Zusammenfassung DC

Paul Lödige
Matrikel: 15405036

SoSe 2020

Inhaltsverzeichnis

Kapitel 1

Substitionsverfahren

1.1 Skytale

Ein Streifen wird um einen Stock gewickelt und dann beschrieben. Nach dem abwickeln erhält man den entsprechenden Code. Durch aufwickeln auf einen Stock mit dem gleichen Umfang lässt sich die Nachricht wieder entschlüsseln.

Tipp:
Zum entschlüsseln der Nachricht am PC ist der Editor mit automatischen Zeilen-Wrap gut geeignet

1.2 Monoalphabetische Substitionsverfahren

Jeder Buchstabe wird bijektiv durch einen anderen Buchstaben des gleichen Alphabets ersetzt.

Alphabet:	$\mathscr{Z} := \{A, B, ..., Z\}$
Schlüsselraum:	$\mathscr{K} := \{k : \mathscr{Z} \rightarrow \mathscr{Z} \mid k \text{ ist bijektiv}\}$
Verschlüsselung von $z \in \mathscr{Z}$:	$k(z)$
Entschlüsselung:	$E(z) := k(z)$

1.2.1 Caesar-Verschlüsselung

Alle Buchstaben des Alphabets werden um einen konstanten Wert verschoben.
Beispiel:
Code: $n = 2$
Verschlüsselung: jeder Buchstabe wird durch den übernächsten Buchstaben im Alphabet ersetzt $(E(m) = (m + 3) \mod 26)$.
Entschlüsselung: jeder Buchstabe wird durch den vor-vorherigen Buchstaben ersetzt.

1.2.2 Häufigkeitsanalyse

Monoalphabetische Substitionsverfahren lassen sich mit moderner Technik sehr einfach durch die Verwendung von Häufigkeitsanalysen entschlüsseln. In jeder Sprache gibt es Buchstaben die deutlich häufiger vorkommen als andere. Durch einer Analyse der Häufigkeit der einzelnen Buchstaben im Geheimtext lassen sich diese den Ausgangsbuchstaben zuordnen.

1.3 Polyalphabetische Substitutionsverfahren

Damit sich ein Geheimtext nicht durch eine Häufigkeitsanalyse (1.2.2) entschlüsseln lässt wird bei polyalphabetischen Verschlüsselungsverfahren der Schlüssel regelmäßig gewechselt. Hierbei wird der Schlüsselwechsel meist selbst durch ein Schlüsselwort kodiert.

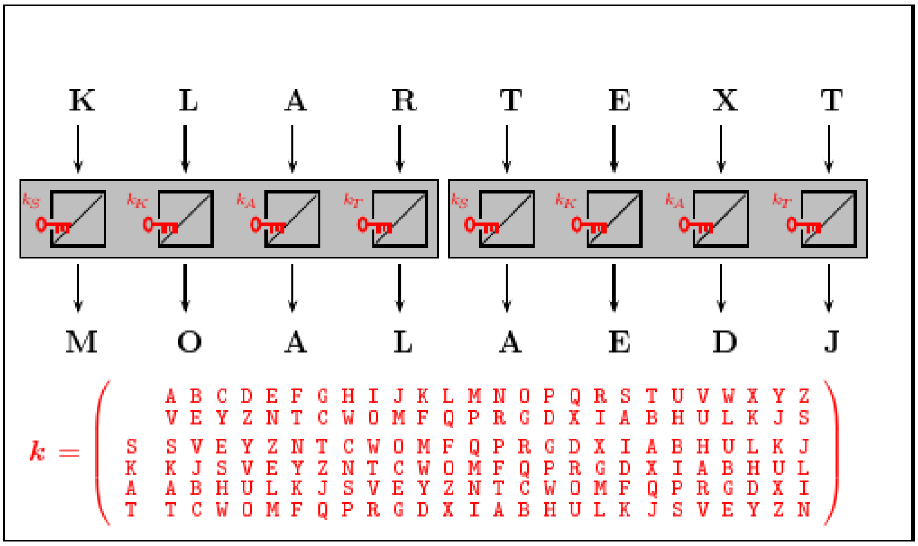
1.3.1 Vignère-Verfahren

Alphabet:	$\mathscr{X} := \{A, B, ..., Z\}$
Menge aller Wörter:	$\mathscr{W} = \mathscr{X}^{>0} := \bigcup_{n\in\mathbb{N}} \mathscr{X}^n = \{(m_1, m_2, ..., m_n) \mid m_i \in \mathscr{X}, n \in \mathbb{N}\}$
Schlüsselraum:	$\mathscr{K} := \Sigma_{\mathscr{X}} \times \mathscr{W}$

Ein Schlüssel $k = (f, w) \in \mathscr{K}$ besteht aus einer Permutation $f \in \Sigma_{\mathscr{X}}$ (siehe 1.2) und einem Schlüsselwort w.
Das Vignère-Verfahren ist ein Blockverschlüsselungsverfahren, bei dem jeweils ein Block von Zeichen (im Beispiel 4) nach dem gleichen Verfahren verschlüsselt wird. Wenn die Blockgröße klein genug oder die Textlänge groß genug sind lässt sich ein solches Verfahren ebenfalls durch eine Häufigkeitsanalyse (siehe 1.2.2) knacken.

1.3.1.1 Verschlüsselung

Der Schlüssel f wird zyklisch mithilfe des Schlüsselwortes w verschoben.
Beispiel:
f = VEYZNTCWOMFQPRGDXIABHULKJS und w = SKAT



1.3.2 One-Time-Pad

Bei dem One-Time-Pad handelt es sich um ein absolut sicheres Substitionsverfahren (Nachrichten, bei denen sich Informationen von der Länge der Nachricht ableiten lassen müssen auf eine konstante Länge gebracht werden).

1.3.2.1 Verschlüsselung

Annahme:	Klar- und Geheimtext sind eine Folge von Zeichen der Menge $\mathscr{X} = \mathbb{Z}$ ($n \in \mathbb{N}$)
Klartextnachricht:	$m = (m_1, m_2, ...m_l) \in \mathbb{Z}_n{}^l$
One-Time-Pad (zufällig):	$k = (k_1, k_2, ...k_l) \in \mathbb{Z}_n{}^l$
Geheimtext:	$E(m) := m + k := (m_1 + k_1, m_2 + k_2, ..., m_l + k_l)$

1.3.2.2 Perfekte Sicherheit

Wenn die Zeichen k_i des Schlüssels mit einem perfekten Zufallsgenerator erzeugt lassen sich **keine** Rückschlüsse auf die Klartextnachricht ziehen (außer Länge).
Allerdings darf ein One-Time-Pad nur ein einziges Mal für die Verschlüsselung verwendet werden. Andernfalls könnte durch eine Berechnung der Differenz der mit dem gleichen OTP verschlüsselten Nachrichten Informationen gewonnen werden. So kann eine Häufigkeitsanalyse (siehe 1.2.2) oder sogar eine komplette Entschlüsselung (falls Klartext einer Nachricht bekannt ist) möglich werden.

1.4 algebraische Substitionsverfahren

1.4.1 Hill-Verfahren

Das Hill-Verfahren ver- und entschlüsselt die Nachrichten mithilfe einer **invertierbaren** Matrix.

1.4.1.1 Verschlüsselung

Annahme:	$n, k \in \mathbb{N}$ sind vorgegeben
Klartextnachricht:	Menge von Blöcken $\mathscr{B} = \mathbb{Z}_n{}^k$
Schlüssel:	**invertierbare** (k, k)-Matrix K
Geheimtext:	$E_K(b) := K \cdot b \qquad (b \in \mathscr{B})$

1.4.1.2 Entschlüsselung

Annahme:	gleiche Bedingungen wie bei der Verschlüsselung
Geheimtextnachricht:	Menge von Blöcken $\mathscr{C} = \mathbb{Z}_n{}^k$
Klartext:	$E_K(c) := K^{-1} \cdot c \qquad (c \in \mathscr{C})$

Kapitel 2

Modulare Arithmetik

2.1 Exkurs: Division mit Rest

Für $a, b \in \mathbb{Z}, b \neq 0$ gibt es eindeutig bestimmte Element $q, r \in \mathbb{Z}, 0 \leq r < |b|$:

$$a = b \cdot q + r$$
$$a/_{\mathbb{Z}}b := q$$
$$a \mod b := r$$

2.2 Der Ring $\mathbb{Z}_n$

Ein Ring $\mathbb{Z}_n$ ist definiert durch:

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

2.2.1 Addition und Multiplikation

$$\begin{aligned} a +_{\mathbb{Z}_n} b &:= (a+b) \mod n \\ a \cdot_{\mathbb{Z}_n} b &:= (a \cdot b) \mod n \end{aligned} \tag{2.1}$$

2.2.1.1 Inverse bezüglich der Addition

jedes $a \in \mathbb{Z}$ hat ein Inverses:

$$-a := \begin{cases} 0 & \text{für } a = 0 \\ n - a & \text{sonst} \end{cases}$$

2.2.1.2 Inverse bezüglich der Multiplikation

ein Element $a \in \mathbb{Z}_n$ ist *(multiplikativ) invertierbar*, falls es ein Element $b \in \mathbb{Z}_n$ gibt, für das gilt:

$$a \cdot b = 1$$

man schreibt auch:

$$a^{-1} := b$$

Die Menge der invertierbaren Elemente in $\mathbb{Z}_n$ wird als $\mathbb{Z}_n^*$ bezeichnet:

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid a \cdot b = 1 \text{ für ein } b \in \mathbb{Z}n\}$$

Zudem gilt, dass ein Element nur dann invertierbar ist, falls $ggT(a, n) = 1$:

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid ggT(a, n) = 1\}$$

2.2.2 Subtraktion

Eine Subtraktion entspricht einer Addition mit der Inverse:

$$a -_{\mathbb{Z}_n} b := a +_{\mathbb{Z}_n} (-b) \mod n$$

2.2.3 Teiler, Vielfache

$b \in \mathbb{Z}$ teilt $a \in \mathbb{Z}$ falls ein $q \in \mathbb{Z}$ existiert mit:

$$a = b \cdot q$$

man schreibt auch $b|a$

2.2.3.1 Teilerregeln

1. $a|0 \; \forall a \in \mathbb{Z}$
2. $a|b \Leftrightarrow a|(-b)$
3. $a|b$ und $a|c \Rightarrow a|(b+c)$

2.2.4 Kongruenz

$a, b \in \mathbb{Z}$ sind *kongruent modulo n*, falls $n \in \mathbb{N}|(a-b)$. Man schreibt auch $a \equiv b$

2.2.5 Matrizen

2.2.5.1 Determinantenberechnung

Die Determinante $\det(A)$ der (N,N)-Matrix $A = (a_{ij})_{1 \leq i,j \leq N}$ (mit ganzzahligen Einträgen) über $\mathbb{Z}_n$ wird definiert durch:

$$\det(A) \mod n = det((a_{i,j} \mod n)_{1 \leq i,j \leq N})$$

Zudem gilt für die Matrizen $A = (a_{ij})_{1 \leq i,j \leq N}$ und $B = (b_{ij})_{1 \leq i,j \leq N}$ (mit ganzzahligen Einträgen):

$$\begin{aligned}\det(A \cdot B) \mod n &= (det(A) \cdot \det(B)) \mod n \\ &= ((\det(A) \mod n) \cdot (\det(B) \mod n)) \mod n \\ &= (\det(A) \mod n) \cdot_{\mathbb{Z}_n} (\det(B) \mod n)\end{aligned}$$

2.2.5.2 Inverse Matrix

Die Inverse einer quadratischen Matrix A über $\mathbb{Z}_n$ lässt sich mithilfe der Adjunkten berechnen:

$$A^{-1} = (\det(A))^{-1} \cdot adj(A)$$

Die Adjunkte lässt sich über $\mathbb{Z}_n$ berechnen, da lediglich Summen und Differenzen von Produkten berechnet werden müssen.

2.2.6 Kartesisches Produkt von Ringen

Sind $R_1, R_2, ..., R_r$ Ringe, so bildet das kartesische Produkt $(R_1 \times R_2 \times \cdots \times R_r)$ebenfalls einen Ring mit den folgenden Verknüpfungen:

$$(a_1, ..., a_r) + (b_1, ..., b_r) = (a_1 + b_1, ..., a_r + b_r)$$
$$(a_1, ..., a_r) \cdot (b_1, ..., b_r) = (a_1 \cdot b_1, ..., a_r \cdot b_r)$$

Auch das kartesische Produkt der Einheitengruppen bildet einen Ring:

$$R^* = R_1^* \times \cdots \times R_r^*$$

2.2.7 Isomorphie von Ringen

Eine bijektive Abbildung $f : R_1 \rightarrow R_2$ wird Isomorphismus genannt, falls für alle $a, b \in R_1$ gilt:

$$f(a+b) = f(a) + f(b)$$
$$f(a \cdot b) = f(a) \cdot f(b)$$

Zwei isomorphe Ringe werden als $R_1 \cong R_2$ geschrieben.

2.3 Der erweiterte Euklid'sche Algorithmus

Der Euklid'sche Algorithmus ist ein sehr effizienter Weg den ggT zweier Zahlen zu ermitteln. Der Euklid'sche Algorithmus lässt sich auch über $\mathbb{Z}_n$ verwenden. Man spricht dann von dem erweiterten Euklid'schen Algorithmus.

2.3.1 Euklid'scher Algorithmus

gegeben: $a_0, b_0 \in \mathbb{Z}$

1. $a := a_0$ und $b := b_0$
2. falls $b = 0$ gebe $|a|$ aus und beende
3. $r := a \mod b$
4. $a := b$
5. $b := r$
6. goto 2.

2.3.2 erweiterter Euklid'scher Algorithmus

gegeben: $a_0, b_0 \in \mathbb{N}_0$
gesucht: $\alpha \cdot a_0 + \beta \cdot b_0 = g = ggT(a_0, b_0)$

1. $a := a_0$, $\alpha_a = 1$, $\beta_b = 0$, $b := b_0$, $\alpha_b := 0$, $\beta_b := 1$
2. falls $b = 0$ gebe $g := a$, $\alpha := \alpha_a$ und $\beta := \beta_a$ aus
3. $q := a/_{\mathbb{Z}}b$
4. $r := a - q \cdot b$,$\alpha_r := \alpha_a - q \cdot \alpha_b$, $\beta_r := \beta_a - q \cdot \beta_b$
5. $a := b$, $\alpha_a := \alpha_b$, $\beta_a := \beta_b$
6. $b := r$, $\alpha_b := \alpha_r$ $\beta_b := \beta_r$
7. goto 2.

