Bezug auf k muss die Erfolgswahrscheinlichkeit für den Angreifer vernachlässigbar sein

Beispiel für Bit-Commitment

Bit-Commitment kann bei ienem Münzwurf über das Telefon eingesetzt werden:
Alice wählt ein Bit $a \in \{0,1\}$ und schickt eine Festlegung $C(a)$ an Bob. Bob kennt a nicht und wählt sein bit $b \in \{0,1\}$, welches er an Alice schickt. alice weist die Wahl von a nach und gibt das Ergebnis $a \oplus b$ bekannt.
Einwegfunktion und Hardcore-Prädikat: Um dies umzusetzen wird als Einwegfunktion die diskrete Exponentialfunktion gewählt mit der Primzahl $p = 19$ und der Primitivwurzel $g = 3$ und als Hardcore-Prädikat $most(x)$:

$$most(x) = \begin{cases} 0 & \text{falls } x \leq \frac{p-1}{2} \\ 1 & \text{sonst} \end{cases}$$

Ablauf:

1. Alice wählt zufällig das Bit $a = 1$ und wählt $x = 14$ als Zeugen($14 \geq \frac{19-1}{2}$). Sieschickt $f(x) = g^x \mod p = 3^{14} \mod 19 = 4$ an Bob
2. Bob empfängt $f(x) = 4$ kann daraus jedoch keine Schlüsse ziehen. Er wählt zufällig das Bit $b = 0$ und schickt es an Alice
3. Alice ermittelt $a \oplus b = 1$ und gibt das Ergebnis des Münzwurfes bekannt. Außerdem schickt sie $x = 14$ an Bob, damit dieser ihre Festlegung nachprüfen kann

Interaktives-Beweissystem für 3-COL

Um zu zeigen, dass jede Sprache in NP ein interaktives Beweissystem hat, wird für das NP-Vollständige Problem "k-Färbbarkeit von Graphen", genauer für die 3-färbarkeit von Graphen ein interaktives Beweissystem erstellt und bewiesen. Da 3-COL NP-Vollständig ist, kann mittels Reduktion dies für alle Sprahen in NP

erweitert werden:

k-Färbbarkeit von Graphen
Dabei ist $k \in \mathbb{N}$ eine feste natürliche Zahl (im weiteren$k = 3$)

Eingabe: Ein ungerichteter Graph G

Ausgabe: Gibt es eine zuordnung von k verschiedenen Farben zu Knoten in V, sodass keine zwei benachbarten Knoten v_1, v_2 dieselbe Farbe haben?

Computationional Zero-Knowledge-Beweissystem für 3-COL
Gegeben: ungerichteter Graph G mit$V = \{v_1, ..., v_n\}$

1. Peggy berechnet eine beliebige 3-Färbung $c : V \mapsto \{1, 2, 3\}$ für G
2. Peggy schickt Festlegung $C(c(v_1), ..., C(v_n))$ an Victor
3. Victor wählt eine zufällige Kante $\{u, w\} \in E$ und schickt sie an Peggy
4. Peggy weist gegenüber Victor nach, dass sie die Farbe $c(u), c(w)$ gewählt hat und dass $c(u) \neq c(w)$

Ein Simulator kann die Transkripte nicht exakt fälschen, wenn er keine 3-Färbung von G kennt, aber es gil: Das obige Protokoll liefert ein computational Zero-Knowledge-Beweissystem

Vollständigkeit: falls G 3-färbbar, kann Peggy korrekte 3-Färbung berechnen.
$\Rightarrow$ Victor akzeptiert mit Wahrscheinlichkeit 1 (1 $ge\frac{1}{4}$ vollständigkeit gegeben)

Korrektheit: falls G nicht 3-färbbar, dann existiert in dem gefärbten Graphen eine Kante $\{u, v\} \in E$ mit $c(u) = c(v)$. Die Wahrscheinlichkeit das Victor eine korrekt gefärbte Kante wählt ist: $1 - \frac{1}{|E|}$[14]
$\Rightarrow$ lasse das Protokoll k-mal durchlaufen sodass die W'keit $(\frac{1}{|E|})^k \leq \frac{1}{4}$ also: $k \geq \frac{-\log 4}{\log(1-\frac{1}{|E|})}$. Peggy muss bei jedem Durchlauf die Färbung neu wählen!

Zero-Knowledge-Eigenschaft: Der Simulatorarbeitet wie folgt: Der Simulator wählt eine Färbung des Graphen zufällig und ruft V^*(bösartiges V) mit Festlegung $C(c(v_1)...)$ auf und erhält Kante $\{u, w\}$. Wenn $c(u) \neq c(w)$ dann gibt er das Transkript aus, wenn nicht wiederholt er dies bis er zwei Kanten erhält die nicht gleich-gefärbt sind.
Bemerkung: Ein Beobachter kann diese Transskripte in Polynomzeit nicht von eine echten Protokollablauf unterscheiden, computational statt perfekter Zero-Knowledge-Eigenschaft gilt hier, da die Festlegung nicht unabhängig von der Färbung c aber in nicht beobachtbarer Weise abhängig sind.

[14] E := Anzahl der Kanten

Wenn das Problem A in Polynomzeit auf $3-COL$ reduzierbar ist, und man eine dazugehörige Reduktionsfunktion g besitzt, dann ist das Zero-Knowledge-Beweissystem für A einfach, dass Peggy und Victor das Eingabe Problem mittels g auf einen Graphen reduzieren und dann das obige Protokoll durchführen. Die Antwort von Victor zu $3 - col$ ist die Antwort auf die Frage $x \stackrel{?}{=} A$.

6.5. Secure Multi-Party Computations

Es soll eine zweistellige Funktion berechnet werden, Alice und Bob wollen gemeinsam das Ergebnis berechnen. Jeder kennt genau ein Argument, ohne das jeweilige Argument dem anderen bekannt zu geben. Jeder bekommt am Ende ein Teilergebnis, welches zum gesamt Ergebnis zusammen gesetzt werden kann. Alices-Anteil ist grün, bob's blau
Hierfür wird ein Hilfprotokoll Oblivious Transfer benötigt!

6.5.1. Oblivious Transfer

Gemeinsame Berechnung der Projektions-Funktion:

$$f : \{0,1\}^k \times \{1,...,k\} \mapsto \{*\} \times \{0,1\}$$
$$f((c_1,...c_k),i) = (*, c_i)$$

Alice und Bob wollen diese Projektion gemeinsam berechnen, so dass:

- Alice kennt $c_1,...,c_k$ und erfährt keine weitere Information (sie weiß nicht welches c_I bob erhalten hat)
- Bob kennt zu beginn des Protokolls i und am Ende nur i und c_i

Hierfür wird eine Bijektive Falltürfunktion (trapdoor permutation) t benötigt. t^{-1} kann nur effizient mit dem privaten Schlüssel (den in diesem Fall Alice besitzt) berechnet werden. Desweiteren wird ein Hardcore-Prädikat B zu t benötigt.
Protokoll:

1. Alice bestimmt eine bijektive Falltürfunktion $t : X \mapsto X$ mit Hardcore-Prädigkat $B : X \mapsto \{0,1\}$ und schickt (t, B) an Bob. (Festlegung auf die Verfahren)
2. Bob wählt zufällig $z_1,...,t(z_i),...,z_k$. Bob verschlüsselt(mit dem "public Key" von Alice) nur z_i, dass ist das Bit, welches er von Alice erfahren möchte (der Rest ist Klartext)

3. Alice erhält $(y_1, ..., y_k)$ und schickt $(c_1 \oplus B(t^{-1}(y_1)), ..., c_k \oplus B(t^{-1}(y_k)))$. Alice "entschlüsselt" alle mit t^{-1} und wendet das Hardcore-Prädikat darauf an und XOR-Verknüpft es mit ihren Bits. (nur $t(z_i)$ wird entschlüsselt, der Rest wird "praktisch" signiert)

4. Bob empfängt $(x_1, ..., x_k)$ und bestimmt $x_i \oplus B(z_i)$
   Korrekthei(einfach einsetzten)t:

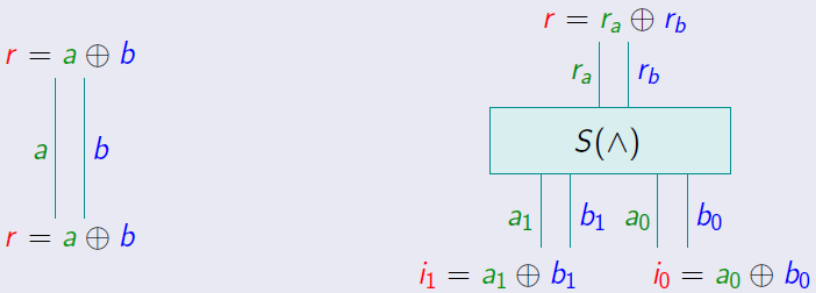
$$x_i \oplus B(z_i) = \underbrace{c_i \oplus B(t^{-1}(y_i))}_{x_i} \oplus B(z_i) = c_i \oplus B(t^{-1}(\underbrace{t(z_i)}_{y_i})) \oplus B(z_i) = c_i$$

Sicherheit:

- Alice kann z_j und $t(z_j)$ nicht unterscheiden und erhält dadurch keine Information darüber, welches c_i Bob erhalten hat
- Bob kann aus $c_j \oplus B(t^{-1}(z_j))$ (für $i \neq j$) nicht $B(t^{-1}(z_j))$ - und damit c_j - bestimmen, da er nur $t(t^{-1}(z_j)) = z_j$, nicht aber $t^{-1}(z_j)$ kennt. **Aber** von Bob muss das korrekte Verhalten erzwungen werden, denn wenn er alle $z_1, .., z_k$ mit t verschlüsselt, würde er alle Bits $c_1, ..., c_n$ bekommen.

Jetzt:
Mit dem Oblivious Transfer protokoll können Schaltkreise erzeugt werden, welche eine sichere Auswertung erlauben. Jeder Teilnehmer stellt seine Eingabebits zur Verfügung ohne das der andere diese erfährt.
Das führt zu folgender **Idee:**
Daraus können nun Schaltkreise erzeugt werden welche die geforderte Berechnung

- Jede Verbindung zwischen Gattern wird verdopppelt.
- Für jede Belegung r einer Verbindung zwischen Gattern bekommen Alice und Bob Anteile a, b mit $r = a \oplus b$.
- Ein Gatter mit k Eingängen und m Ausgängen wird durch ein "sicheres Gatter" mit $2k$ Eingängen und $2m$ Ausgängen ersetzt.



durchführen können.

Bemerkungen:

- Unter den Voraussetzungen, dass die (bei Oblivious Transfer) verwendeten Funktionen tatsächlich Falltürfunktionen mit dazugehörigen Hardcore-Prädigkat sind und sich Alice und Bob korrekt (semo-hones behaviour) bezüglich des Protokolls verhalten, liefert dieser Ansatz eine Möglichkeit gemeinsame Berechnungen durchzuführen ohne dass einer der Teilnehmer in polynomieller Zeit Informationen über die Eingabewerte des jeweils anderen bestimmen kann.(abgesehen von den Informationen, die sie durch Kenntnis des berechneten Funktionswert erhalten).
- Das System kann auf mehr als zwei Parteien erweitert werden.

7. Mathematische Definitionen

7.1. Wahrscheinlichkeitsrechnung

7.2. AES Mathe-Grundlage

Zwecks Byte-weiser Verarbeitung beruht unter anderem AES auf algebraischen Rechenoperationen im so genannten Galois-Körper GF(2^8):

- Polynome mit Koeffizenten aus $\mathbb{Z}_2$[15]$= \{0, 1\}$
- bilde Restklassen modulo des **irreduziblen** Polynomen(Polynomdivision und den Rest nehmen):
$$m(x) = x^8 + x^4 + x^3 + x + 1$$
- repräsentiert sind alle Polynome Grad ≤ 7 über $\mathbb{Z}_2 = \{0, 1\}$;diese bilden den Körper GF(2^8)
- Elemente von GF(2^8) lassen sich als Bytes auffassen:
$$x^7 + x^6 + 1 = (11000001)_2 = (c1)_{\text{hex}})$$

Addition

- Addition von Polynomen mit Koeffizienten in $\mathbb{Z}_2 = \{0, 1\}$:
$$\left.\begin{array}{l}(b_7x^7 + ... + b_1x + b_0)\\(b'_7x^7 + ... + b'_1x + b'_0)\end{array}\right\} = (b_7 \oplus b'_7)x^7 + ... + (b_1 \oplus b'_1)x + (b_0 \oplus b'_0) \quad (7.1)$$
- d.h. bitweises XOR:
$$(57)_{10} \oplus (83)_{10} = (10010111)_2 \oplus (10000011)_2 = (00010100)_2 = (14)_{10}$$

[15] $\mathbb{Z}_2$ ist der Ring mit 2 Elementen. In dem Ring existieren Addition(XOR-Verknüpfung) und Multiplikation(UND-Verknüpfung) und Bilden somit einen Körper

Multiplikation

- Polynome multiplizieren und Ergebnis reduzieren modulo $m(x) = x^8 + x^4 + x^3 + x + 1$ und den Rest nehmen
- die Implementierung erfolgt modifiziert für eine schnellere Berechnung (X-Time)

7.3. Grundlagen für RSA

7.3.1. Zahlenmengen

Wenn folgende Axiome gelten:

$$(a+b)+c = a+(b+c) \qquad a(bc) = (ab)c \qquad \text{Assoziativität}$$
$$a+0 = 0 \qquad a \cdot 1 = a \qquad \text{Neutrales Element}$$
$$a+b = b+a \qquad ab = ba \qquad \text{Kommutativität}$$
$$a+(-a) = 0 \qquad \text{Additives Inverses}$$

Dann ist $\langle \mathbb{Z}, +, 0 \rangle$ eine kommutative Gruppe und $\langle \mathbb{Z}, \cdot, 1 \rangle$ ein kommutatives Monoid Mit:

$$a(b+c) = ab + ac \qquad \text{Distributivität}$$

$\Rightarrow$ Damit ist $\langle \mathbb{Z}, +, 0, \cdot, 1 \rangle$ ein kommutativer Ring mit 0 und 1.

7.3.2. Restklassenrechnung

Es sei $n \in \mathbb{N}$. Die Menge der Restklassen modulo n ist:

$$\mathbb{Z}_n = 0, 1, 2, ..., n-1$$

Alle möglichen Reste der Division ganzer Zahlen durch n.
Für $n \in N$ bildet

$$(-) \mod n : \mathbb{Z} \to \mathbb{Z}_n$$

ganze Zahlen auf ihren nicht negativen Rest bei Division durch n ab.
$a, b \in \mathbb{Z}$ heißen **kongruent modulo n** wenn $a \mod n = b \mod n$ gilt.
Arithmetische Operationen und Reduktion $\mod n$ sind vertauschbar!

Exponentiation

Es gelten dieselben Exponentiationsregeln wie in $\langle \mathbb{Z}, +, 0, \cdot, 1 \rangle$:

$$e^z \mod n = (\underbrace{e \cdot e \cdots e}_{z\text{-mal}}) \mod n$$

Anstatt e z-mal zu multiplizieren und erst dann $\mod n$ zu nehmen, kann man nach jeder teil-multiplikation $\mod n$ rechnen. Somit bleibt an in den Restklassen und die Zahlen wachsen nicht an, dieses vorgehen nennt man schnelle Exponentiation.
Würden erst alle multiplikationen durchgeführt bräuchte man $z-1$ multiplikationen, was exponentiell in der Bitlänge k des Exponenten z wäre.
Stattdessen ergibt die Binärdarstellung des Exponenten mit den Exponentiationsregeln:

$$a^z \mod n = a^{\sum_{i<k} z_i \cdot 2^i} \mod n = \prod_{i<k} (a^{2^i} \mod n)^{z_i} \mod n \tag{7.2}$$

Die Binärdarstellung des Exponenten kann im Verlauf des Algorithmus mitberechnet werden mit $z_i = \lfloor z/2^i \rfloor \mod 2, i < k$ (das heißt in Hardware einen linksshift um i bits, und $\mod 2$ heißt nichts anderes als das letzte Bit Betrachtet ob 0 oder 1):
Der Algorithmus benötigt zwischen k und $2k-1$ multiplikationen modulo n, wobei

```
Data: n, z ∈ ℕ, a ∈ ℤ
Result: x = a^z mod n
a_1 := a;
z_1 := z;
x := 1;
// Schleifeninvariante: (x · a_1^{z_1}) mod n = a^z mod n
while z_1 ≠ 0 do
    while (z_1 mod 2) = 0 do
        z_1 := z_1/2;
        a_z := (a_1 · a_1) mod n;
    end
    z_1 := z_1 − 1;
    x := (x · a_z) mod n;
end
```

Abbildung 7.1.: Pseudo-Code schnelle Exponentiation

k die Bitlänge von z ist.

7.3.3. Multiplikatives Inverse

Existiert ein Inverses zu $a \in \mathbb{Z}_n$, ist es eindeutig und wird mit $a^{-1} \mod n$ bezeichnet (aber nicht jedes $a \in \mathbb{Z}_n$ hat ein Inverses).
Für $n \geq 2; a \in \mathbb{Z}_n$ gilt:

Es existiert $a^{-1} \mod n$ genau dann, wenn $\text{ggT}(a, n) = 1$ gilt.

also wenn a und n teilerfremd sind.

reduzierte Menge der Reste

Alle Elemente von $\mathbb{Z}_n$ welche teilerfremd zu n sind (und damit ein inverses besitzen) werden in der Menge $\mathbb{Z}_n^*$ zusammengefasst:

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n | \ \text{ggT}(a, n) = 1\} \tag{7.3}$$

Die Anzahl der invertierbaren Elemente in $\mathbb{Z}_n$ (also Mächtigkeit von $\mathbb{Z}_n^*$) wird $\varphi(n)$ genannt und wird als Euler'sche φ-Funktion bezeichnet. Bzgl. der Multiplikation verhält sich $\mathbb{Z}_n^*$ 'gut'. Multipliziert man zwei Zahlen $a \cdot b \mod n, a, b \in \mathbb{Z}_n^*$ erhält man wieder eine teilerfremde Zahl zu n.