2.3.2.1 Beispiel

Eingabe:

$$a_0 = 1224 \text{ und } b_0 = 156$$

Berechnung:

1224	156	a,b	q
1	0	1224	
0	1	156	7
1	-7	132	1
-1	8	24	5
6	-47	12	2
		0	

Ergebnis:

$$6 \cdot 1224 + (-47) \cdot 156 = 12$$

2.4 Euler'sche φ-Funktion

Die Euler'sche φ-Funktion bezeichnet die Anzahl invertierbarer Elemente in $\mathbb{Z}_n$

$$\varphi(n) := \begin{cases} |\mathbb{Z}_n^*| & \text{für } n \in \mathbb{N}, n \geq 2 \\ 1 & \text{für } n = 1 \end{cases}$$

Für $a, b \in \mathbb{N}$ mit $ggT(a,b) = 1$ gilt:

$$\varphi(a \cdot b) = \varphi(a) \cdot \varphi(b)$$

Zudem gilt für ein $n \in \mathbb{N}$ dessen Primzahlzerlegung $n = p_1^{e_1} \cdots p_r^{e_r}$:

$$\varphi(n) = n \cdot \prod_{i=1}^{r} \left(1 - \frac{a}{p_i}\right)$$

2.4.1 φ-Funktion und Primzahlen

für eine Primzahl p gilt:

$$\varphi(p) = p - 1$$

Für Primzahlpotenzen gilt zudem:

$$\varphi(p^e) = p^{e-1}(p-1)$$

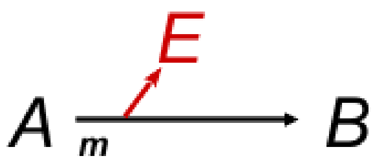
Kapitel 3

IT-Sicherheit: Gefährdungen und Maßnahmen

Im Folgenden wird auf mehrere Schutzziele eingegangen, welche für die IT-Sicherheit wichtig sind.

3.1 Vertraulichkeit

Bei dem Kriterium der Vertraulichkeit geht es darum, dass Daten nicht an unbefugte gelangt



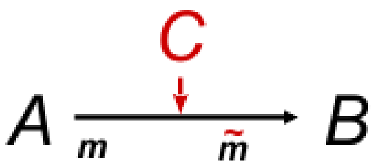
3.1.1 Schutzmaßnahmen: Verschlüsselungsverfahren

Durch die Verschlüsselung der Daten kann eine Gefährdungen der Vertraulichkeit verhindert werden. Allerdings muss hierbei auf folgende Punkte geachtet werden:

- Schlüsselerzeugung:
  Schlüssel müssen mit einem kryptographisch sicheren Zufallsgenerator erzeugt werden
- Schlüsselspeicherung:
  Schlüssel müssen sicher gespeichert sein
- Schlüsselaustausch:
  Damit zwei Systeme informationen austauschen können müssen zunächst Schlüssel ausgetauscht werden. Bei synchronen Verschlüsselungsverfahren muss dieser Austausch auf einem sicheren Weg geschehen.

3.2 Integrität

Bei dem Kriterium der Integrität geht es darum, dass die Daten, die verschickt werden auch unverändert empfangen werden.

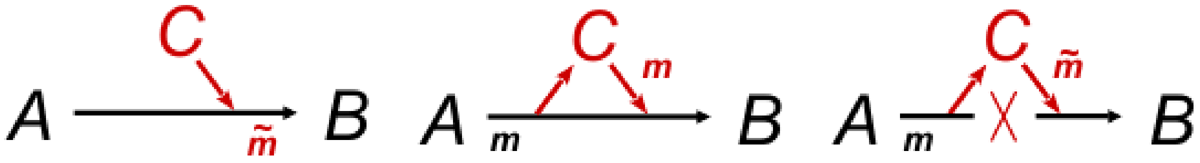


3.2.1 Schutzmaßnahme: Hashfunktionen, Whitelists

zur Sicherung der Integrität werden mithilfe von Hashfunktionen Prüfsummen errechnet und in einer Whitelist abgespeichert.

3.3 Authenzität der Daten

Bei dem Kriterium der Authenzität geht es darum, sicherzustellen, dass die empfangenen Daten auch tatsächlich vom angegebenen Absender stammen.



3.3.1 Schutzmaßnahme: Signaturen

Um die Authenzität sicherzustellen gibt es mehrer Möglichkeiten:

1. es wird ein zweiter Kommunikationsweg für die Authentifikation verwendet (2-Factor-Authentification)
2. **Signaturverfahren:**
   Eine Signatur wird (mit dem Signaturverfahren S) berechnet und mithilfe eines privaten Schlüssels k_{pri} verschlüsselt. Der Empfänger nutzt den öffentlichen Schlüssel k_{pub} und das zu S gehörigen Verifikationsverfahren V um die Nachricht zu authentifizieren.
3. **MAC-Verfahren:**
   Sender und Empfänger einigen sich auf einen geheimen Schlüssel k. Anschließend nutzt der Sender diesen Schlüssel um den MAC(Message Authentication Code)-Wert der Nachricht zu verschlüsseln. Wenn der Empfänger den MAC-Wert entschlüsselt kann er die Nachricht authentifizieren.

3.3.2 Schutz vor Replay-Angriffen

Um einen Replay-Angriff zu verhindern muss dafür gesorgt werden, dass jede Nachricht nur ein einziges Mal akzeptiert wird. Mögliche Verfahren hierfür sind Zählwerte, die mit jeder Nachricht inkrementiert werden oder Zeitstempel.

3.4 Authenzität von Nutzern

In vielen Fällen ist es nötig einen Nutzer zu authentifizieren (z.B. Anmeldung auf einer Webseite). Diese Authentifikation ist eine spezielle Form der Nachrichtenauthentifikation, bei der der Inhalt der übertragenen Daten nicht relevant ist.

3.4.1 Schutzmaßnahmen

- Nutzen eines gemeinsamen Geheimnisses (z.B. WLAN-Passwort)
- Nutzen eines Schlüssels, den nur eine Seite besitzt (siehe 3.3.1)
- Nutzen von einmaligen Eigenschaften (z.B. Fingerabdruck)

3.5 Zugriffskontrolle

Bei Systemen, die eine Aktion ausführen ist es wichtig abzusichern, dass nur erlaubte Aktionen angefragt werden können.

3.5.1 Schutzmaßname: Zugriffskontrollsystem

Es werden Listen (Access Control Lists) darüber geführt, welcher Nutzer welche Aktionen veranlassen darf. Hierbei werden die Rechte häufig in Form von Rollen vergeben (Role Based Access Control).

3.6 Nichtabstreitbarkeit, Verbindlichkeit

Eine Form der Authentifikation oder Authentifizierung, die auch gegenüber dritten unwiderlegbar ist. Dies ist vor allem für Kommunikationen wichtig, bei denen es für eine Partei vorteilhaft wäre sie abzustreiten (z.B. Verträge).

3.6.1 Schutzmaßname: Signaturen und PKI

Signaturen (siehe 3.3.1) können auch als Beweis für die Nichtabstreitbarkeit verwendet werden, falls der öffentliche Schlüssel der dritten Partei bekannt ist. Hierfür wird eine öffentliche Infrastruktur (Public Key Infrastructur), welche von Zertifizierungsstellen zur Verfügung gestellt wird (z.B. ITU (für X.509)).

3.7 Verfügbarkeit

Bei dem Kriterium der Verfügbarkeit geht es darum, dass die Daten und IT-Systeme wie angedacht erreichbar sind. Mögliche Bedrohungsszenarien hierfür sind:

- Datenverlust durch defekte Daten
- Datenverlust durch Schadsoftware
- Nichterreichbarkeit von Diensten aufgrund von Netzwerkproblemen
- Nichterreichbarkeit von Webdiensten aufgrund erfolgreicher Denial-of-Service-Angriffen

3.7.1 Schutzmaßnahmen

- redundante örtlich verteilte Datenspeicherung
- Virenscanner und Paketfilter zum Schutz vor Malware
- Firewalls

3.8 Anonymität

Es gibt viele Gründe, wegen denen es sinnvoll ist, dass eine Datenübertragung sicher aber ohne den Versand persönlicher Daten funktioniert.

Kapitel 4

Verschlüsselungsverfahren

4.1 Das Kerckhoffs'sche Prinzip

„Ein Verschlüsselungssystem darf nicht der Geheimhaltung bedürfen und soll ohne Schaden in Feindeshand fallen können."

Folglich ist für die Entschlüsselung der Nachricht nicht die Kenntnis über das Verfahren, sondern der Schlüssel die relevante Information.

„Es soll nicht möglich sein, einen Geheimtext ohne Kenntnis des hierfür vorgesehenen Schlüssels **effizient** zu entschlüsseln."

4.2 Mathematische Modelierung von Verschlüsselungsverfahren

Klartextnachrichten und Geheimtextnachrichten sind Elemente einer Menge $\mathscr{M}$.
Schlüssel sind Elemente eines Menge $\mathscr{K}$, die als Schlüsselraum bezeichnet wird.
Die Verschlüsselungsverfahren bestehen aus einem Paar von Funktionen zur Verschlüsselung (E) und Entschlüsselung (D):

$$E : \mathscr{K} \times \mathscr{M} \to \mathscr{M}$$

$$D : \mathscr{K} \times \mathscr{M} \to \mathscr{M}$$

Hierbei sind die definierten Abbildungen bijektiv ($D_k := E_k^{-1}$):

$$E_k : \mathscr{M} \to \mathscr{M}, E_k(m) := E(k, m)$$

$$D_k : \mathscr{M} \to \mathscr{M}, D_k(m) := D(k, m)$$

Die Nachrichtenmenge $\mathscr{M}$ besteht in Realität aus einer endlichen Folgen (Tupeln) von Bit- oder Bytewerten. Diese können entweder die gleiche Länge besitzen (Blockverschlüsselung) oder von beliebiger Länge sein (Stromverschlüsselung).

$$\mathscr{M} = \mathscr{Z}^n = \{(z_1, z_2, ..., z_n) \mid z_i \in \mathscr{Z}\} \text{ für ein festes } n \in \mathbb{N} \text{ (Blockverschlüsselung)}$$

$$\mathscr{M} = \mathscr{Z}^{>0} = \bigcup_{n \in \mathbb{N}} \mathscr{Z}^n = \{(z_1, z_2, ..., z_n) \mid z_i \in \mathscr{Z}, n \in \mathbb{N}\} \text{ (Stromverschlüsselung)}$$

4.3 Schlüsselaustausch

Um mithilfe eines symmetrischen Schlüssels Daten austauschen zu können muss der Schlüssel auf eine Sichere Art und Weise ausgetauscht werden. Dies ist zwar offline möglich, stellt allerdings kein praktikables Verfahren für die Kommunikation im Internet dar. Daher werden für einen sicheren Schlüsselaustausch über eine unsichere Infrastruktur asymmetrische Verschlüsselungsverfahren benötigt.

4.4 Angriffsszenarien

Da für eine effiziente Entschlüsselung einer Geheimtextnachrichten die Kenntnis des Schlüssels essentiell ist versuchen die meisten Angriffe diesen herauszufinden.

4.4.1 Ciphertext-only Angriffe

Vorraussetzung: ein oder mehrere Geheimtexte $c_i = E_k(m_i)$ bekannt
Angriffsziel: Bestimmung von m oder von k

4.4.2 Known-plaintext Angriffe

Vorraussetzung: Geheimtext $c = E_k(m)$ und eine Reihe von bekannten Paaren $(m_i, E_k(m_i))$ sind bekannt
Angriffsziel: Bestimmung von m oder von k

4.4.3 Chosen-plaintext Angriffe

Vorraussetzung: Geheimtext $c = E_k(m)$ und für eine Reihe von beliebig vorgegebenen Klartextnachrichten m_i kan
Angriffsziel: Bestimmung von m oder von k

4.5 Brute-Force Angriffe

Da die Vorraussetzung für ein gutes Verschlüsselungsverfahren ist, dass es nicht **effizient** entschlüsselt werden kann (siehe 4.1) muss die **Effizienz** definiert sein. An dieser Stelle setzen Brute-Force Angriffe an. Sie versuchen wie der Name schon sagt mit roher Rechenleistung den Schlüssel zu ermitteln.

4.5.1 Beispiel: Brute-Force Angriff auf k

Unter der Annahme, dass ein known-plaintext Angriff (siehe 4.4.2) vorliegt lässt sich der Schlüssel bestimmen, indem alle möglichen Schlüssel k des Schlüsselraums $\mathscr{K}$ „ausprobiert" werden. Hierbei können für die einzelnen bekannten Klartextnachrichten mehrere Schlüssel passen:

$$\mathscr{K}_i := \{\tilde{k} \in \mathscr{K} \mid E_{\tilde{k}}(m_i) = c_i\}$$

Bei einer ausreichenden Menge bekannter Klartextnachrichten lässt sich der Schlüssel aus der Schnittmenge der möglichen Schlüssel bestimmen:

$$\bigcap_{i=1}^{N} \mathscr{K}_i = \{k\}$$

4.5.2 Beispiel: Brute-Force Angriff auf m

Liegen die Vorraussetzungen für einen chosen-plaintext Angriff vor, so kann die Klartextnachrichten herausgefunden werden, indem alle möglichen Klartextnachrichten $\tilde{m} \in \mathscr{M}$ „ausprobiert" werden:

$$E_k(\tilde{m}) = c = E_k(m) \Longrightarrow \tilde{m} = m$$

4.5.3 Anforderungen zum Schutz vor Brute-Force

1. Es soll keinen Angriff auf den Schlüssel k geben, der durchschnittlich weniger als $\frac{|\mathscr{K}|}{2}$ Ver- oder Entschlüsselungsoperationen braucht.

2. Es soll keinen Angriff auf die Klartextnachricht m geben, der durchschnittlich weniger als $\min\left\{\frac{|\mathscr{K}|}{2}, \frac{|\mathscr{M}|}{2}\right\}$ Ver- oder Entschlüsselungsoperationen braucht.

4.6 Wörterbuchangriffe

Nachrichtenpaare (Geheim- und Klarttext), die mithilfe eines Schlüssels k verschlüsselt wurden können in einem Wörterbuch abgespeichert werden. Mithilfe des Wörterbuchs können diese Paare jederzeit entschlüsselt werden. Zudem kann für einen chosen-plaintext Angriff (siehe 4.4.3) ein Wörterbuch mit häufig vorkommenden Wörtern erstellt werden, sodass eine Entschlüsselung in wesentlich kürzerer Zeit möglich wird.