7.3.4. Euler'sche φ-Funktion

Es gilt:

- ist p eine Primzahl gilt $\varphi(p) = p - 1$.
- ist p eine Primzahl und $k > 0$ gilt $\varphi(p^k) = p^k - p^{k-1}$. Also alle $p^{k-1} - 1$ vielfachen von p sind nicht teilerfremd.
- sind p, q Primzahlen und $p \neq q$ gilt $\varphi(pq) = (p-1)(q-1)$. Besitzt also $\varphi(p) \cdot \varphi(q)$ multiplikative Inverse zu $\mathbb{Z}_{pq}$.

Um $\varphi(n)$ zu berechnen, muss die Faktorisierung von n bekannt sein.

Satz von Euler

Sei $n \in \mathbb{N}$ und $a \in \mathbb{Z}_n$ mit $\text{ggT}(a, n) = 1$, dann gilt[16]:

$$a^{\varphi(n)} \mod n = 1 \tag{7.4}$$

Das liefert die Rechenregel (und die Ver-/Entschlüsselungsfunktion vom RSA !):

> Rechnet man in der Basis modulo n, so darf man im Exponenten modulo $\varphi(n)$ rechnen

[16] ist n eine Primzahl wird es zum Satz von Fermat

heißt für $n \in \mathbb{N}, a \in \mathbb{Z}_n^*$ und $r_0, r_1 \in \mathbb{N}$:

$$r_0 \equiv_{\varphi(n)} r_1 \quad \text{impliziert} \quad a^{r_0} \equiv_n a^{r_1}$$

schnelle Inversenberechnung

Die Euler'sche φ-Funktion liefert eine Möglichkeit das inverse zu einer Zahl $a \in \mathbb{Z}_n^*$ zu bestimmen:

$$a^{-1} \equiv_n a^{\varphi(n)-1} \mod n \tag{7.5}$$

7.3.5. Schnelle Inversenberechnung: Euklidischer algorithmus

Der Euklidische Algorithmus liefert eine Möglichkeit das Inverse zu einer Zahl $a \in \mathbb{Z}_n^*$ zu berechnen.

Idee:

Sei $n \in \mathbb{N}$ und $a \in \mathbb{Z}_n^*$ dann berechnet man mit dem euklidischen Algorithmus den ggT(n, a). Da $a \in \mathbb{Z}_n^*$ liefert der Algorithmus 1 als Ergebnis. Verfolgt man nun den Algorithmus, ausgehen von dem Ergebnis, rückwärts so liefert dieser eine linear Kombination des Ergebnisses:

$$1 = \underbrace{c \cdot n}_{\text{mod } n==0} + d \cdot a$$

Da a bekannt ist muss d das multiplikative Inverse von a bzgl. $\mathbb{Z}_n^*$ sein damit die Gleichung erfüllt ist !

Bei der Vorwärtsberechnung des euklidischen Algorithmus kann parallel dazu Berechnungen ausführen, sodass direkt nach der Berechnung des ggT(n, a) das Inverse von a bzgl. $\mathbb{Z}_n^*$ bekannt ist.

Beispiel-Berechnung von $51^{-1} \mod 56350$: Damit erhält man:

$$\begin{array}{rcl} 56350 = 1104 \cdot 51 + 46 \\ 51 = 1 \cdot 46 + 5 \\ 46 = 9 \cdot 5 + 1 \\ \hline 5 = 5 \cdot 1 + 0 \end{array} \qquad \begin{array}{rcl} 0 &=& 1104 \cdot 1 + (-1104) \\ 1 &=& 1 \cdot (-1104) + 1105 \\ -(1104) &=& 9 \cdot 1105 + (-11049) \end{array}$$

$$51^{-1} \mod 56350 = (-11049) \mod 56350 = \underline{45301}$$

7.4. Ordnung eines Elementes (für DSA)

Es sei $n \in \mathbb{N}$. Die Ordnung eines Elementes $a \in \mathbb{Z}_n^*$ ist die kleinste natürliche Zahl k mit:

$$a^k \mod n = 1$$

Bemerkung:

- die Existenz der Ordnung ist durch den Satz von Euler gesichert. Spätestens $a^{\varphi(n)} \mod n = 1$ gilt, es kann jedoch ein k existieren mit $k < \varphi(n)$
- Die Existenz einer möglichst großen Ordnung $< p$ durch:

  Es seien p und q Primzahlen mit $q|(p-1)$[17]. Dann existieren Elemente $g \in \mathbb{Z}_p^*$ der Ordnung q. Die Menge $\{g^k|0 \leq k < q\}$ hat genau q Elemente

Algorithmus Ein g lässt sich algorithmisch finden:
repeat

wähle $\alpha \in \{2, ..., p-2\}$ zufällig und berechne $g : \alpha^{\frac{p-1}{q}} \mod p$

until $g \neq 1$
Im Erfolgsfall hat ord(g) höchstens den Wert q, wegen ord(g) $\neq 1$ kann ord(g) als Teiler von q aber auch nicht kleiner sein, da q Primzahl. Also ord$(g) = q$

7.5. Komplexitätstheorie

In der Komplexitätstheorie geht es darum das man Zeit- und Raumbeschränkung von Algorithmen analysiert und klassifiziert. Als Algorithmenmodell nimmt man Turingmaschinen.

7.5.1. deterministische Turingmaschine + Sprachklassen

Zunächst: Sprachen die von **deterministische** Turingmaschinen mit **zeitbeschränkung** aktzeptiert werden können:
Die Klasse TIME($f(n)$) besteht aus allen Sprachen L, für die es eine deterministische Mehrband-Turingmaschine M gibt welche die Sprache akzeptiert und deren Laufzeit durch eine Funktion f beschränkt ist bzgl. der Länge der Eingabe $time_M(x) \leq f(|x|)$.

[17] q ist ein Teiler von $p-1$

$time_M(x)$ gibt die Anzahl der Rechenschritte von M bei Eingabe x an. Das heißt, die Anzahl der Schritte der Turingmaschine ist beschränkt und die Beschränkung ist abhängig von der Länge der Eingabe.

7.5.1.1. Klasse P

Sprachen, die von deterministischen Turingmaschinen mit **polynomialer Laufzeitbeschränkung** erkannt werden: $time_M(x) \leq p(|x|)$, wobei p ein Polynom ist.

7.5.1.2. Funktionsklasse FP

Analog zur Komplexitätsklasse P kann man auch eine Klasse FP von Funktionen definieren, die in polynomieller Zeit berechnet werden können:
FP ist die Klasse aller Funktionen der Form $f : \Sigma^* \mapsto \Sigma^*$, für die es eine deterministische Turingmaschine M und ein Polynom p gibt, so dass M die Funktion f berechnet und $time_M(x) \leq p(|x|)$ für alle Wörter $x \in \Sigma^*$

7.5.2. nichtdeterministische Turingmaschine + Sprachklassen

Sprachen die von nichtdeterministischen Turingmaschinen mit Zeitbeschränkung akzeptiert werden:
Die Klasse NTIME($f(n)$) besteht aus allesn Sprachen L, für die es eine nichtdeterministische Mehrband-Turingmaschine M gibt welche die Sprache akzeptiert und deren Laufzeit durch eine Funktion f beschränkt ist bzgl. der Länge der Eingabe: $ntime_M(x) \leq f(|x|)$ für alle $x \in L$:

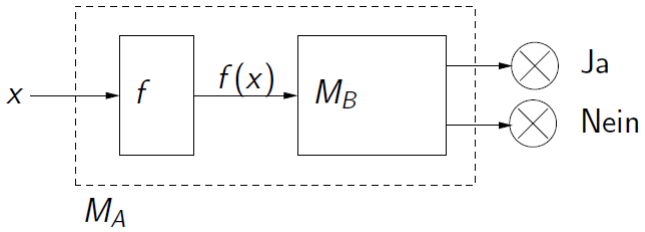
$$ntime_M(x) = \begin{cases} \min\{\text{Länge akzeptierender Rechnungen von } M \text{ auf } x\}, & \text{falls akzeptiert} \\ 0, & \text{falls nicht akzeptiert} \end{cases}$$

7.5.2.1. Klasse NP

Sprachen, die von nichtdeterministischen Turingmaschine mit **polynomialer Laufzeitbeschränkung** erkannt werden: $ntime_M(x) \leq p(|x|)$, wobei p ein Polynom ist.
Es gilt: $P \subseteq NP$
Nicht bekannt ob gilt: $P = NP$?

7.5.3. Polynomiale Reduzierbarkeit

Vergleich der Schwere von Problemen durch polynomialer Reduzierbarkeit: A heißt auf B reduzierbar $A \leq_p B$ wenn mithilfe einer Funktion f aus einer Maschine M_B für das Problem B immer eine Maschine M_A für das Problem A konstruiert werden kann. Dabei hat f polynomielle Laufzeit. Anhand der Abbildung heißt das: es exisitiert eine Funktion f die die Eingabeinstanz x bzgl. des Problems A in polynomieller Zeit korrekterweise zu einer Eingabeinstanz bzgl. Problem B übersetzt. Wird als x in A akzeptiert so wird die übersetzte Instanz $f(x)$ von B akzeptiert und für nicht Akzeptanz genauso.



7.5.4. NP-Hart & NP-Vollständig

NP-hart: Eine Sprache A heißt **NP-hart**, falls für alle Sprachen $L \in NP$ gilt:

$$L \leq_p A$$

Das heißt eine Sprache A die von jeder anderen Sprache aus NP eine polynomielle Reduktion hat, A ist folglich für alle Sprachen in NP stellvertretend schwer lösbar (existiert ein Lösungsalgorithmus für A, so hat man einen für alle Sprachen in NP)

NP-vollständig: Eine Sprache A heißt **NP-vollständig**, falls A NP-hart ist und $A \in NP$ gilt[18]. Das bedeutet: eine NP-vollständige Sprache ist mindestens so schwierig wie jedes andere Problem in NP

7.5.5. Komplement Klassen

Neben NP ist oft auf die Klasse aller Komplemente von NP-Problemen interessant. Die Komplexitätsklasse cp-NP enthält genau die Sprachen, deren Komplement in NP liegt.

[18] könnte man Zeigen das eine NP-Vollständige Sprache A in P liegt, hätte man die Gleichheit der Klassen gezeigt

Es gilt: $P \subseteq NP \cap co\text{-}NP$
P liegt im Durchschnitt von $NP \cap co\text{-}NP$, aber man weiß nicht wie NP und co-NP in Beziehung stehen.

7.6. Probabilistische Komplexitätstheorie

Um probabilistische Algorithmen klassifizieren zu können werden wieder Komplexitätsklassen benötigt. Zuallerst muss der Begriff "probabilistischer Algorithmus" definiert werden, hierfür wird eine **Probabilistische Turingmaschine verwendet.**

7.6.1. Probabilistische Turingmaschine

Eine probabilistische Turingmaschine M ist ein 7-Tupel
$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ wobei alle Komponenten außer δ wie bei einer herkömmlichen Turingmaschine definiert sind. Der unterschied zur "normalen" Turingmaschine

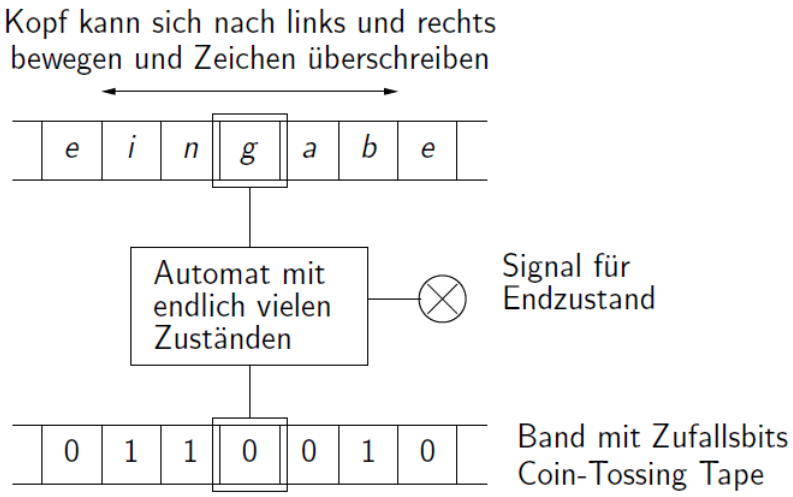


Abbildung 7.2.: probabilistische Turingmaschine

ist, dass diese ein weiteres Band(read-only) besitzt (das untere) mit Zufallsbit belegt ist(gleichverteilte Folge von 0 und 1). Vereinfacht gesagt, heißt das die Turingmaschine wirft bei jedem Schritt eine Münze und macht bei 0 Übergangsfunktion A und bei 1 Übergangsfunktion B wählt. Wenn die Turingmaschine auf einer Eingabe x hält, dann ist das Ergebnis 0 (Ablehnung von x) oder 1 (Annahme von x)

7.6.2. Halten und Laufzeit

Hält: Eine probabilistische Turingmaschine **hält**, wenn sie auf jeder Eingabe - unabhängig von den verwendeten Zufallsbits - hält, d.h. einen Endzustand erreicht.

Laufzeit: Eine probabilistische Turingmaschine hat eine polynomiale Laufzeit, wenn sie auf jeder Eingabe x - unabhängig von den verwendeten Zufallsbits - höchstens $p(|x|)$ Schritte macht, wobei p ein Polynom ist(oder eine andere wenn p kein polynom).

Die verwendeten Zufallsbits werden als Elementarereignisse aufgefasst. Für ein festes Eingabewort x ist die Menge der Elementarereignisse definiert als: $\Omega = \{0,1\}^{p(|x|)}$. Das ergibt die Zufallsvariablen:

- Ob das Eingabewort x von der Maschine M akzeptiert wird ist die Zufallsvariable:

  $a_M(x)(\tilde{b}) = 1 \leftrightarrow$ M akzeptiert das Wort x unter Verwendung der zufallsbit $\tilde{b} \in \Omega$

- Die Laufzeit der Maschine M auf dem Wort x ist ebenfalls eine Zufallsvariable:

  $$t_M(x)(\tilde{b}) = \text{Anzahl der Schritte der Berechnung auf x mit Zufallsbit } \tilde{b}$$

7.6.3. Klasse RP (Randomized polynomial time)

Eine Sprache $L \subseteq \Sigma^*$ gehört zu der Klasse RP genau dann, wenn es eine probabilistische Turingmaschine M mit polynomialer Laufzeit gibt, so dass für jede Eingabe $x \in \Sigma^*$ gilt:

- falls $x \in L$, dann gilt P(M akzeptiert x) $= P(a_M(x) = 1) \geq \frac{1}{2}$
- falls $x \notin L$, dann gilt P(M akzeptiert x) $= P(a_M(x) = 1) = 0$
  heißt die Wahrscheinlichkeit das M x akzeptiert wenn x nicht in der Sprache liegt ist gleich 0

Die Wahrscheinlichkeit von $\frac{1}{2}$ kann erhöht werden indem die Maschine mehrmals(mit jeweils anderen Zufallsbits) laufen lässt: Betrachte die Maschine M': M' führt die Maschine M k-mal aus und akzeptiert, sobald M einmal akzeptiert. Da die jeweils verwendeten Zufallsbits unabhängig voneinander sind, gilt für $x \in L$:

$$P(M' \text{ akzeptiert x nicht }) \leq \left(\frac{1}{2}\right)^k$$

$$P(M' \text{ akzeptiert x }) \geq 1 - \left(\frac{1}{2}\right)^k$$

Um mit Wahrscheinlichkeit $\geq p$ zu akzeptieren, muss M k-mal laufen:

$$k = \log_2 \frac{1}{1-p}$$

Beispiel NON-ZERO-POLY: Entscheidungsproblem: Geben ist ein Polynom p mit Grad kleiner gleich k. Ist das Polynom identisch zu Null oder nicht? Gibt es ganze Zahlen sodass das Polynom ungleich 0 ist?
Idee:
Zur Berechnung von NON-ZERO-POLY: wähle zufällig beliebige ganze zahlen $a_1, ..., a_n \in \{1, ...N\}$ und setze sie in p ein. Das Polynom wird akzeptiert, falls $p(a_1, ..., a_n) \neq 0$ gilt:

- Dieser Test läuft sicherich in Polynomzeit, aufgrund des gegebenen arithmetischen Ausdrucks. (Polynom bedingung für RP erfüllt)
- Das verfahren akzeotiert niemals ein Polynom, das identisch zu Null ist. D.h. die zweite Bedingung in der Definition von RP ist aufjeden fall erfüllt. (liegt es nicht in der Sprache, ist die wahrscheinlichkeit der Akzeptanz = 0)
- Damit NON-ZERO-POLY in RP liegt, muss die Wahrscheinlichkeit das eine gültige Eingabe akzeptiert wird $\geq \frac{1}{2}$ sein.
  Die Wahrscheinlichkeit für Nullstellen (polynom p com grad k) (Schwartz-Zippel-Lemma) ist $P(p(a_1, ..., a_n) = 0) \leq \frac{k}{N}$. Werden die Koeffizienten wie oben gewählt: $a_1, ..., a_n \in \{1, ...N\}$ und setzt für N $2k$ ein, ergibt das die geforderte Wahrscheinlichkeit $P(p(a_1, ..., a_n) = 0) \leq \frac{k}{2k} = \frac{1}{2}$. Damit liegt NON-ZERO-POLY in RP !