4.6.1 Schutz vor Wörterbuchangriffen

Um einem Wörterbuchangriff vorzubeugen ist es wichtig, dass sich der Geheimtext bei jeder Verschlüsselung des gleichen Klartextes unterscheidet. Dies wird meist dadurch erreicht, dass die Klartextnachricht vor Anwendung des Verschlüsselungsverfahrens abgeändert wird. Meist wird hierfür eine Nonce (Number used Once) verwendet. Die Nonce kann ein Zähler, ein Zeitstempel oder eine Zufallszahl sein. Der Nonce-Wert muss beiden Seiten bekannt sein. Hierbei kann er entweder mitverschlüsselt übertragen werden oder besteht aus einem Wert (z.B. Zähler), der beiden Seiten bekannt ist.

4.6.1.1 Nonce-Verschlüsselung

Ein Nonce-Wert $v \in \mathscr{M}$ wird dazu genutzt die Klartextnachricht $m \in \mathscr{M}$ abzuändern:

$$\tilde{m} = m + v$$

Anschließend wird die veränderte Nachricht verschlüsselt:

$$c = E_k(\tilde{m})$$

Für die Entschlüsselung wird der Schlüssel k und der Nonce-Wert v benötigt:

$$m = D_k(c) + v$$

Kapitel 5

Stromverschlüsselungsverfahren

Bei einem Stromverschlüsselungsverfahren wird eine Nachricht zeichenweise verschlüsselt. Die Menge der verschlüsselten Zeichen $\mathscr{M}$ ist als endliche Folge von Elementen einer Zeichenmenge $\mathscr{Z}$ definiert.

$$\mathscr{M} = \mathscr{Z}^{>0} := \bigcup_{l \in \mathbb{N}} \mathscr{Z}^l = \{(m_1, m_2, ..., m_l) \mid m_i \in \mathscr{Z}, l \in \mathbb{N}\}$$

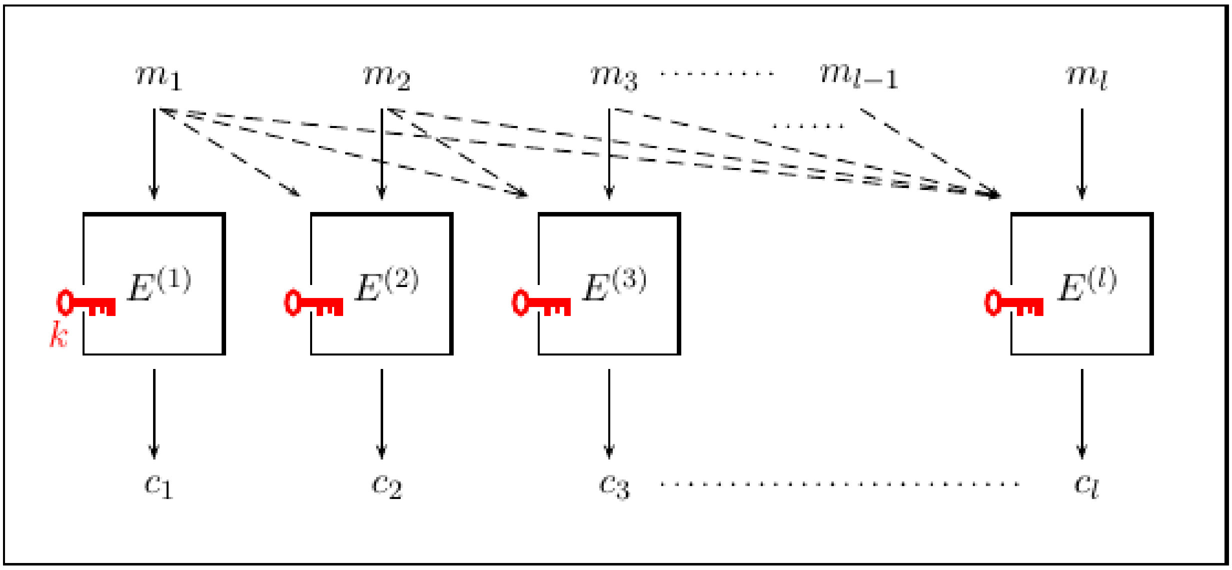
Die Verschlüsselungsfunktion E_k lässt sich zeichenweise beschreiben:

$$E^{(i)} : \mathscr{K} \times \mathscr{Z}^i \to \mathscr{Z} \text{für alle } i \in \mathbb{N}$$

mit:

$$E_k((m_1, m_2, ..., m_l)) = (E^{(1)}(k, (m_1)), E^{(2)}(k, (m_1, m_2)), ..., E^{(l)}(k, (m_1, m_2, ..., m_l)))$$

Bei manchen Verfahren können bei der Berechnung des i-ten Geheimtextzeichens $c_i = E^{(i)}(k, (m_1, m_2, ..., m_i))$ auch die vorherigen Zeichen miteinfließen.



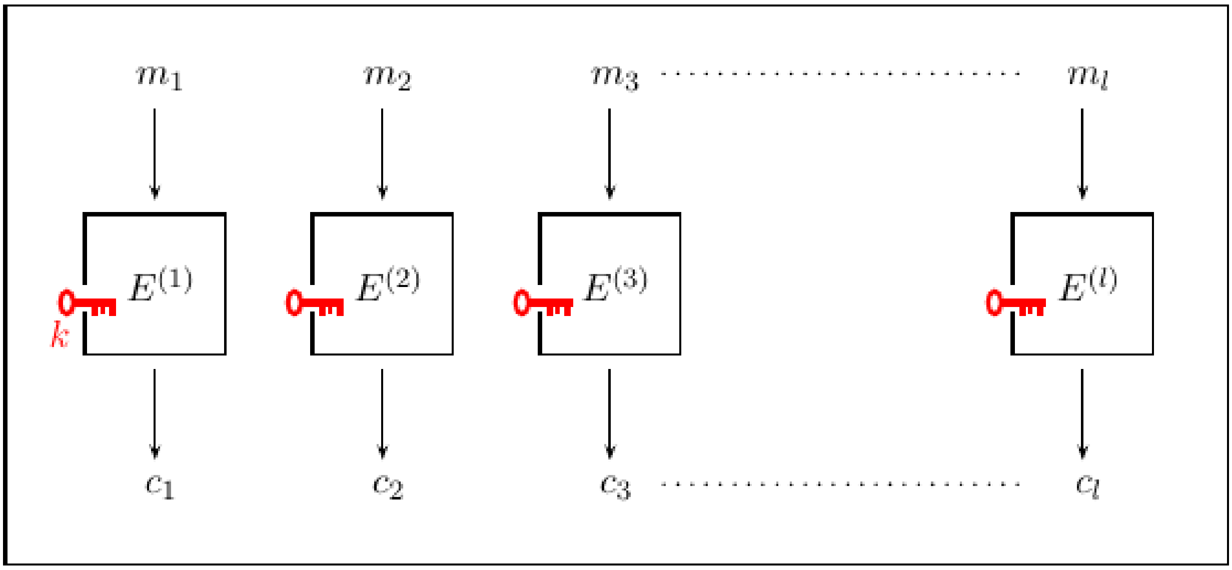
5.1 Synchrone Stromverschlüsselungsverfahren

Bei synchronen Stromverschlüsselungsverfahren hängen die Zeichen einer Geheimtextnachricht nicht von den vorherigen ab. So ist es möglich die einzelnen Zeichen des Klartextes gleichzeitig (synchron) zu verschlüsseln. Ein Beispiel hierfür sind monoalphabetische Verschlüsselungsverfahren (siehe 1.2).

$$E^{(i)} : \mathscr{K} \times \mathscr{Z} \to \mathscr{Z}$$

wobei gelten muss:

$$E_k((m_1, m_2, ..., m_l)) = (E^{(1)}(k, m_1), E^{(2)}(k, m_2), ..., E^{(l)}(k, m_l))$$



5.2 Zustandsabhängige Stromverschlüsselungsverfahren

Bei einer zustandsabhängigen Stromverschlüsselung werden die Zeichen mithilfe eines Zustands verschlüsselt, der sich abhängig vom vorherigen Zeichen ändert. Hierzu werden benötigt:

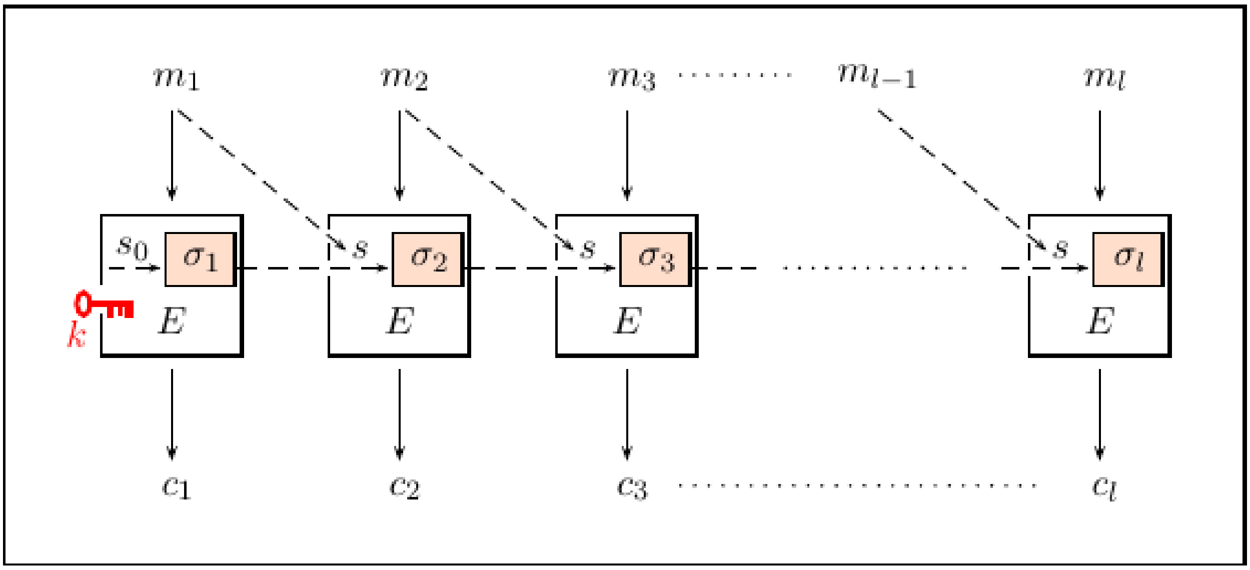
$$\begin{aligned} \mathscr{S} &: \text{Menge der Zustände} \\ s_0 &: \mathscr{K} \to \mathscr{S} \\ s &: \mathscr{S} \times \mathscr{Z} \to \mathscr{S} \\ E &: \mathscr{S} \times \mathscr{Z} \to \mathscr{Z} \text{(bijektiv)} \end{aligned}$$

Mithilfe von s_0 und einem Schlüssel k wird ein Startzustand $\sigma_1 := s_0(k)$ festgelegt. Die folgenden Zustände σ_i werden mithilfe der Abbildung s errechnet:

$$\sigma_i := s(\sigma_{i-1}, m_{i-1})$$

Die einzelnen Zeichen lassen sich dann mithilfe von E in Abhängigkeit von dem jeweiligen Zustand bestimmen:

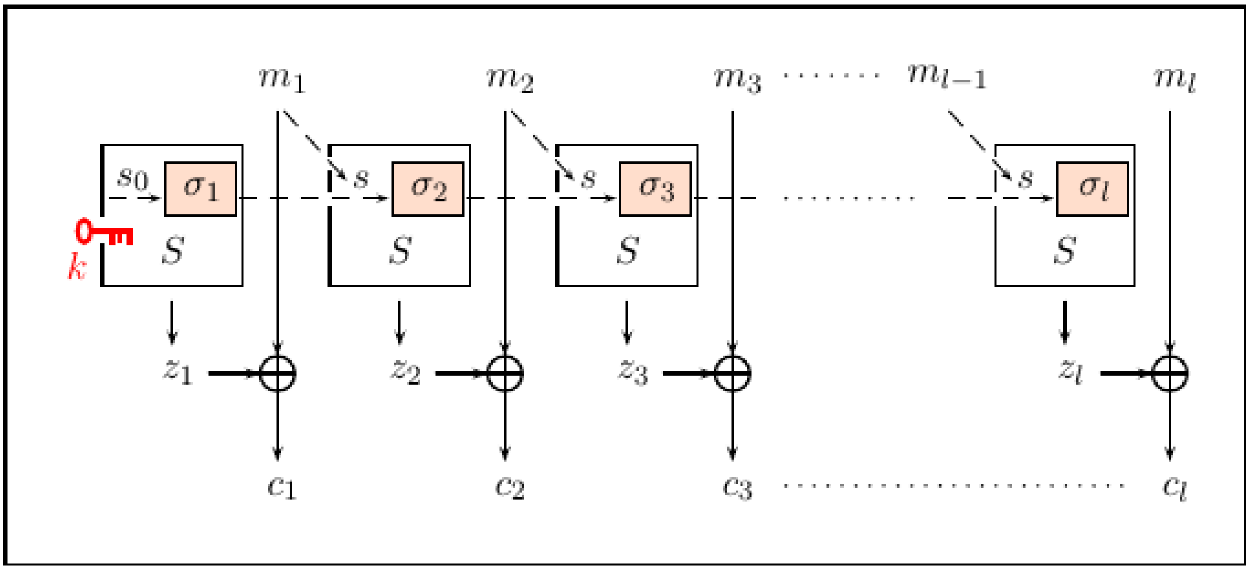
$$c_l := E(\sigma_l, m_l)$$



5.2.1 Additive zustandsabhängige Stromverschlüsselungsverfahren

Additive zustandsabhängige Stromverschlüsselungsverfahren sind zustandsabhängige Stromverschlüsselungsverfahren, mit einer Funktion $S : \mathscr{S} \rightarrow \mathscr{Z}$. Die Verschlüsselungsfunktion ist hierbei durch $E(\sigma, m) = S(\sigma) + m$ definiert, wobei hierbei eine modulare Addition (siehe 2.2.1) über $\mathscr{Z}$ stattfindet. s_0,s und S bilden den Schlüsselstromgenerator, der den folgenden Schlüsselstrom erzeugt:

$$z_i := S(\sigma_i) = S(s(\sigma_{i-1}, m_{i-1}))$$



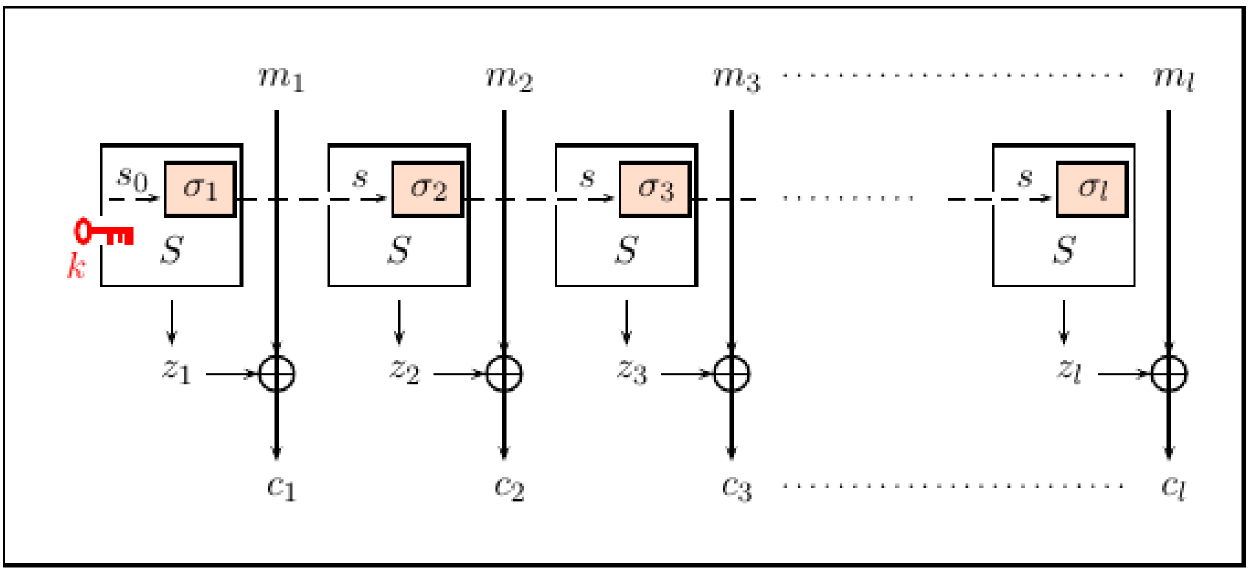
5.2.1.1 Synchrone additive Stromverschlüsselungsverfahren

eine additive Stromverschlüsselung lässt sich synchron durchführen, falls die Übergangsfunktion s nicht von den Zeichen des Klartextes abhängt.

$$\sigma_{i+1} = s(\sigma_i) \text{ mit } s : \mathscr{S} \rightarrow \mathscr{S}$$

Hierdurch definiert sich der Schlüsselstrom:

$$z_i = S(s^{i-1}(s_0(k))) \quad (i \in \mathbb{N})$$



5.3 Schlüsselstrom vs. One-Time-Pad

Ein One-Time-Pad (siehe 1.3.2) ist eine spezielle Form einer additiven Stromverschlüsselung, wobei die Schlüsselstormzeichen $S\sigma_i$ direkt aus dem Schlüssel k entnommen werden. Aufgrund dieser Tatsache kann eine Schlüsselstrom als Ersatz für ein One-Time-Pad betrachtet werden. Hierbei ist allerdings darauf zu achten, dass folgendende Bedingungen erfüllt sind:

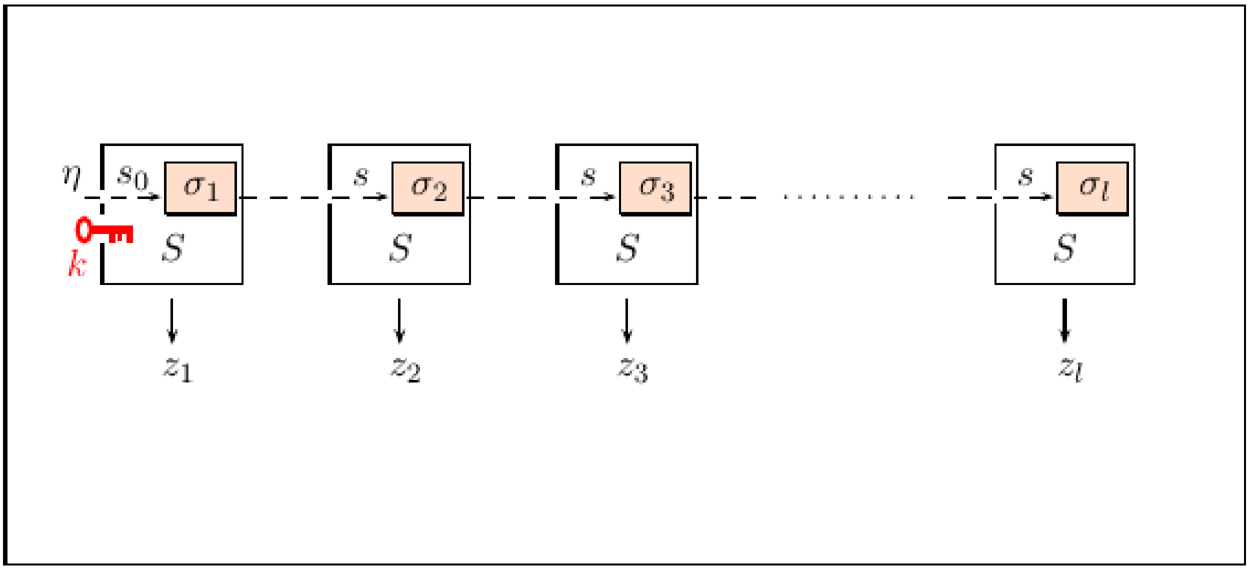
- Schlüsselstromzeichen dürfen keinen Aufschluss über die nachfolgenden Zeichen geben
  - Schlüsselstromzeichen müssen statistisch gleichverteilt und unkorreliert sein
  - ein Schlüssel $k \in \mathscr{K}$ dient gut zur Initialisierung eines Pseudozufallsgenerators
- die Bestimmung des Schlüssels k darf nicht effizienter als mit einer Brute-Force-Suche (siehe 4.5) im Schlüsselraum $\mathscr{K}$ möglich sein.

5.4 Nonces zur Initialisierung eines Schlüsselstromgenerators

Wie bei einem One-Time-Pad (siehe 1.3.2) sollte ein Schlüsselstrom nur einmalig verwendet werden. Da der Austausch von Schlüsseln bei synchronen Verschlüsselungsverfahren allerdings sehr aufwendig ist, wird versucht den Schlüssel weiter zu verwenden. Hierfür gibt es mehrere Möglichkeiten:

- Der Schlüsselstrom wird nicht neu initialisiert und durchgehend weiterverwendet (Schlüsselstrom muss zwischen Sender und Empfänger synchron bleiben).
- Der verwendete Schlüssel wird durch einen Nonce-Wert η_m verändert.
  - η_m muss mit übertragen werden und zwischen Sender und Empfänger synchron gehalten werden
  - Der Schlüssleraum wird kleiner, da er sich in einen Raum $\mathscr{K}_{eff}$, in dem k liegt und einen Bereich $\mathscr{N}$ für den Nonce-Wert aufteilt
- Die Funktion s_0 (siehe 5.2) wird so erweitert, dass sowohl Werte aus $\mathscr{K}$, als auch aus $\mathscr{N}$ einfließen:

$$s_0 : \mathscr{K} \times \mathscr{N} \to \mathscr{S}$$



5.5 ChaCha20

Der ChaCha20-Schlüsselstromgenerator nutzt für die Initialisierung des Zustands s_0 einen Schlüsselwert $k \in \mathscr{K}$ und einen Nonce-Wert $\eta \in \mathscr{N}$. Für die Beschreibung des Algorithmus werden die folgenden Mengen benötigt:

$$\begin{aligned}
&\text{(byte-basiert)} \qquad && \mathscr{Z} = \mathbb{Z}_{2^8} \\
&\text{(int-basiert)} \qquad && \mathscr{K} = (\mathbb{Z}_{2^{32}})^8 \\
&\text{(int-basiert)} \qquad && \mathscr{S} = (\mathbb{Z}_{2^{32}})^{16} \times (\mathbb{Z}_{2^{32}})^{16} \times \mathbb{Z}_{2^{38}+1} \\
&\text{(int-basiert)} \qquad && \mathscr{N} = (\mathbb{Z}_{2^{32}})^3
\end{aligned}$$

Elemente im Zustandsraum $\mathscr{S}$ bestehen aus 3-Tupeln:

$$(s^0, s, c) = (\mathbb{Z}_{2^{32}})^{16} \times (\mathbb{Z}_{2^{32}})^{16} \times \mathbb{Z}_{2^{38}+1}$$

In s^0 werden der Schlüssel k und der Nonce-Wert η mit fest definierten Konstanten nach der Initialisierung abgespeichert. Bei der Initialisierung und nach Ausgabe von jeweils 64 Bytes wird der Wert von s^0 nach s kopiert und in einen Block von 64 Schlüsselstrombytes überführt. c speichert die Anzahl der bereits verwendeten Schlüsselstrombytes (nach Spezifikation max. 2^{38} Bytes)

Initalisierung:

$$s_0 : \mathscr{K} \times \mathscr{N} \to \mathscr{S}$$

daraus ergibt sich:

$$\begin{aligned}
s_0((k, \eta)) =& s_0(((k_1, ..., k_8), (n_1, n_2, n_3))) \\
:=& s(((c_1, c_2, c_3, c_4, k_1, ..., k_8, 0, n_1, n_2, n_3), (0, ..., 0), 0)) \\
=& s((s^0, 0, 0))
\end{aligned}$$

wobei c_i die folgende Konstanten sind:

$$\begin{aligned}
c_1 =& 0x61707865 \\
c_2 =& 0x3320646e \\
c_3 =& 0x79622d32 \\
c_4 =& 0x6b206574
\end{aligned}$$

und

$$s^0 := (c_1, c_2, c_3, c_4, k_1, ..., k_8, 0, n_1, n_2, n_3)$$

Update-Funktion: $s : \mathscr{S} \to \mathscr{S}$

Solange $c < 2^{38}$ gilt:

$$s((s^i, s, c)) := \begin{cases} (s^i, s, c+1) & \text{falls } c \mod 64 \neq 0 \\ (s^{i+1}, s^i \oplus {f_{ib}}^{10}(s^i), c+1) & \text{falls } c \mod 64 = 0 \end{cases}$$

f_{ib} wird 10 mal wiederholt und bildet die folgende Abbildung:

$$f_{ib} : (\mathbb{Z}_{2^{32}})^{16} \to (\mathbb{Z}_{2^{32}})^{16}$$

Die Funktion ist in RFC 8439 definiert.

Extraktion der Schlüsselstromzeichen: $S; \mathscr{S} \to \mathscr{S}$

Mit der Funktion S werden die Bytewerte, aus denen die Zahlen $s_0, s_1, ..., s_{15}$ der zweiten Komponente eines Elements aus $\mathscr{S}$ zusammengesetzt sind, extrahiert:

Für

$$(s^0, s, c) = (s^0, (s_0, s_1, ..., s_{15}), c) \in \mathscr{S}$$

werden Hierzu

$$\begin{aligned}
i &= \lfloor c/4 \rfloor \mod 16 \\
j &= c \mod 4
\end{aligned}$$

berechnet und

$$S((s^0, (s_0, s_1, ..., s_{15}), c)) := \lfloor \frac{s_i}{2^{8j}} \rfloor \mod 2^8$$

gesetzt.

5.6 Cipher-Instanzen: Verschlüsselungsalgorithmen in Java-Laufzeitumgebung

siehe Skript2 1.5 (Seite 19).

Kapitel 6

Blockverschlüsselungsverfahren

Ein Verschlüsselungsverfahren wird als Blockverschlüsselungsverfahren bezeichnet, wenn die Menge der Nachrichten $\mathscr{M}$ durch die Menge der Blöcker einer festen Länge $n \in \mathbb{N}$ gegeben ist:

$$\mathscr{M} := (\mathbb{Z}_2{}^8)^n = \{(z_1, z_2, ..., z_n) \mid z_i \in \mathbb{Z}_2{}^8\}$$

Wichtig

- Häufig wird für Blöcke die aus n Bytewerten die Blocklänge in Bits (Blocklänge= $n \cdot 8$) angegeben
- Die Menge der Blöcke $\mathscr{M}$ kann als Vektorraum über $\mathbb{Z}_2$ aufgefasst werden. Hierbei wird die Summe von zwei Blöcken $m_1, m_2 \in \mathscr{M}$ durch eine bitweise Addition definiert, welcher einer Bitweisen XOR-Verknüpfung ($\oplus$) entspricht.
- Da sich die Vektoren aus $\mathbb{Z}_2{}^{8 \cdot n}$ in Zahlen aus $\mathbb{Z}_{2^{8 \cdot n}}$ umrechnen lassen könnte man sie auch auf diese Art addieren. Hierbei erhält man allerdings ein deutlich anderes Ergebnis als bei XOR-Addition. Aus diesem Grund wird in diesem Fall das Symbol $\boxplus$ verwendet.

6.1 Padding-Verfahren

Damit Daten beliebiger Länge mit einem Blockverschlüsselungsverfahren verschlüsselt werden können muss die Nachricht auf ein Vielfaches der Blocklänge aufgestockt werden. Man spricht von Padding. Um dem Empfänger mitzuteilen, welche übertragenen Daten zum Padding und nicht zur Nachricht gehören gibt es mehrere Möglichkeiten:

- Die Anzahl der Padding-Bytes wird mit übertragen
- Als Paddingbytes werden Zeichen verwendet, die nicht in die Kodierung passen (z.B. 0x00 bei ASCII)
- Es findet immer Padding statt (bei passender Nachrichtenlänge ist der ganze letzte Block Padding), wobei im Padding die Paddinglänge kodiert ist.