Beziehung zu anderen Klassen:

- Es gilt: P $\subseteq$ RP. Denn jede deterministische Turingmaschine kann als probabilistische aufgefasst werden die ihre Zufallsbits ignoriert
- Es gibt: RP $\subseteq$ NP. Eine nichtdeterministische TM wird zu einer probabilistischen, wenn diese $p(|x|)$ Bits nichtdeterministisch auf ein Hilfsband schreibt und diese anschließend als Zufallsbits interpretiert.

7.6.4. Klasse BPP (bounded-error probabilistic polynomial time)

Eine Sprache L $\subseteq \Sigma^*$ gehört zu der Klasse BPP genau dann, wenn es eine probabilistische Turingmaschine M mit polynomialer Laufzeit gibt, so dass für jede Eingabe $x \in \Sigma^*$ gilt:

- falls $x \in L$, dann gilt P(M akzeptiert x) = P$(a_M(x) = 1) \geq \frac{3}{4}$
- falls $x \notin L$, dann gilt P(M akzeptiert x) = P$(a_M(x) = 1) \leq \frac{1}{4}$
- Wie bei RP kann die Wahrscheinlichkeit für eine falsche Antwort vermindert werden, indem man die Maschine mehrfach laufen lässt !

Beziehung zu anderen Klassen:

- es gilt: RP $\subseteq$ BPP
- Es ist unbekannt wie sich NP und BPP zueinander verhalten ! weder NP $\subseteq$ BPP noch BPP $\subseteq$ NP ist bekannt.

7.6.5. Klasse ZPP (zero-error probabilistic polynomial time)

Eine Sprache $L \subseteq \Sigma^*$ gehört zu der Klasse ZPP genau dann, wenn es eine probabilistische Turingmaschine M gibt, so dass der **Erwartungswert** $E[X]$ der Laufzeit polynomial beschränkt ist und für jede Eingabe $x \in \Sigma^*$ gilt:

- falls $x \in L$, dann gilt P(M akzeptiert x) = P($a_M(x) = 1$) = 1
- falls $x \notin L$, dann gilt P(M akzeptiert x) = P($a_M(x) = 1$) = 0
- Für die TM nach der Definition von ZPP gilt für jedes Eingabewort x: die Zufallsvariable $t_M(x)$, die die Laufzeit von M in Abhängigkeit der Zufallsbits beschreibt erfüllt $E[t_M(x)] \leq p(|x|)$ für ein Polynom p

Die Idee dieser Klasse ist, dass der Algorithmus immer richtig Antwortet. Also nicht die Antwort ist eine Zufallsvariable wie bei RP und BPP sondern die Laufzeit ist die Zufallsvariable. Also die Laufzeit ist nicht immer polynomial, aber im Mittel ist diese polynomial.
Für Probleme in ZPP gilt also: Es gibt einen Algorithmus, der immer die richtige Antwort gibt, aber es ist unklar, ob dieser Algorithmus immer polynomiale Laufzeit hat. Inuitiv Bedeutet das der Algorithmus läuft nur selten lange.
Unterscheidung bei Algorithmen:

- Las-Vegas-Algorithmen: arbeiten wie TM, die ZPP-Probleme entscheiden, Laufzeit aber nicht die Ausgabe ist randomisert (die Ausgabe ist unabhängig von den Zufallsbits) !
- Monte-Carlo-Verfahren: arbeiten wie TM für RP bzw. BPP. Laufzeit ist immer polynomial beschränkt aber die Ausgabe ist randomisiert (also abhängig von den Zufallsbits)

Beziehung zu anderen Klassen:

- es gilt: Ein Problem L liegt in ZPP genau dann, wenn L und sein Komplement in RP liegen.
  $\rightarrow$ ZPP = RP $\cap$ co-RP

7.6.6. Gesamt Klassenbild

Die Klassen verhalten sich, mit den genannten unbekannten folgendermaßen:

$$\text{P} \subseteq \text{ZPP} = \text{RP} \cap \text{cp-RP} \subseteq \text{RP} \subseteq \text{RP} \subseteq \text{BPP}$$
$$\text{RP} \subseteq \text{NP}$$

Es ist nichts über das Verhältnis zwischen BPP und NP bekannt. Außerdem gibt es Vermutungen, dass P und ZPP gleich sein könnten, aber auch das ist nicht bewiesen.

7.6.7. Wichtige Algorithmen zugeordnet

Der Rabin-Miller-Primzahltest mit:

- Eingabe: n $>$ 1 ungerade
- Ausgabe: “n zusammengesetzt“ (sicher) oder “n Primzahl“ mit Fehler-W’keit $\leq \frac{1}{4}$

Gefragt: “ist n zusammengesetzt“:

- wenn n zusammengesetzt, dann gilt P(M sagt n zusammengesetzt) mit w’keit $\geq \frac{3}{4}$
- wenn n Prim, dann gilt P(M sagt n zusammengesetzt) mit w’keit $= 0$

und damit $\in$ RP. Damit liegt auch “COMPOSITE“ und “PRIMES“ in RP . (PRIMES sogar in P)

7.7. Funktionen für Kryptosicherheit

7.7.1. Einwegfunktionen

Informell: Einwegfunktionen sind Funktionen, die leicht zu berechnen sind - sie liegen in FP - und deren Umkehrung fast immer schwer zu berechnen ist. Um die Sicherheit von Kryptosystemen zu gewährleisten wird zusätzlich die Forderung gestellt, dass die Umkehrung auch durch probabilistische Verfahren schwer zu berechnen ist. Es wird noch die Einschränkung auf ehrliche Funktionen benötigt:

> Eine Funktion f $\Sigma^* \mapsto \Gamma$ heißt ehrlich, falls es ein Polynom q gibt, so dass für alle $c \in \Sigma^*$ gilt: $|x| \leq q(|f(x)|)$

Das heißt, die Verkürzung der Eingabe durch die Funktion f ist durch ein Polynom beschränkt. Da ansonsten allein das schreiben des Ergebnisses von f^{-1} exponentiell ist.

Definition: Einwegfunktion

Die Wahrscheinlichkeit ein Kryptosystem zu brechen soll vernachlässigbar sein:
Vernachlässigbar heißt: Eine Funktion $r : \mathbb{N} \mapsto \mathbb{R}_0^+$ heißt vernachlässigbar, wenn es für jedes Polynom p eine Zahl $C \in \mathbb{N}$ gibt mit:

$$r(n) \leq \frac{1}{p(n)}$$

heißt, für wachsendes Argument n, fällt r schneller gegen 0 als jedes Polynom wächst.
Eine Einwegfunktion ist dann:

Eine Funktion $f : \{0,1\}^* \mapsto \{0,1\}^*$ heißt Einwegfunktion, falls gilt:

1. f in FP liegt
2. f ehrlich ist
3. für jede probabilistische polynomzeitbeschränkte Turingmaschine M ist die Wahrscheinlichkeit vernachlässigbar, dass M ein korrektes Urbild für $f(x)$ berechnet, falls x zufällig aus der Menge $\{0,1\}^n$ gewählt wird.
   - Ein Korrektes Urbild berechnet heißt:

     $$f(M(f(x))) = f(x)$$

     da f im Allgemeinen nicht bijektiv ist, kann $M(f(x)) \neq x$ gelten.
   - Vernachlässigbar ist also die W'keit für ein korrektes Urbild für eine feste Länge n von x

     $$P(f(M(f(x))) = f(x))$$

     Es kann nicht geforderten werden das für jedes $x \in \{0,,1\}^n$ die W'keit vernachlässigbar ist, da man nie verhindern kann, dass ein Algorithmus zum Brechen eines Kryptosystems auf bestimmten Chiffretexten korrekt funktioniert. (Gebe bspw. immer einen festen Klartext zurück)

Bemerkungen:

- Die Sicherheitsforderung, dass die Verschlüsselungsfunktion E_K für einen festen Schlüssel K in einem Kryptosystem eine Einwegfunktion sein soll ist **nicht** stellbar. Da eine Einwegfunktion nicht direkt für ein Kryptosystem verwendbar ist, denn auch die Dechiffrierfunktion D_K muss **effizient** berechenbar sein !
- Die Existenz von Einwegfunktionen garantiert die Existenz von Pseudo-Zufallsgeneratoren.

ABER: man weiß nicht ob Einwegfunktionen existieren !

- aus der Existenz von Einwegfunktionen würde folgen:
  - P $\neq$ NP und NP $\not\subseteq$ BPP
  - Existenz von Pseudo-Random-Generatoren
  - Es ist unbekannt ob aus P $\neq$ NP oder NP $\not\subseteq$ BPP die Existenz von Einwegfunktionen folgt. Grund: eine Einwegfunktion sollte nicht nur im worst-case schwer zu invertieren sein.