6.2 Betriebsmodi

Es gibt eine Vielzahl von Betriebsmodi, die für die Blockverschlüsselung verwendet werden. Auf diese wird im Folgenden eingegangen

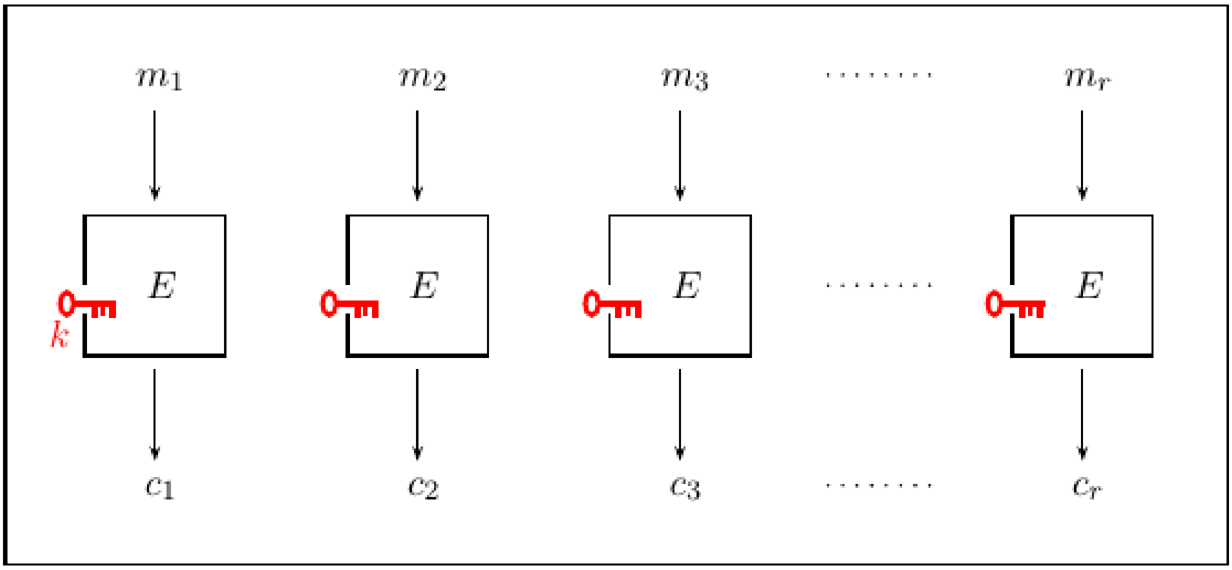
6.2.1 ECB (Electronic Code Book)

Im ECB-Modus wir mit dem Verschlüsselungsverfahren E jedes Tupel von Blöcken blockweise verschlüsselt:

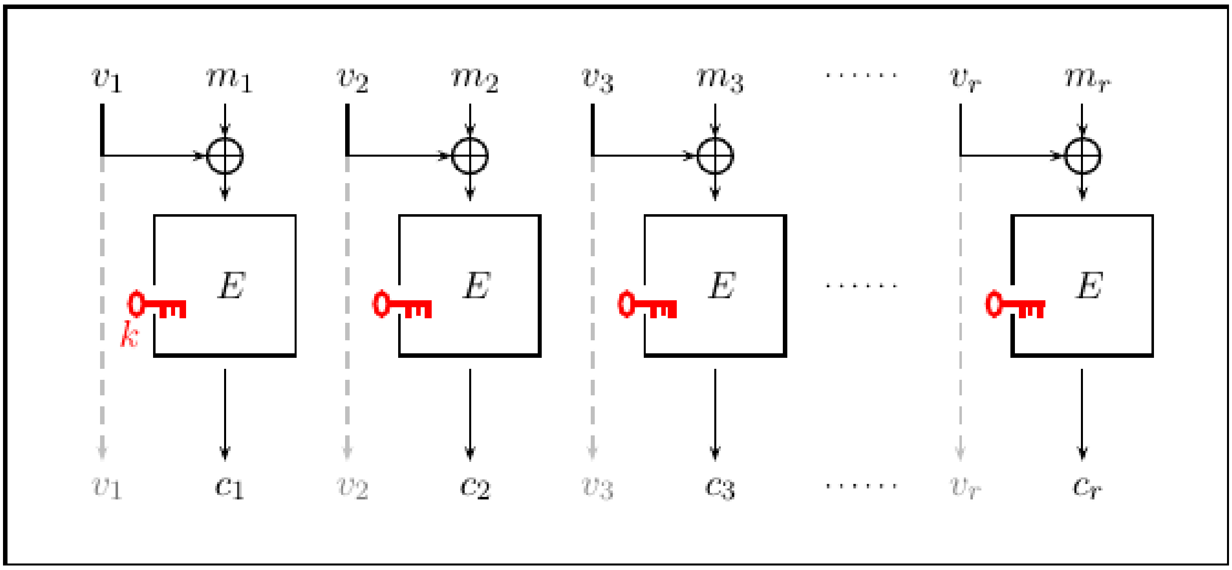
$$E_k((m_1, ..., m_r)) := (E_k(m_1), ..., E_k(m_r))$$

eine auf diese Weise verschlüsselte Nachricht kann ebenfalls Blockweise entschlüsselt werden:

$$D_k((c_1, ..., c_r)) := (D_k(c_1), ..., D_k(c_r))$$



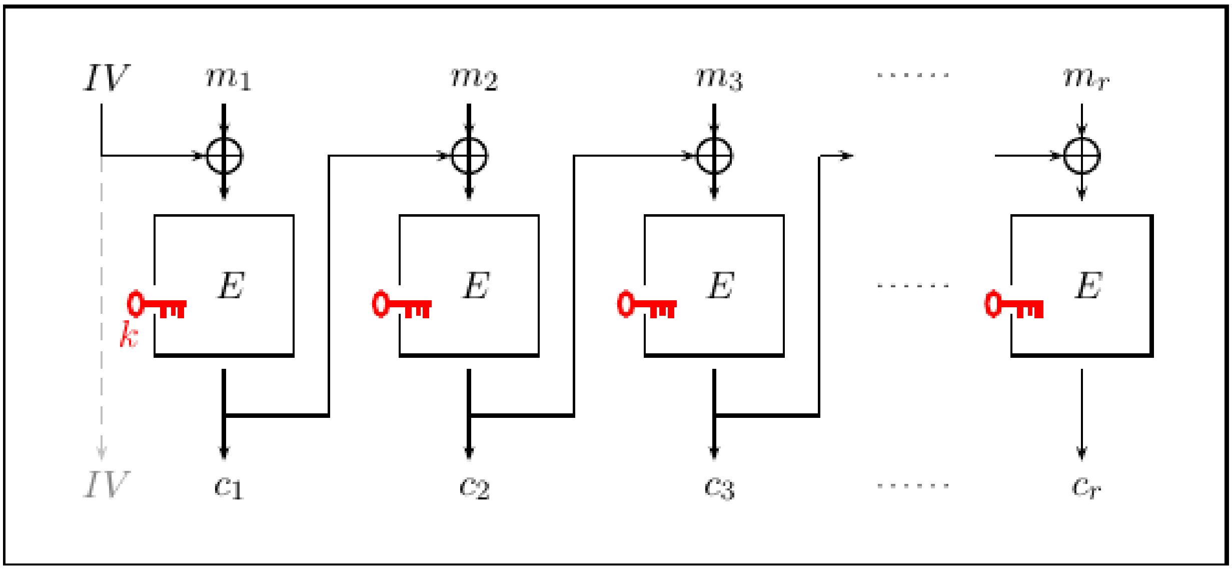
Da diese Modus anfällig für Wörterbuchangriffe ist wird häufig ein Nonce-Wert verwendet, der jeweils mit dem Klartextblock addiert ($\oplus$) wird:



6.2.2 CBC (Cipher Block Chaining)

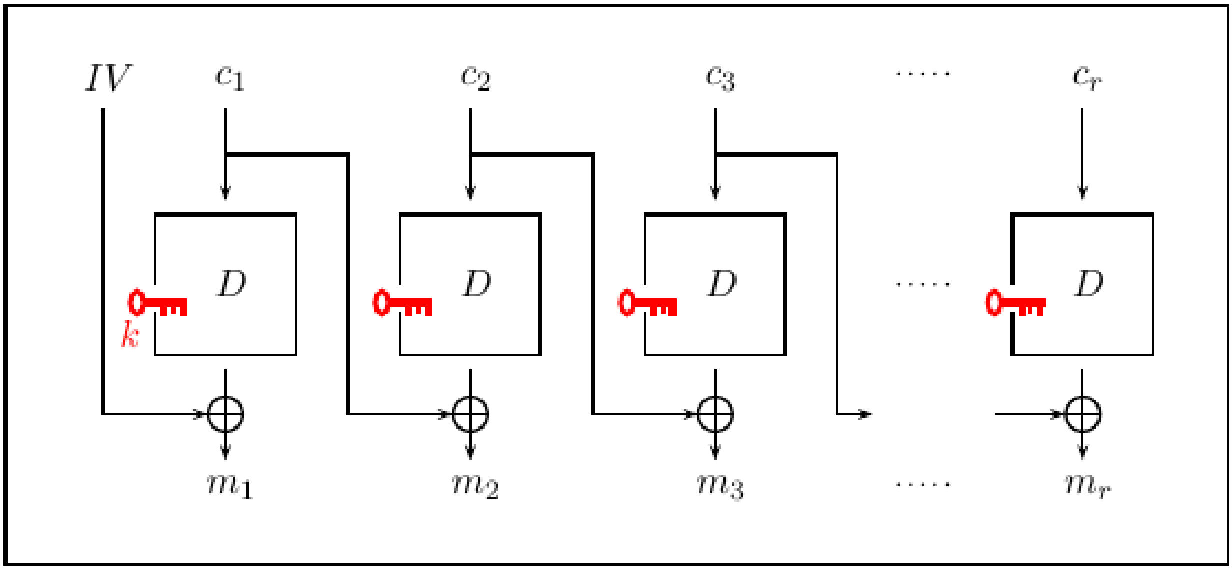
Der CBC-Modus ist eine spezielle Form des ECB-Modus, bei dem der errechnete Geheimtextblock als Nonce-Wert für die Verschlüsselung des nächsten Blocks verwendet wird. Hierbei wird der erste Nonce-Wert c_0 durch einen Initialisierungsvektor $IV \in \mathscr{M}$ gegeben.

$$\begin{aligned} c_i :=& E_k(m_i + c_{i-1}) \\ E_{k,IV}(m_1, ..., m_r) :=& (c_1, ..., c_r) \end{aligned}$$



Bei der Entschlüsselung wird wie folgt vorgegangen:

$$m_i := D_k(c_i) + c_{i-1}$$
$$D_{k,IV}(c_1, ..., c_r) := (m_1, ..., m_r)$$



6.2.3 CBC-CS (Chiphertext Stealing for CBC Mode)

Der CBC-CS-Modus wird auch als CTS-Modus (Chiphertext Stealing Mode) bezeichnet. Dieser Modus basiert auf dem CBC-Modus (siehe 6.2.2), erlaubt aber eine Verschlüsselung von Nachrichten mit beliebiger Länge (ohne Padding). Um das Padding zu umgehen gibt es mehrere Varianten:

6.2.3.1 CBC-CS1

Bei der Variante 1 der CBC-CS-Verschlüsselung wird der letzte Block m_r^* der Nachricht mit 00-Bytes auf die Blocklänge aufgefüllt (00-Padding). Anschließend wird die Nachricht inkl. dem gepaddeten Block m_r mit dem CBC-Verfahren (siehe 6.2.2) verschlüsselt. Um wieder auf die ursprüngliche Länge der Nachricht zu kommen werden aus dem vorletzten Block c_{r-1} des Geheimtextes die gleiche Anzahl Bytes entfernt, die zu m_r^* hinzugefügt wurden. Hierdurch ergibt sich die Geheimtextnachricht $c_1, ..., c_{r-2}, c_{r-1}^*, c_r$, die die gleiche Länge wie m hat.
Wenn die Länge der Nachricht m ein Vielfaches des Blocklänge l ist wird das normale CBC-Verschlüsselungsverfahren angewandt.

6.2.3.2 CBC-CS2

Bei der Variante 2 der CBC-CS-Verschlüsselung wird auf die gleiche Weise vorgegangen, wie bei Variante 1. Allerdings werden, **falls** ein Padding notwendig ist nach der Verschlüsselung und dem Stealing die letzten beiden Blöcke vertauscht.

$$E - CBC - CS2_{k,IV}(m_1, ..., m_{r-1}, m_r^*) := (c_1, ..., c_r, c_{r-1}^*)$$

Wenn die Länge der Nachricht m ein Vielfaches des Blocklänge l ist wird das normale CBC-Verschlüsselungsverfahren angewandt.

6.2.3.3 CBC-CS3

Die Variante 3 der CBC-CS-Verschlüsselung unterscheidet sich nur darin von der Variante 2, dass immer die letzten beiden Blöcke vertauscht werden. Dies geschieht ungeachtet davon, ob die Länge der Nachricht m ein Vielfaches der Blocklänge l ist.

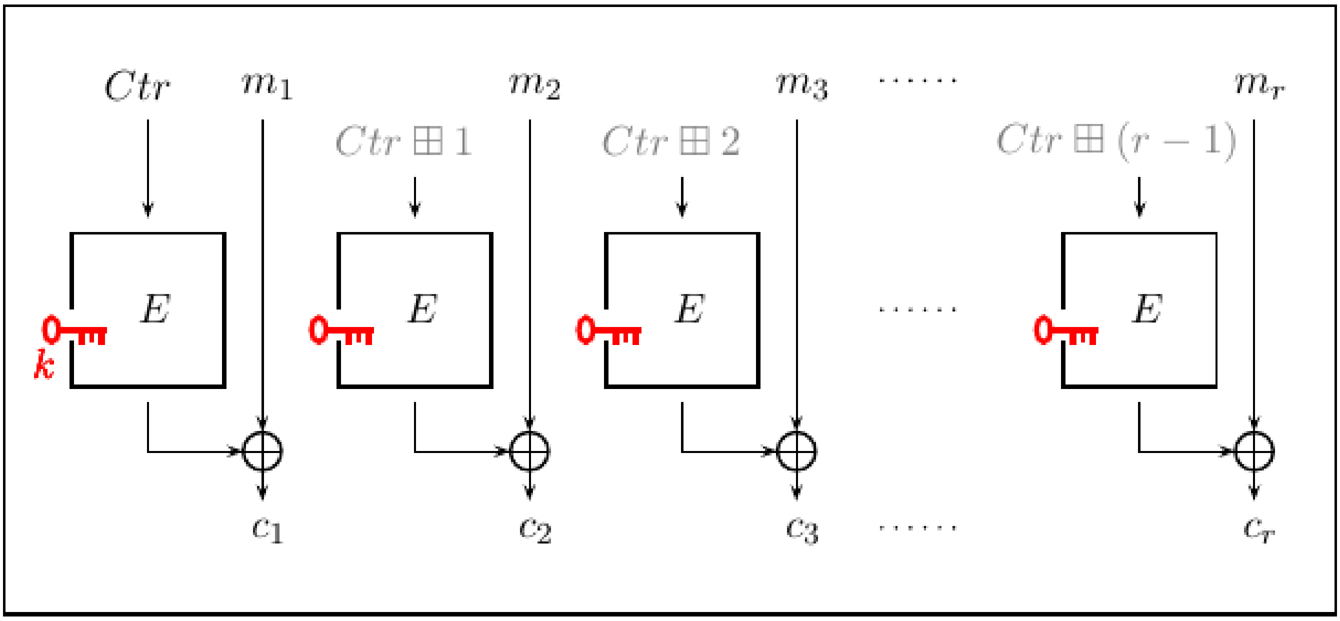
6.2.4 CTR (Counter)

Im CTR-Modus wird ein Blockverschlüsselungsverfahren E mit einem Schlüssel k und einem Nonce-Wert $Ctr \in \mathscr{M}$ wie folgt verwendet:

$$\begin{aligned} c_i :=& m_i + E_k(Ctr \boxplus (i-1)) \\ E - CTR_{k,Ctr}(m_1, ..., m_r) :=& (c_1, ..., c_r) \end{aligned}$$

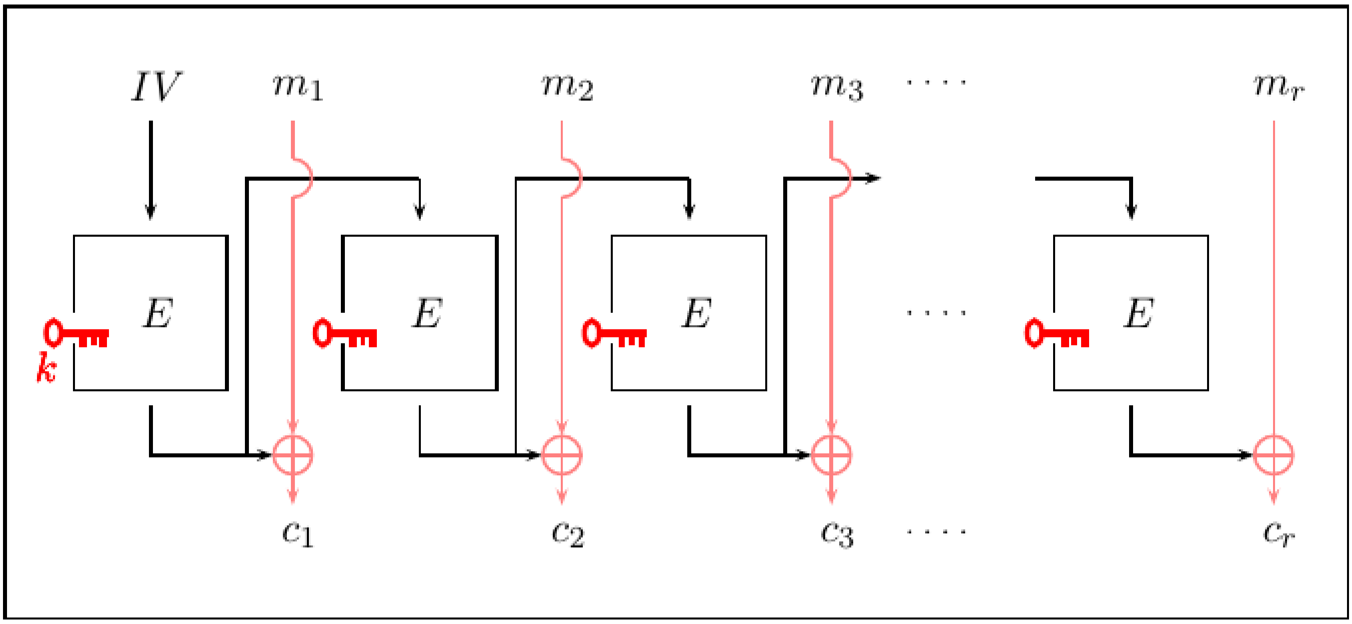
Der CTR-Modus definiert ein synchrones additives Stromverschlüsselungsverfahren (siehe 5.2.1.1). Die Menge der $z_i = E_k(Ctr \boxplus (i-1))$ definiert hierbei den Schlüsselstrom.
Aufgrund dessen ist die Entschlüsselungsfunktion D gleich der Verschlüsselungsfunktion E:

$$D - CTR_{k,Ctr}(c_1, ..., c_r) = E - CTR_{k,Ctr}(c_1, ..., c_r)$$



6.2.5 OFB (Output Feedback)

Bei der Konstruktion im OFB-Modus handelt es sich um ein synchrones Stromverschlüsselungsverfahren (siehe 5.1) Hierbei wird die Ausgabe der Verschlüsselungsfunktion jeweils als Eingabe für die nächste Verschlüsselungsfunktion genutzt:

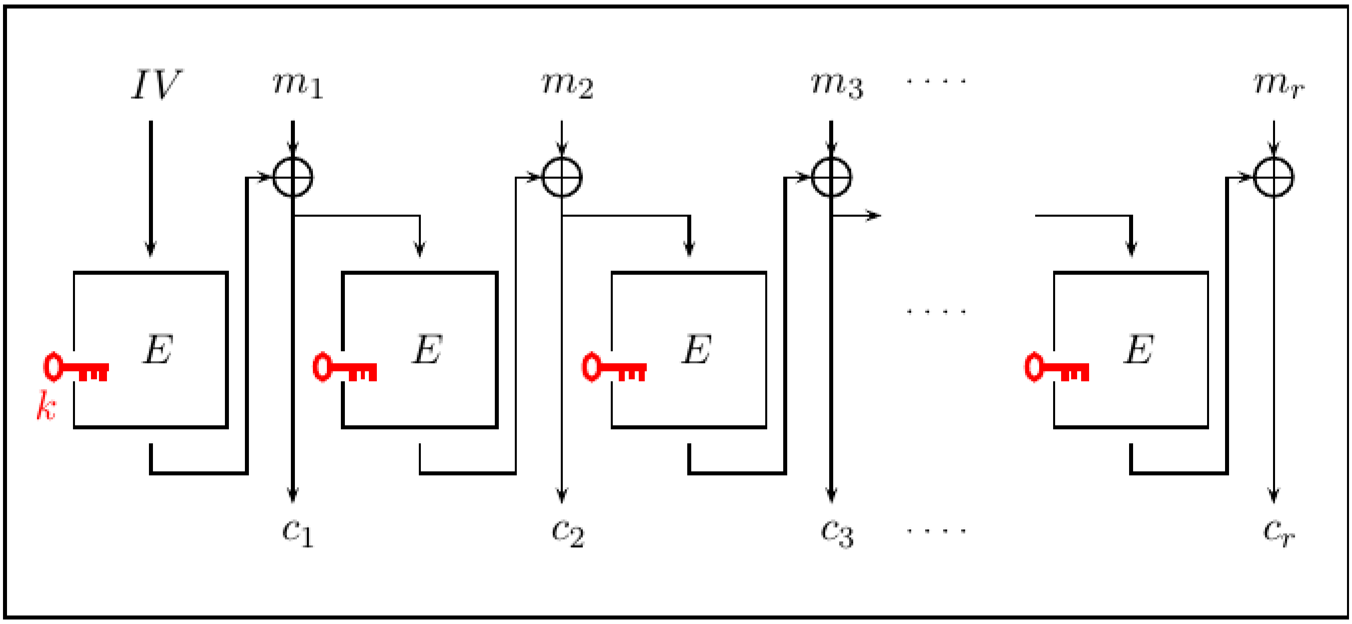


6.2.5.1 OFB-8n

bei dieser Variante des des OFB-Modus werden immer nur $n \in \mathbb{N}$ Bytes des verschlüsselten Blocks in der XOR-Verknüpfung genutzt. Der Rest des Klartextblocks wird in den folgenden Schritten verschlüsselt. n darf hierbei maximal so groß wie die Blocklänge l der Verschlüsselungsfunktion E sein.

6.2.6 CFB (Cipher-Feedback)

Der CFB-Modus hat viele Ähnlichkeiten mit dem OFB-Modus (siehe 6.2.5). Allerdings wird hierbei immer der verschlüsselte Block als Eingabe für die jeweils nächste Verschlüsselungsfunktion genutzt. Anders als beim OFB-Modus handelt es sich beim CFB-Modus aufgrund der Nutzung der Geheimtextblöcke nicht um eine synchrone Stromverschlüsselung.



6.2.6.1 CFB-8n

Diese Sonderfunktion des CFB-Modus stellt das Pendant zum OFB-8n-Modus im OFB-Modus (siehe 6.2.5) dar.

6.3 Konstruktionsprinzipien von Blockverschlüsselungsverfahren

Alle Blockverschlüsselungsverfahren führen innerhalb des jeweiligen Blocks Permutationen aus. Diese Permutationen werden in Schleifen (Runden) angewandt. Die Permutationen können in folgende elementare Operationen zusammengefasst werden:

- *Transpositionen* von Bitwerten innerhalb eines Blocks
- *Substitution* von Werten innerhalb kleinerer Teilblöcke. Hierbei handelt es sich um eine Abbildung, welche durch eine Rechenvorschrift oder eine Wertetabelle gegeben werden kann. Wertetabellen werden in diesem Zusammenhang auch als *S-Boxen* bezeichnet.

Zudem findet der Schlüssel k ebenfalls eine Berücksichtigung. Aus ihm werden Rundenschlüssel k_i mit $i \in \{1, ..., r\}$ abgeleitet, welche bei jeder Runde mit in die Berechnung einfließen. Bei einer Entschlüsselung wird das durch die Runden beschriebene Verfahren einfach rückwärts angewandt.

6.4 DES

Das einfach DES wurde 1977 veröffentlicht und 2005 zurückgezogen. Sowohl die Blöcke als auch die Schlüssel setzten sich jeweils aus 8 Bytes zusammen:

$$\begin{aligned} \mathscr{M} &= (\mathbb{Z}_2{}^8)^8 = \{(m_1, ...m_8) \mid m_i \in \mathbb{Z}_2{}^8\} \\ \mathscr{K} &= \{(k_1, ...k_8) \mid k_i \in \mathbb{Z}_2{}^8, \text{parity}(k_i) = 1\} \subseteq (\mathbb{Z}_2{}^8)^8 \end{aligned}$$

Die Parität ist definiert durch:

$$\mathbb{Z}_2{}^8 \to \mathbb{Z}_2 \quad \text{parity}\left(\sum_{i=0}^{7} b_i \cdot 2^i\right) = \left(\sum_{i=0}^{7} b_i\right) \mod 2$$

Mit anderen Worten: „Die Anzahl der 1-Bits im Byte muss ungerade sein“
Durch die Parität ist der Wert des achten Bits schon festgelegt. Aufgrund dessen ist hat der Schlüsselraum eine Bitlänge von $8 \cdot 7 = 56$. Da dies für moderne PCs mithilfe eines Brute-Force-Angriffs (siehe 4.5) in trivialer Zeit zu knacken ist sollte DES nicht mehr verwendet werden.

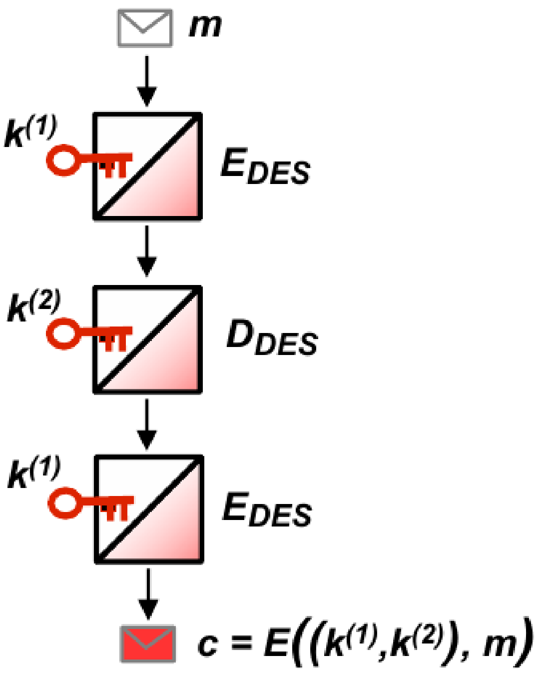
6.4.1 Triple-DES (3DES)

Beim Triple-DES wird das DES verfahren 3mal hintereinander ausgeführt. Üblicherweise im EDE-Modus (Encrypt-Decrypt-Encrypt) mit 16Byte langen Schlüsseln. Dadurch ergibt sich für den Schlüsselraum eine effektive Länge von 112 Bits. Die Menge der 8-Byte-langen Blöcke und der Schlüsselraum definieren sich wie folgt:

$$\begin{aligned} \mathscr{M} &= (\mathbb{Z}_2{}^8)^8 = \{(m_1, ...m_8) \mid m_i \in \mathbb{Z}_2{}^8\} \\ \mathscr{K} &= \{(k_1, ...k_8) \mid k_i \in \mathbb{Z}_2{}^8, \text{parity}(k_i) = 1\}^{2} \end{aligned}$$

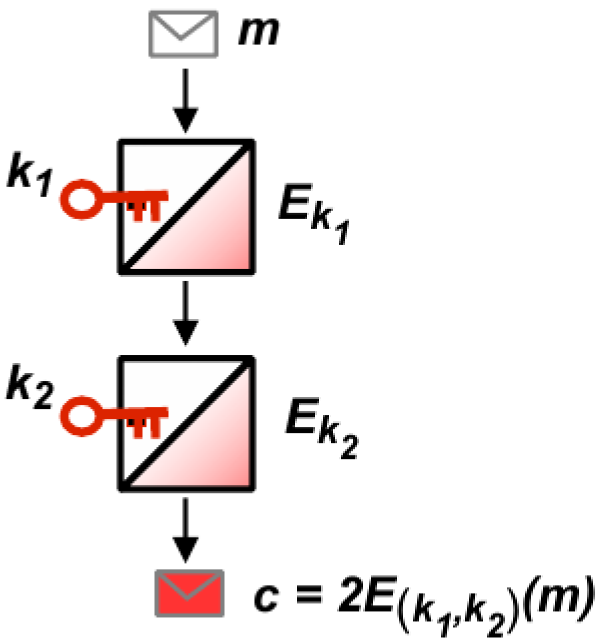
Die 3-DES Entschlüsselungsfunktion definiert sich durch:

$$E((k^{(1)}, k^{(2)}, m)) := E_{DES}\left(k^{(1)}, D_{DES}\left(k^{(2)}, E_{DES}\left(k^{(1)}, m\right)\right)\right)$$



6.5 Meet-in-the-Middle-Angriff

Aufgrund des Meet-in-the-Middle-Angriffs führt eine **zweifache** Hintereinanderreihung von Verschlüsselungsfunktionen nur zu einer kleinen Vergrößerung des Schlüsselraums. Bei diesem Angriff handelt es sich um einen known-plaintext-Angriff (siehe 4.4.2). Es wird angenommen, dass es ein $2E_{k_1,k_2}$ verfahren gibt, dass sich wie folgt definiert:



Es wird zudem angenommen, dass genügend Speicher zur Verfügung steht um alle $\tilde{k}_1 \in \mathscr{K}$ zusammen mit den jeweiligen $c_{\tilde{k}_1}(m_1) := E(\tilde{k}_1, m_1)$ in eine Tabelle zu schreiben:

$\tilde{k}_1$	$c_{\tilde{k}_1}(m_1) := E(\tilde{k}_1, m_1)$
0101010101010101	8A549EC56733AB66
0101010101010102	653148AE6B688132
$\vdots$	$\vdots$
FEFEFEFEFEFEFEFE	CE55464B6485684E

Anschließend wird die Tabelle nach der zweiten Spalte sortiert. Beim Ausprobieren alle Möglichkeiten von $\tilde{k}_2 \in \mathscr{K}$ kann für $D_{\tilde{k}_2}(c_1)$ nachgeschlagen werden, ob sich dieser in der Tabelle befindet. Hierdurch ergeben sich dann $(\tilde{k}_1, \tilde{k}_2)$ Paare, für die gilt:

$$2E_{\tilde{k}_1,\tilde{k}_2}(m_{\color{red}1}) = c_{\color{red}1}$$

Durchschnittlich ist mit $1 + \frac{|\mathscr{K}|}{|\mathscr{M}|}$ solcher Paare zu rechnen. Die Paare können nun durch Ausprobieren der anderen bekannten Nachrichten $\{m_i \setminus m_1\}$ schnell ausgeschlossen werden.