7.7.2. Faktorisierung - potentielle Einwegfunktion

Primzahlmultiplikation *pmult*:

$$pmult(q, p) = p \cdot q \tag{7.6}$$

wobei q, p beide k-Bit Primzahlen sind.
Die Funktion *pmult* liegt in FP ! (1. Bedingung für Einweg erfüllt, "einfach" zu berechnen)
Umkehrfunktion von *pmult*:
Die Umkehrfunktion von *pmult* bestimmt die Faktorisierung eines Produkts von Primzahlen. Dies kann auch als Entscheidungsproblem formuliert werden:

Eingabe: Zwei Zahlen $n, r \in \mathbb{N}$, wobei n das Produkt von zwei k-Bit Primzahlen ist

Ausgabe: Besitzt n einen Faktor $p > 1$, der kleiner als r ist?

- Das Problem liegt in NP !
- Es ist jedoch nicht bekannt, ob es NP-vollständig ist
- Auch aus der NP-Vollständigkeit und aus P $\neq$ NP kann nicht geschlossen werden das *pmult* eine Einwegfunktion ist!

7.7.3. Diskreter Logarithmus - potentielle Einwegfunktion

Für den Diskreten Logarithmus wird ein zahlentheoretischer Grundbegriff benötigt:

7.7.3.1. Einschub: Primitivwurzel modulo n

Eine Zahl $g \in \{1, ..., n-1\}$ heißt Primitivwurzel modulo n, falls g ein Generator der multiplikativen Gruppe $\mathbb{Z}_n^* = \{a \in \{1, ..., n-1\} | \text{ggT}(a, n) = 1\}$ ist. Das heißt es muss gelten:

$$\{g^k \mod n | k \in \mathbb{N}\} = \mathbb{Z}_n^* \tag{7.7}$$

Bemerkungen:

- Nicht für alle $n \in \mathbb{N}$ existieren Primitivwurzeln
- Falls n Primzahl ist, existiert eine Primitivwurzel modulo n
- Falls n Primzahl ist, ist nicht jedes Element von $\mathbb{Z}_n^*$ ungleich 1 eine Primitivwurzel. Beispiel:

  n=7 : $2^0 \mod 7 = 1,\ 2^1 \mod 7 = 2,\ 2^0 \mod 7 = 1,\ 2^2 \mod 7 = 4,$
  $2^3 \mod 7 = 1$
  $\rightsquigarrow$ 2 ist keine Primitivwurzel modulo 7 (bei 2^3 war das Ergebnis 1, damit ist man einmal "rum" und hat noch nicht jedes Element von $\mathbb{Z}_7^*$ erzeugt)

  n=7 : $3^0 \mod 7 = 1,\ 3^1 \mod 7 = 3,\ 3^2 \mod 7 = 2,\ 3^3 \mod 7 = 6,$
  $3^4 \mod 7 = 4,\ 3^5 \mod 7 = 5,\ 3^6 \mod 7 = 1$
  $\rightsquigarrow$ 3 ist eine Primitivwurzel modulo 7

  - Die Anzahl der Primitivwurzeln modulo p ist $\varphi(p-1)$. (es gibt genug :))
  - Ein $g \in \mathbb{N}$ ist Primitivwurzel mpdulo p $\leftrightarrow g^{\frac{p-1}{q}} \mod p \neq 1$

    1. Wähle eine große Primzhl q so dass $p = 2q + 1$ auch Prim. (so ein p hießt **sichere Primzahl**)
    2. Wiederhole:
       wähle $1 < g < p - 1$ zufällig und teste ob $g^q \mod p\ ne1$
       bis g gefunden

7.7.3.2. Definition: Diskrete- Exponentialfunktion \Logarithmusfunktion

Diskrete Exponentialfunktion *dexp*:

> Gegeben sei eine Primzahl p, eine Primitivwurzel $g \mod p$ und $x \in \mathbb{Z}_p^* = \{1, ..., p-1\}$. Dann ist die diskrete Exponentialfunktion definiert:

$$dexp(p, g, x) = (p, g, \underbrace{g^x \mod p}_{\text{eigentliche Funktion}})$$

Die Funktion *dexp* liegt in FP ! (schnelle Exponentation !)
Die **Umkehrfunktion** von *dexp* führt auf das Problem: Die Umkehrfunktion von *dexp* bestimmt den Exponenten x:

Eingabe: Primzahl p, Primitivwurzel g modulo p, $y \in \mathbb{Z}_p^*$

Ausgabe: der eindeutige Exponent $x \in \mathbb{Z}_p^*$ mit $g^x \mod p = y$

Bemerkungen:

- Der diskrete Logarithmus gilt als schwer berechenbar - bspw. benötigt Shank's Algorithmus: Laufzeit: $\mathcal{O}(\sqrt{p})$ und Speicher: $\mathcal{O}(\sqrt{p})$
- Auch für den diskreten Logarithmus exisitiert ein dazugehöriges Entscheidungsproblem in NP, unbekannt jedoch ob dies NP-Vollständig ist.

7.8. Falltürfunktionen

Für asymmetrische Verschlüsselung muss noch mehr gefordert werden: sogenannte **Falltürfunktionen**:
(Informell:) Die Verschlüsselungsfunktion E eines asymmetrischen Kryptosystems heißt Falltürfunktion wenn:

- E in Polynomzeit berechenbar ist, wenn der öffentliche Schlüssel bekannt ist,
- Es existiert ein Polynom $p(x)$ mit: $E(m, e) = c \Rightarrow |m| \leq p(|c|)$
- die Entschlüsselungsfunktion D in Polynomzeit berechenbar ist, wenn der private Schlüssel bekannt ist
- **und** die Wahrscheinlichkeit vernachlässigbar (für eine probabilistische polynomzeitbeschränkte Turingmaschine) ist, für einen zufällig gewählten Klartext und Schlüssel, E nur unter Kenntnis des öffentlichen Schlüssels zu invertieren.

oder Anders: für jede probabilistische polynomzeitbeschränkte Turingmaschine M ist die Wahrscheinlichkeit vernachlässigbar, dass M für ein zufällig gewählte m, e ein Urbild von $E(m, e)$ unter Kenntniss von e berechnet

Bezug zu RSA:
Über die Sicherheit von RSA weiß man: Falls die RSA-Schlüssel (e, n) und (d, n) bekannt sind, gibt es einen probabilistischen Polynomzeit Algorithmus, der n faktorisiert.
Das Finden des privaten Schlüssels ist gleichbedeutend mit der Faktorisierung von n. **Aber** das bedeutet nicht, dass das Brechen des KRyptosystems gleichbedeutend mit der Faktorisierung ist !

7.9. Pseudo-Zufallsgeneratoren

Ein Pseudo-Zufallsgenerator ist ein deterministischer Algorithmus, der aus einem gegebenen Startwert eine Sequenz von Bits erzeugt, die von einer probabilistischen polynomzeit-beschränkten Turingmaschine nicht von einer zufällig erzeugten Bit-Sequenz unterschieden werden kann.
Anwendungen:

- Nach austausch eines symmetrischen Schlüssels kann dieser als Startwert für einen Pseudo-Zufallsgenerator verwendet werden. Damit kann der One-Time-Pad simuliert werden und sichere symmetrische Verschlüsselung erreicht werden.
- Generierung von Nonces[19]
- Generierung von Zufallszahlen für randomisierte Verfahren, Derandomisierung von randomisierten Verfahren

$\Longrightarrow$ Der private Schlüssel ist eine “geheime Falltür“, mit deren Hilfe die Verschlüsselungsfunktion doch einfach zu invertieren ist.
Bemerkungen:

- Vermutung: RSA-Kryptosystem stellt eine Fallrütfunktion dar. Unbewiesen denn:
- Es gilt: Wenn Falltürfunktionen existieren, dann existieren auch Einwegfunktionen (und deren existenz ist unbekannt !)

[19] In der Kryptographie wurde die Bezeichnung Nonce (Abkürzung für: ?used only once?[4] oder ?number used once?[5]) aufgegriffen, um einzelne Zahlen- oder eine Buchstabenkombination zu bezeichnen, die nur ein einziges Mal in dem jeweiligen Kontext verwendet werden. Somit ist eine Nonce im Prinzip ein zufällig generierter und eindeutiger Sitzungsschlüssel.

- Offenes Problem: Wüsste man das die Primzahlmultiplikation eine Einwegfunktion ist, wüsste man trotzdem erstmal nicht ob die RSA-Funktion eine Falltürfunktion ist

8. Protokollverifikation

Für die Verifikation findet eine Änderung der Sichtweise statt:

> Sowohl symmetrische als auch asymmetrische Verschlüsselung ist sicher und kann nicht gebrochen werden !