6.6 AES (Advanced Encryption Standard)

Der AES ist für Schlüssel mit den Bitlängen 128, 192 und 256 definiert. Im folgenden wird AES-128 als Synonym für alle möglichen Schlüssellängen verwendet.

6.6.1 AES-128

Sowohl die Blöcke als auch die Schlüssel setzen sich aus jeweils 16 Bytes zusammen:

$$\begin{aligned} \mathscr{M} =& ({\mathbb{Z}_2}^8)^{16} = \{(m_1, ...m_{16}) \mid m_i \in {\mathbb{Z}_2}^8\} \\ \mathscr{K} =& ({\mathbb{Z}_2}^8)^{16} = \{(k_1, ...k_{16}) \mid k_i \in {\mathbb{Z}_2}^8\} \end{aligned}$$

Blöcke werden in Form von Matrizen dargestellt, die als State-Array bezeichnet werden:

$$S = \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} := \begin{pmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m12 & m_{16} \end{pmatrix}$$

Im AES gibt es 4 elementare Operationen auf den State-Arrays:

1. SubBytes()
   Es wird auf Basis der folgenden Tabelle eine Substitution von jedem Element des State-Arrays durchgeführt.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

2. ShiftRows()
   auf dem S-Array wird die folgende Transposition durchgeführt:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \mapsto \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}$$

3. MixColumns()
   auf dem S-Array wird eine Substitution ausgeführt, die durch die folgende Multiplikation definiert ist:

$$S \mapsto M \cdot S \quad \text{mit } M = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 02 & 01 & 02 \end{pmatrix}$$

   Hierbei ist wichtig anzumerken, dass die Multiplikation nicht in $\mathbf{Z}_{256}$ berechnet wird. Stattdessen sind die Elemente als Elemente des Körpers $\mathbf{F}_{256}$ aufzufassen, der bezüglich des irreduziblen Polynoms $m(x) = x^8 + x^4 + x^3 + x + 1 \in \mathbb{F}_2[x]$ definiert ist.

4. AddRoundKey()
   Es wird eine XOR-Operation der Einträge des S-Arrays mit den entsprechenden Einträgen aus **K** durchgeführt. Dies entspricht einer Matrixaddition in $\mathbb{F}_{256}$:

$$\mathbf{S} \mapsto \mathbf{K} + \mathbf{S}$$

AES-128 führt die zuvor beschriebenen Schritte nach dem folgenden Algorithmus aus:

```
AES_ENCRYPT(S,K0,...,K10){
  AddRoundKey(S,K0)
  for( int i = 1; i < 10; ++i){
    SubBytes(S);
    ShiftRows(S);
    MixColumns(S);
    AddRoundKey(S,K0)
  }
  SubBytes(S);
  ShiftRows(S);
  AddRoundKey(S,K0)
}
```

Kapitel 7

Hashfunktionen

Eine Hashfunktion dient als „digitaler Fingerabdruck“ einer Nachricht. Es wird für jede nachricht ein (nahezu) eindeutiger Wert einer festen Länge bestimmt. Eine Abbildung $H : \mathscr{M} \to {\mathbb{Z}_2}^l$, die von der Nachrichtenmenge $mathscrM$ auf eine Bitfolge mit fester Länge l abbildet, wird als Hashfunktion bezeichnet, falls gilt:

1. H ist eine „Einwegsfunktion“ (es gibt keinen effizienten Algorithmus zum Auffinden eines Urbilds)
2. H ist „kollisionsfrei“ (es gibt keinen effizienten Algorithmus zum Finden weiterer Nachrichten mit dem gleichen Hashwert)

Die beiden Eigenschaften stehen in keiner Verbindung zueinander (Einwegsfunktion $\not\Leftrightarrow$ kollisionsfrei). Hashfunktionen haben üblicherweise die folgenden Eigenschaften:

- Die Berechnung der Hashwerte ist effizient möglich und die Berechnungzeit ist im Wesentlichen proportional zur Nachrichtenlänge ($O(n)$)
- $\mathscr{M}$ ist eine Teilmenge von $\mathscr{X}^{\geq 0}$ ($\mathscr{X} = {\mathbb{Z}_2}^8$), wobei die maximale Länge der in $\mathscr{M}$ vorkommenden Bytefolgen durch eine sehr große Zahl $N \in \mathbb{N}$ beschränkt ist ($\mathscr{M} = ({\mathbb{Z}_2}^8)^{\leq N}$). In der Regel ist N so groß, dass alle Nachrichten eine kleinere Länge haben.
- Die Bitlänge l des Hashwertes ist ein Vielfaches von 8.

Hashfunktionen sind wie die meisten Verschlüsselungsverfahren (Ausnahme OTP (siehe 1.3.2)) nicht **beweisbar** sicher. Beispielsweise wurden für MD5 und SHA1 Algorithmen gefunden, die effizienter als durch Brute-Force eine Kollision finden.

7.1 schwache Kollisionsfreiheit

Eine Hashfunktion $H : \mathscr{M} \to {\mathbb{Z}_2}^l$ ist schwach-kollisionsfrei, falls es keinen effizienten Algorithmus gibt, der zu einem beliebigen $m \in \mathscr{M}$ ein $\tilde{m} \in$ findet, für das gilt:

$$\tilde{m} \neq m \text{ und } H(\tilde{m}) = H(m)$$

Sie unterscheidet sich von der starken Kollisionsfreiheit darin, dass m vorgegeben wird.

7.2 `MessageDigest`-Instanzen: Hashfunktionen in Java

siehe Skript 2 Kapitel 4.2 (Seite 56(62)).

7.3 Anwendungsbeispiele

7.3.1 Anwendungsbeispiel: Passwortdatei

Damit ein IT-System erkennt, ob ein vom Nutzer genanntes Passwort korrekt ist muss es Informationen über dieses Passwort abspeichern. Am Einfachsten wäre es alle Passwörter in Plain-Text in einer Datenbank abzuspeichern. Sollte diese Datenbank allerdings gehackt werden könnte der Hacker auf alle Nutzerkonten zugreifen. Um dies zu verhindern müssen die Passwörter auf eine verifizierbare Art und Weise verschlüsselt (gehashed) werden. Dies schützt allerdings nicht vor Wörterbuchangriffen, da es einem Angreifer möglich ist eine sortierte Liste der Hashwerte der häufig verwendeten Passwörter zu erstellen und abzugleichen.

7.3.1.1 Anwendungsbeispiel: Passwortdatei mit Salt und Iteration Count

Zur Abwehr von Wörterbuchangriffen ist es üblich das Passwort p_i mit einem für jeden Nutzer unterschiedlichen Salt-Wert s_i zu verrechnen. Dieser Saltwert wird ebenfalls in der Datenbank abgespeichert und führt dazu, dass jeweils $H(p_i||s_i)$ statt $H(p_i)$ bestimmt wird. Eine Attacke ist somit nur noch Nutzerspezifisch durchführbar. Um dies weiter zu erschweren wird die Hashfunktion häufig mehrmals hintereinander angewandt:

$$H^c(p_i||s_i) = \underbrace{H(H(...H(}_{c} p_i||s_i)...))$$

Die Zahl c wird als Iteration Count bezeichnet. Hierdurch verzögert sich sowohl die reguläre Überprüfung von Passwörtern als auch die Laufzeit eines Angriffs um den Faktor c. Für die Überprüfung ist dies meist kaum relevant (Verfahren wird einmal durchgeführt), während es für den Angriff einen großen Mehraufwand bedeutet (Verfahren wird **sehr** häufig durchgeführt).

7.3.2 Anwendungsbeispiel: Integritätsschutz von Dateien

Um ein Rechnersystem vor der Ausführung von Schadsoftware zu schützen ist es Praxis, die Hashwerte der zur Ausführung zugelassenen Programme in einer Whitelist zu speichern. Ein Angreifer kann Schadsoftware nur dann ausführen, wenn sie den gleichen Hashwert wie ein Programm auf der Whitelist hat.

7.3.3 Anwendungsbeispiel: Integritätsschutz bei einem Dateidownload

Auf Webseiten wird häufig ein Hashwert für eine zu downloadenende Datei angegeben. Dieser lässt sich dann nach dem Download mit der selbst aus der gedownloadeten Datei errechneten Hashwert überprüfen. Dies bietet allerdings nur minimalen Schutz, da ein Angreifer, der die Webseite übernommen hat ebensogut den angegebenen Hashwert ändern kann. Dies lässt sich erreichen, falls die Hashwerte der Datei auf mehreren Webseiten angegeben werden.

7.4 Brute-Force-Angriffe auf Hashfunktionen

Ein Brute-Force-Angriff sucht nach einem Konflikt oder einem Urbild, indem alle möglichen Nachrichten durchprobiert werden.

7.4.1 Brute-Force-Urbildsuche

Ein Angriff auf die Urbildresistenz stellt einen Angriff auf die schwache Kollisionsfreiheit (siehe 7.1) dar. D.h. es wird nach einem zweiten Urbild für ein bekanntes $(m, H(m))$-Paar gesucht. Hierfür werden die Hashwerte für alle $\tilde{m} \in \mathscr{M} \setminus \{m\}$ berechnet und mit $H(m)$ verglichen. Um zu bestimmen, wie viele Versuche r notwendig sind um mit einer Wahrscheinlichkeit p ein $\tilde{m}$ kann die folgende Formel verwendet werden:

$$r \geq \frac{\ln(1-p_0)}{\ln(1-\frac{1}{n})}$$
$$r \geq |\ln(1-p_0)| \cdot n$$

mit $p \geq p_0$, $n = 2^l \geq 2$. Diese Formel ergibt für die folgenden Beispiele:

$$\begin{aligned} r &\approx ln(2) \cdot 2^l && \approx 0,7 \cdot 2^l && \text{für } p = 0,5 \\ r &\approx ln(10) \cdot 2^l && \approx 2,3 \cdot 2^l && \text{für } p = 0,9 \\ r &\approx ln(100) \cdot 2^l && \approx 4,6 \cdot 2^l && \text{für } p = 0,99 \end{aligned}$$

7.4.2 Brute-Force-Kollisionssuche

Ein Angriff auf die starke Kollisionsfreiheit (siehe 7.1), bei der zwei verschiedene Elemente $m_1, m_2 \in \mathscr{M}$ gesucht werden, für die gilt:

$$H(m_1) = H(m_2)$$

Hierbei werden alle $m \in \mathscr{M}$ ausprobiert und die zugehörigen Hashwerte $H(m)$ solange in eine Tabelle geschrieben, bis ein Hashwert doppelt vorkommt. Hierdurch ist die Anforderung an den verfügbaren Speicher nicht zu vernachlässigen. Auch hierfür lässt sich ebenfalls eine Formel errechnen, die vorgibt, wie viele Versuche r zu erwarten sind, bis sich ein Hashwert H mit einer Wahrscheinlichkeit p wiederholt:

$$r \geq \sqrt{-2 \cdot \ln(1-p_0) \cdot n + \frac{1}{4}} + \frac{1}{2} \qquad \text{für } c(n,r) \geq p_0, r \leq \sqrt{2n}$$

Dies ergibt beispielhaft für die Kollisionswahrscheinlichkeit $p = c(n,r) = \frac{1}{2}$ die zwei mögliche Ungleichungen:

$$r \geq \sqrt{\ln(4)} \cdot \sqrt{n} + 1$$
$$r \geq \sqrt{\ln(4) \cdot n + \frac{1}{4}} + \frac{1}{2}$$

7.5 Konstruktionsverfahren von Hashfunktionen

Im allgemeinen Bestehen Hashfunktionen aus 2 Teilen:

1. eine iterative Anwendung einer **Kompressionsfunktion** $f : \mathbb{Z}_2{}^{r+s} \to \mathbb{Z}_2{}^s$ auf r Bits von m und ein internes Zustandsregister s
2. eine einmalige Ausführung einer **Ausgabefunktion** $g : \mathbb{Z}_2{}^s \to \mathbb{Z}_2{}^l$

Bei vielen Verfahren ist $l = s$ und es ist keine Ausgabefunktion nötig.
Mithilfe von f, g und einem Initialisierungswert $t_0 \in \mathbb{Z}_2{}^s$ für das Zustandsregister wird $H(m)$ wie folgt berechnet:

1. m wird durch ein Padding-Verfahren so erweitert, dass für $l(m||\text{pad}) = l(m) + l(\text{pad})$ gilt: $r|l(m||\text{pad})$
2. m wird in Blöcke $m_1, ..., m_n$ zerlegt, die alle r Bit lang sind.
3. der Wert des Zustandsregisters t wird durch iteratives Anwenden der Kompressionsfunktion f auf den jeweiligen Block m_i berechnet:

   $$t_i := f(m_i||t_{i-1}) \qquad \text{für } i = 1, ..., n$$

4. der Hashwert wird ausgegeben: $H(m) := g(t_n)$

Die unterschiedlichen Hashfunktionen unterscheiden sich in der Bitlänge der Hashwerte, Blöcke und Zustandsregister:

H	$l = l(H(m))$	$r = l(m_i)$	$s = l(t_i)$	$\min\{l(\text{pad})\}$
MD5	128	512	128	65
SHA-1	160	512	160	65
SHA-224	224	512	256	65
SHA-256	256	512	256	65
SHA-512/224	224	1024	512	129
SHA-512/256	256	1024	512	129
SHA-384	384	1024	512	129
SHA-512	512	1024	512	129
SHA3-224	224	1152	1600	4
SHA3-256	256	1088	1600	4
SHA3-384	384	832	1600	4
SHA3-512	512	576	1600	4

Kapitel 8

MAC-Verfahren

Der *Message Authentication Code* (MAC) stellt einen „digitalen Fingerabdruck“ einer Nachricht dar. Dieser hängt von der Nachricht m und einem Schlüssel k ab. Die Abbildung $M : \mathscr{K} \times \mathscr{M} \to {\mathbb{Z}_2}^l$ mit einem konstanten $l \in \mathbb{N}$ wird als MAC-Verfahren bezeichnet, falls ohne einen Schlüssel $k \in \mathscr{K}$ die Berechnung des MAC-Wertes $t = M_k(m) := M(k, m)$ für kein $m \in \mathscr{M}$ möglich ist. Dies soll auch dann nicht möglich sein, falls andere (m_i,t_i)-Paare bekannt sind.