Das Verifikationsziel ist:

- Der Angreifer hat keinerlei Möglichkeiten, das Protokoll zu unterlaufen
- D.h. er kommt nicht in den Besitz von geheimen Informationen, kann sich nicht für einen Dritten ausgeben etc.
- Dies soll durch (halb-) automatische Techniken verifiziert werden
- d.h. der Angreifer wird nicht mehr als polynomzeitbeschränkte probabilistische Turingmaschine

8.1. Angreifer Modell:

Der Angreifer kann

- an beliebigen Protokollabläufen teilnehmen
- beliebige Nachrichten abfangen
- Nachrichten entschlüsseln, falls er den entsprechenden Schlüssel besitzt
- alte Nachrichten wieder einspielen und
- aus bekannten Komponenten neue Nachrichten bauen und an beliebige Kommunikationspartner verschicken

Der Angreifer kann explizit **nicht**

- Nachrichten entschlüsseln, wenn er den entsprechenden Schlüssel **nicht** besitzt oder
- Nonces erraten, die er nicht abgefangen hat

Das heißt er hat nicht nur eine vernachlässigbare Wahrscheinlichkeit, Nachrichten zu entschlüsseln, oder Nonces zu erraten, sondern **gar keine** Möglichkeit.
Annahme:
Ein Teilnehmer, der eine verschlüsselte Nachricht empfängt, kann herausfinden, ob diese Nachricht mit einem bekannten Schlüssel K verschlüsselt ist. Er erhält dann entweder die entschlüsselte Nachricht oder eine Fehlermeldung (selbiges gilt für Signaturen).

8.2. Protokollbeschreibung

Für die Verifikation (vor allem für die automatische) wird eine exakte formale Protokollbeschreibung benötigt, die keine Mehrdeutigkeiten enthält. Wir betrachten als Beschreibungsformalismus den **angewandten π-Kalkül**
Die bisherigen Beschreibungsmechanismen reichen nicht aus, bsp:

1. $B \mapsto A: \quad P_B$
2. $A \mapsto B: \quad \{\{K\}_{S_A}\}_{P_B}$
3. $B \mapsto A: \quad \{m\}_K$

Diese Notation ist unpräzise:

- Sie sagt nichts darüber aus, was ein Teilnehmer berechnet, nachdem eine Nachricht empfangen wurde:
  - Wird überprüft, ob alle Komponenten der Nachricht korrekt sind?
  - Werden zurückgeschickte Nonces mit früheren generierten oder erhaltenen Nonces verglichen?
  - WErden Zeitstempel überprüft?
  - Was macht der Teilnehmer, wenn die Nachricht nicht das korrekte Format hat?
- Es ist unklar, welche Möglichkeiten der Angreifer hat
  - Können mehrere Protokollabläufe gleichzeitig durchgeführt werden?
  - Könnte ein (oder beide) Teilnehmer auch gleichzeitig einen Protokollablauf mit dem Angreifer durchführen?

Das obige Beispielprotokoll besitzt einen Man-In-The-Middel Angriffsmöglichkeit.
Grundlegendes-Verifikations Schema:

- Modelliert man dieses Protokoll nun im angewandten π-Kalkül so erhält man auch ein Angreifermodell und die Möglichkeit, das mögliche Systemverhalten als Transitionssystem darzustellen. Es kann das Verhalten aller Protokoll-Teilnehmer formal beschreiben.
- In diesem Modell kann man dann mit Analysetechniken nach ungewünschten zuständen durchsucht werden:
  - Einschränkung auf endlichen Zustandsraum und Durchsuchen des Zustandsraum (Model-Checking)
  - Horn-Formeln und Resolution.(wenn es sagt es gibt keinen Angriff, dann gibt es keinen. Wenn es sat es gibt einen könnte dies ein False-Negative sein)
  - Verhaltensäquivalenzen

Man kann unterschiedliche Eigenschaften des Protokolls verifizieren, wir betrachten hauptsächlich:

- Geheimhaltung: bestimme Nachrichteninhalte bleiben gegenüber dem Angreifer geheim
- Authentifizierung: der Angreifer ist nicht in der Lage, sich als ein anderer Teilnehmer auszugeben.

8.3. Angewandter π-Kalkül

Ist ein Prozesskalküle sind "Mini-Programmiersprachen", in denen Interaktion von Prozessen sehr kompakt auch sehr präzise beschrieben werden kann.

8.3.1. Syntax des angewandterπ-Kalkül

8.3.1.1. Namen & Variablen

Es gibt **Namen** und **Variablen**.
Es gibt eine Menge von Namen, deren Elemente als Kanalsnamen, Schlüssel, Nonces, Nachrichteninhalte, etc. verwendet werden.
Außerdem gibt es eine Menge von Variablen.

Namen: Kanelnamen: a,b,c,.. || Schlüssel: K || Nonces: n || Klartextnachrichten: m

Variablen: $x, y, z, x_1, x_2, ...$

8.3.1.2. Funktionssymbole & Terme

Es gibt eine Menge von Funktionssymbolen. Aus den Namen, Variablen und Funktionssy,bolen werden Terme gebildet.
Def: Funktionssymbol:

> Eine Signatur ist eine Menge von Funktionssymbolen mit Stelligkeit. Das heißt eine Signatur $(S_n)_{n\in\mathbb{N}}$ ist eine folge von Mengen $f \in S_n$. Ein Symbol f aus S_n hat eine Stelligkeit von n. S_0 sind sog. Konstanten (kein Argument).

Beispiel: Funktionssymbole für Verschlüsselung, Entschlüsselung... : $S_2 = \{encrypt, decrypt\}$
Terme: encrypt(m,K);decrypt(encrypt(m,x),y);...

8.3.1.3. Gleichungen

Gegeben ist eine Menge Σ von Gleichungen auf Termen, die das Verhalten der Funktionssymbole bzw. der dazugehörigen Funktionen beschreiben. Aus gegebenen Gleichungen Σ sind weitere Gleichungen herleitbar, Notation $\Sigma \vdash M = N$, mittels regeln (lesbar: Wenn man das was im Zähler steht beweisen kann, dann ist es einem erlaubt den Nenner syntaktisch zu verwenden.):

$$\text{Reflexivität}\ \frac{}{\Sigma \vdash M = M} \qquad \text{Symmetrie}\ \frac{\Sigma \vdash M = N}{\Sigma \vdash N = M}$$

$$\text{Transitivität}\ \frac{\Sigma \vdash M = M' \quad \Sigma \vdash M' = M''}{\Sigma \vdash M = M''}$$

$$\text{Axiom}\ \frac{(M = N) \in \Sigma}{\Sigma \vdash M = N}$$

$$\text{Substitution}\ (x\ \text{Variable})\quad \frac{\Sigma \vdash M' = N'}{\Sigma \vdash L[M'/x] = L[N'/x]} \qquad \frac{\Sigma \vdash M' = N'}{\Sigma \vdash M'[L/x] = N'[L/x]}$$

$$\text{Umbenennung}\ (n\ \text{Name})\quad \frac{\Sigma \vdash M = N}{\Sigma \vdash M[n \mapsto m] = N[n \mapsto m]}$$

- Substitution: Der Term L hat eine Variable x und für diese wird M' eingesetzt. Beispiel: encrypt$(x, k)[xor(m, n)/x]$ heißt, das x wird mit $xor(m, n)$ ausgetauscht, also syntaktisch wird daraus: encrypt$(xor(m, n), k)$

8.3.2. Funktionssammlung

Wir brauchen Funktionen die unsere Kryptographischen Grundbausteine darstellen, diese werden im Folgenden definiert:

Symmetrische Verschlüsselung

- Funktionen sdecrypt und sencrypt, die als ersten Parameter eine Nachricht ud als zweiten Paramter einen Schlüssel erhalten
- Definierende Gleichungen (es werden variablen verwendet, damit diese durch Substitution mit beliebig komplizierten Klartexten / Schlüssel ersetzt werden können):

$$\Sigma = \{\text{sdecrypt}(\text{sencrypt}(x, y), y) = x\}$$

Asymmetrische Verschlüsselung

- Funktionen adecrypt und aencrypt, plus zusätzlich einstellige Funktionen sk, pk, die aus einem Geheimnis einen privaten und lffentlichen Schlüssel ableiten.
- Definierende Gleichungen:

$$\Sigma = \{\text{adecrypt}(\text{aencrypt}(x, \text{pk}(y)), \text{sk}(y)) = x\}$$

- Zusätzlich z den Funktionen pk,sk gibt es zweistellige Funktionen sigcheck, sigget und sign und eine Konstante ok
- Definierende Gleichungen:

$$\begin{aligned}\Sigma = \{&\text{sigcheck}(\text{sign}(x, \text{sk}(y)), \text{pk}(y)) = ok \\ &\text{sigget}(\text{sign}(x, \text{sk}(y)), \text{pk}(y)) = x\}\end{aligned}$$

Logischer Operator xor

- Wir verwenden eine zweistellige Funktion xor, definiert auf den Wahrheitswerten 0,1 (Konstantensymbole)

- Definierende Gleichung:

$$\begin{aligned}\Sigma = \{&xor(x,y) = xor(y,x),\\ &xor(xor(x,y),z) = xor(x,xor(y,z))\\ &xor(0,x) = x\\ &xor(x,x) = 0\}\end{aligned}$$

Paare

- Zweistellige Funktionen pair und einstellige Projektionsfunktionen fst, snd
- Definierende Gleichungen:

$$\begin{aligned}\Sigma = \{&\text{fst}(\text{pair}(x,y)) = x\\ &\text{snd}(\text{pair(x,y)}) = \text{y}\}\end{aligned}$$

Notation: (x, y) statt pair(x,y)

Einwegfunktionen

Einwegfunktionen werden modelliert mit einem einstelligen Funktionssymbol f ohne Gleichungen. Da es kein Funktionssymbol für die Umkehrfunktion gibt, kann auch niemand Urbilder ermitteln. Beispiel: $A = \bar{c}(f(g)).0$ Dabei ist g ein "Geheimnis". Kein Angreifer der $f(g)$ als Nachricht empfängt, kann den Wert g ermitteln. Hashfunktionen können ähnlich modelliert werden

8.3.3. Prozess im π-Kalkül

Ein Prozess ist:

- der inaktive Prozess 0 - Prozess ohne Aktion ,
- eine parallele Komposition P|Q - die Prozesse Q und P laufen parallel ,
- eine Replikation !P - erzeugt unendlich viele Kopien von P z.B. P|!P(Erzeugung beliebig vieler paralleler kopien von P),
- eine Restriktion (νu)P - u ist ein Name; dieser ist nur in P bekannt und nicht außerhalb (z.B. für Schlüssel oder Nonces). Der Name u ist gebunden,
- eine Verzweigung if M = N then P else Q,

- ein Prozess mit Eingabe-Präfix c(x).P - c ist ein Kanalname. Das Konstrukt beschreibt eine Eingabe, der Prozess P erhält auf dem Kanal c eine Eingabe, die in x gespeichert ist. Die variable x ist gebunden !,
- ein Prozess mit Ausgabe-Präfix $\bar{c}(M).P$ - Das Konstrukt beschreibt eine Ausgabe, der Prozess P schreibt einen Term M auf dem Kanal c

Erweiterung: event(M).P - Der Prozess P signalisiert den Term M (events werden fpr authentifizierung benötigt, ein Angreifer kann diese nicht benutzten)

Die Bindungskonventionen sind: $\rightsquigarrow$ = "bindet stärker als"

$$!, (\nu n), c(x), \bar{c}(M) \rightsquigarrow (if M = N then P else Q) \rightsquigarrow P|Q$$

Ein Prozess ist abgeschlossen wenn er keine freie[20] variablen hat ! Gebundene Variablen und Namen können (gebunden) umbenannt werden ohne de Semantik zu ändern z.b.:

$$P = (\nu k).b(x).\bar{b}\langle \text{sencrypt}(x, k)\rangle$$
$$P' = (\nu l).b(y).\bar{b}\langle \text{sencrypt}(y, l)\rangle$$

Folgende strukturelle Kongruenzen gelten: **Bemerkungen:**

Strukturelle Kongruenz ($\equiv$) ist die kleinste Äquivalenzrelation auf Prozessen, die die folgenden Paare enthält:

$$P \equiv P \mid 0 \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad P \mid Q \equiv Q \mid P$$
$$!P \equiv P \mid !P \qquad (\nu u)0 \equiv 0 \qquad (\nu u)(\nu v)P \equiv (\nu v)(\nu u)P$$
$$P \mid (\nu u)Q \equiv (\nu u)(P \mid Q) \quad \text{falls } u \text{ nicht frei in } P \text{ vorkommt}$$
$$P[M/x] \equiv P[N/x] \quad \text{falls } \Sigma \vdash M = N$$
$$C[P] \equiv C[Q] \quad \text{falls } P \equiv Q \text{ und } C[-] \text{ ein Evaluationskontext ist}$$
$$P \equiv Q \quad \text{falls } P \text{ und } Q \ \alpha\text{-äquivalent sind}$$

- P[M/x] sit der Prozess P, in dem jedes freies Vorkommen der Variablen x durch den Term M ersetzt wurde. Beachte: freie Namen oder Variablen, die in M vorkommen, dürfen durch die Ersetzung nicht gebunden werden, ist dies der fall müssen die zuerst umbenannt werden. Beispiel: $((\nu n)\bar{a}(x).\bar{b}(n).0)[n/x]$ n ist ein gebundener Name und x eine freie variable. Würde nun die Umbenennung $[n/x]$

[20] In der Mathematik und Logik bezeichnet man eine Variable als in einer mathematischen Formel frei vorkommend, wenn sie in dieser Formel an mindestens einer Stelle nicht im Bereich eines Operators auftritt. Sind hingegen alle Vorkommen der Variable innerhalb der Formel an Operatoren gebunden, bezeichnet man die Variable als in dieser Formel gebunden

einfach durchgeführt, würde x danach gebunden sein. Daher muss das gebundene n vorher umbenannt werden: $((\nu n)\bar{a}(x).\bar{b}(n).0)[n/x] \equiv ((\nu n')\bar{a}(n).\bar{b}(n').0)$

8.3.4. Semantik des angewandten π-kalküls (Reduktionsregeln)

Regeln für die Reduktionsrelation $\longrightarrow$ zwischen Prozessen; formal ist $\longrightarrow$ die kleinste Relation auf Prozessen:

- $\bar{c}(M).P|c(x).Q \longrightarrow P|Q[M/x]$ P übergibt den Term M auf Kanal c aus und Prozess Q liest M parallel auf Kanal c. Das Konstrukt kann reduziert werden sodass die Variable x in Q mit M substituiert wird und P einfach weiter macht.
- $ifM = NthenPelseQ \longrightarrow P, \quad$ falls $\Sigma \vdash M = N$, falls man die Gleichheit von M und N nachweisen kann, dann wird im Fall einer Verzweigung der Thenzweig beteten. Wenn die Gleichheit nicht nachgewiesen werden kann, dann darf der else-zweig nur dann betreten werden, wenn die Terme M,N variablenfrei sind. Das heißt es gibt keine Möglichkeit mehr die Gleichheit von M und N herzustellen.
- event(M).P $\longrightarrow$ P

Damit bekommt man ein unmarkiertes Transitionssystem, also die allgemeinste Form eines nicht-deterministischen Automaten. Unmarkiert, da an den Zustandsübergängen keine Symbole stehen. Es ist ein Tripel $(S, \rightarrow, s_0)$ mit $S :=$ Zustandsmenge, $\rightarrow \subseteq S \times S$ Transitionsrelation und $s_0 \in S$ Startzustand.
Das Transitionssystem eines Prozesses P, sagt im Endeffekt aus wie ein Prozess P als Programm auszuführen ist. Die Idee ist, dass man den Prozess mit den Reduktionen und strukturellen Kongruenzen von P aus jede Möglichkeit durchführt (bis es bei 0 endet, wobei es auch unendliche Läufe geben kann (zb. durch !P)).

8.4. Modellierung des Angreifers

Der herkömmliche Angreifer E ist ein beliebiger Prozess des Kalküls. Dieser wird mit dem eigentlichen System Sys(z.b. Protokoll zwischen Alice und Bob) parallel komponiert: $Sys|E$.
Der Angreifer Prozess kann **keine** Events benutzten !

8.4.1. Geheimhaltung einer Nachricht

Ein System Sys hält eine Nachricht m geheim, wenn es keinen Angreifer E gibt mit einer Reduktionsfolge $Sys|E \longrightarrow^* P$, sodass in P m als Nachrichteninhalt auf einem nicht-verschatteten Kanal verschickt wird. Man reduziert $Sys|E$, gibt es eine Reduktionsfolge in der der Angreifer E zugriff auf m bekommt, so hält das System die Nachricht m nicht geheim und das Protokoll ist nicht sicher

8.4.2. Authentifizierung

> Wenn Alice der Meinung ist, dass sie einen Protokollablauf mit Bob durchführt, dann hat Bob **zuvor** beschlossen, dass er einen Protokollablauf mit Alice durchführt (oder Umgekehrt). Das heißt wenn Alice das Event event((A,B)) auslöst, muss Bob zuvor event((B,A)) auslösen, sonst ist die Authentifizierung **nicht** erfolgt !

Gibt es bei $Sys|E$ eine Reduktion bei der Alice event((A,B)) auslöst ohne dass Bob zuvor event((B,A)) ausgelöst hat, so ist das Protokoll fehlerhaft, die Authentizität ist nicht gesichert, und die Angreiferin Eve kann sich als Bob ausgeben.

8.4.3. unsiversaler Angreifer

Damit man nicht unendlich viele Angreifer ausprobieren muss existiert das konzept des universellen Angreifers. Es existiert ein herkömmlicher Angreifer genau dann wenn ein universeller Angreifer eine Angriffsmöglichkeit findet:

A. Anhang