8.1 HMAC

Jede Hashfunktion (siehe 7) H lässt sich in ein MAC-Verfahren $M = \text{HMAC-}H$ umwandeln durch:

$$M_k(m) := H\left((k \oplus \text{opad})||H((k \oplus ipad)||m)\right)$$

opad und ipad sind fest definierte Bytearrays der Bytelänge $r/8$.

8.2 CMAC, CBC-MAC

Im Folgenden wird auf zwei MAC-Konstruktionen eingegangen, die auf der Basis von Blockverschlüsselungsverfahren (siehe 6) definiert sind:

8.2.1 CMAC

Die Berechnung des MAC-Wertes $M_k(m) = M(k, m)$ mithilfe eines Blockverschlüsselungsverfahrens $E : \mathscr{K} \times ({\mathbb{Z}_2}^8)^n \to ({\mathbb{Z}_2}^8)^n$ erfolgt, indem die durch ein Bytearray gegebene Nachricht m wie folgt verändert wird:

1. m wird durch ein Padding von der Form „80 00 ... 00“ auf ein Vielfaches der Blocklänge n gebracht:
$$\tilde{m} := \begin{cases} m & \text{falls } l(m) \mod n = 0 \text{ und } l(m) \neq 0 \\ m|| \underbrace{8000...00}_{n-(l(m) \mod n)} & \text{falls } l(m) \mod n \neq 0 \text{ oder } l(m) = 0 \end{cases}$$
2. $\tilde{m}$ wird in Blöcke der Länge n zerlegt
3. der letzte Block $\tilde{m}_r$ wird wie folgt XOR-Verschlüsselt:
$$m_r := \begin{cases} \tilde{m}_r \oplus k_1 & \text{falls } l(m) \mod n = 0 \text{ und } l(m) \neq 0 \\ \tilde{m}_r \oplus k_2 & \text{falls } l(m) \mod n \neq 0 \text{ oder } l(m) = 0 \end{cases}$$
k_1 und k_2 sind Modifikationen von $k_0 := E_k(0)$
4. Die Blöcke inkl. des Verschlüsselten Blocks m_r werden mit $IV = 0$ CBC-Verschlüsselt (siehe 6.2.2)
5. Das Ergebnis wird als MAC-Wert ausgegeben

8.2.2 CBC-MAC

Falls die Nachrichtenmenge nur Nachrichten mit einer festen Bytelänge, die ein Vielfaches der Blocklänge ist, enthält, kann das CBC-Verfahren (siehe 6.2.2) direkt als MAC-Verfahren verwendet werden.

Kapitel 9

Modulare Arithmetik - Teil 2

9.1 Potenzen

Potenzen sind für beliebige Zahlen $a \in \mathbb{Z}_n$ für natürliche Exponenten $i \in \mathbb{N}$ wie üblich definiert:

$$a^i := \underbrace{a \cdot a \cdots a}_{i}$$

Ist a invertierbar ($a \in \mathbb{Z}_n^*$) so sind die Potenzen für alle ganzzahligen Exponenten $i \in \mathbb{Z}$ definiert, indem $a^0 = 1$ und $a^{-i} := (a^{-1})^i$ definiert werden. Folglich gelten die Folgenden Regeln:

für $a, b \in \mathbb{Z}_n$ und $i, j \in \mathbb{N}$ oder für $a, b \in \mathbb{Z}_n^*$ und $i, j \in \mathbb{Z}$ gilt:

$$a^i \cdot a^j = a^{i+j}$$
$$(a^i)^j = a^{ij}$$
$$a^i \cdot b^i = (ab)^i$$

9.1.1 Erzeugnis und Ordnung eines invertierbaren Elements

Die zyklische Untergruppe, die aus den Potenzen eines Elementes $a \in \mathbb{Z}_n^*$ entsteht wird als Erzeugnis von a in $\mathbb{Z}_n^*$ bezeichnet.

$$< a > := \{a^i \mid i \in \mathbb{Z}\}$$

Die Anzahl der verschiedenen Potenzen von a wird als Ordnung von a in $\mathbb{Z}_n^*$ bezeichnet:

$$o_n(a) := | < a > |$$

Hierbei ergibt sich $a^{o(a)} = 1$ und $< a > = \{1, a, a^2, a^3, ..., a^{o(a)-1}\}$.

für $n \in \mathbb{N}(n \geq 2); a, b \in \mathbb{Z}_n^*$ und $i \in \mathbb{Z}$ gilt:

(i) $a^i = 1 \Longrightarrow o(a) | i$

(ii) $o(a^i) | o(a)$

(iii) $i | o(a) \Longrightarrow o(a^i) = \frac{o(a)}{|i|}$

(iv) $\text{ggT}(o(a), i) = 1 \Longrightarrow o(a^i) = o(a)$

(v) $o(ab) | o(a) \cdot o(b)$

(vi) $\text{ggT}(o(a), o(b)) = 1 \Longrightarrow o(a \cdot b) = o(a) \cdot o(b)$

Wenn μ_n für $n \in \mathbb{N}, n \geq 2$ die maximale Ordnung eines Elementes aus $\mathbb{Z}_n^*$ ($\mu_n := \max\{o(a) \mid a \in \mathbb{Z}_n^*\}$), ist jede Ordnung von a ein Teiler von μ_n

9.1.2 Faktorenzerlegung

Falls es für ein Polynom $p(x) \in \mathbb{Z}_n[x]$ ($n \in \mathbb{N}$) vom Grad k eine Nullstelle $a \in \mathbb{Z}_n$ gibt, muss es ein Polynom $q(x) \in \mathbb{Z}_n[x]$ vom Grad $k-1$ geben, für das gilt:

$$p(x) = (x - a) \cdot q(x)$$

9.1.3 Kleiner Satz von Fermat

Für eine Primzahl p gilt für alle $a \in \mathbb{Z}$ mit $\mathrm{ggT}(a, p) = 1$:

$$a^{p-1} \equiv 1 \pmod p$$

9.1.4 Berechnung modularer Potenzen

Es gibt zwei verschiedene Varianten zur effizienten Berechnung modularer Potenzen, die sich beide an der Darstellung des Exponenten e im Stellenwertsystem zur Basis 2 (binär) orientieren: Es lässt sich ein Algorithmus aufstellen, bei dem sukzessive die Quadrate $a^2, a^4 = (a^2)^2, a^8 = (a^4)^2, ..., a^{2^{l-1}} = \left(a^{2^{l-2}}\right)^2$ in $\mathbb{Z}_n$ berechnet werden. Diese fließen als Faktor in die Berechnung von $p = a^e$ ein, falls das entsprechende Bit $e_i = 1$ ist. Es folgen die beiden Algorithmen:

Algorithmus 1
Sei $n \in \mathbb{N}, a \in \mathbb{Z}_n$ und $e \in \mathbb{N}$ (in binärer Schreibweise):

$$e = (e_{l-1}...e_2e_1e_0)_2 = \sum_{i=0}^{l-1} e_i \cdot 2^i$$

Für die Berechnung von $p = a^e \mod n$ wird der folgende Algorithmus angewandt:

(1) $i := 0, p := 1$ und $s := a$

(2) $\mathrm{if}(e_i = 1) p := p \cdot s \mod n$

(3) $\mathrm{if}(i = l - 1)$beende den Algorithmus und gebe p aus

(4) $i := i + 1$

(5) $s := s^2 \mod n$

(6) goto (2)

Algorithmus 2
Sei $n \in \mathbb{N}, a \in \mathbb{Z}_n$ und $e \in \mathbb{N}$ (in binärer Schreibweise):

$$e = (e_{l-1}...e_2e_1e_0)_2 = \sum_{i=0}^{l-1} e_i \cdot 2^i$$

Für die Berechnung von $p = a^e \mod n$ wird der folgende Algorithmus angewandt:

(1) $i := l - 1, p := 1$ und $s := a$

(2) $\mathrm{if}(e_i = 1) p := p \cdot a \mod n$

(3) if($i = 0$)beende den Algorithmus und gebe p aus

(4) $i := i - 1$

(5) $p := p^2 \mod n$

(6) goto (2)

9.2 Exkurs: Einheitengruppe $\mathbb{Z}_{p^e}^*$

Für eine Primzahl p gibt es ein Element $g \in \mathbb{Z}_p^*$ mit $o(g) = p - 1$ ($\mathbb{Z}_p^* =< g >= \{1, g, g^2, ..., g^{p-2}\}$). Diese Elemente g werden als **primitive Elemente** bezeichnet.
Mithilfe der Potenzen der primitiven Elemente lassen sich alle Elemente von $\mathbb{Z}_p^*$ generieren:

$$< g >= \mathbb{Z}_p^*$$

Diese Elemente werden daher auch als **Generatoren** von $\mathbb{Z}_p^*$ bezeichnet. Auch für Einheitengruppen $\mathbb{Z}_{p^e}$ für Primzahlpotenzen mit $e > 1$ gibt es solche zyklische Generatoren:

für eine Primzahl $p \neq 2$ und $e \in \mathbb{N}$ gibt es ein $g \in \mathbb{Z}_{p^e}^*$ für das gilt:

$$\begin{aligned} o(g) = p^{e-1}(p-1) = \varphi(p^e) = |\mathbb{Z}_{p^e}^* \\ \Leftrightarrow \qquad < g > = \mathbb{Z}_{p^e}^* \end{aligned}$$

Hierbei gilt:

Für ein primitives Element $g \in \mathbb{Z}_p^*$ gilt mit $a := g^{\frac{o_{p^e}(g)}{p-1}}$:

$$\begin{aligned} \mathbb{Z}_{p^e}^* &=< a > \cdot < 1 + p > \\ &: = \{a^i \cdot (1+p)^j \mid i = 0, 1, ..., p-2; j = 0, 1, ...p^{e-1} - 1\} \end{aligned}$$

9.2.1 $\mathbb{Z}_{2^e}^*$

Die Primzahl 2 stellt eine besonderheit dar, da $\mathbb{Z}_{2^e}^*$ für $e \geq 3$ nicht zyklisch ist. So gilt $o(5) = 2^{e-2}$ und allgemein:

$$\begin{aligned} \mathbb{Z}_{2^e}^* &=< 2^e - 1 > \cdot < 5 > \\ &= \{(2^e - 1)^i \cdot 5^j \mid i = 0, 1; j = 0, 1, ..., 2^{e-2} - 1\} \end{aligned}$$

9.3 Der chinesische Restsatz

Für eine Menge teilerfremder Zahlen $\{(n_1, ..., n_r) \mid n_i \in \mathbb{N}\}$ bildet einen Menge $\{(a_1, ..., a_r) \mid a_i \in \mathbb{Z}\}$ ein System aus Kongruenzen:

$$\begin{aligned} x \equiv& a_1 \pmod{n_1} \\ x \equiv& a_2 \pmod{n_2} \\ &\vdots \\ x \equiv& a_r \pmod{n_r} \end{aligned}$$

Die Menge der ganzzahligen Lösungen ist gegeben durch:

$$\mathbb{L} = \{a + v \cdot n_1 n_2 \cdots n_r \mid v \in \mathbb{Z}\}$$

Hierbei ist $0 \leq a < n_1 n_2 \cdots n_r$ eindeutig. a wird im folgenden Beispiel mithilfe des erweiterten Euklid'schen Algorithmus (siehe 2.3.2) berechnet. Aus dem Restsatz leitet sich folgender Satz ab:

Sind $n_1, n_2, ..., n_r$ paarweise teilerfremde Zahlen gibt es die folgende Isomorphie (2.2.7):

$$\mathbb{Z}_{n_1 n_2 \cdots n_r} \cong \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times \cdots \times \mathbb{Z}_{n_r}$$

9.3.1 Beispiel

folgende Kongruenzen sind gegeben:

$$\begin{aligned} x \equiv& 2 \pmod 5 \\ x \equiv& 7 \pmod 2 \\ x \equiv& 2 \pmod 5 \end{aligned}$$

- **Bestimmung von e_1**
  für $n_1 = 5$ und $m_1 = 21 \cdot 11 = 231$ liefert der erweiterte Euklid'sche Algorithmus (2.3.2):
  $$(-46) \cdot 5 + 1 \cdot 231 = 1$$
  Daher ist $e_1 = 1 \cdot 231$
- **Bestimmung von e_2**
  für $n_2 = 21$ und $m_2 = 5 \cdot 11 = 55$ liefert der erweiterte Euklid'sche Algorithmus (2.3.2):
  $$(-21) \cdot 21 + (-8) \cdot 55 = 1$$
  Daher ist $e_2 = (-8) \cdot 55 = -440$
- **Bestimmung von e_3**
  für $n_3 = 11$ und $m_3 = 5 \cdot 21 = 105$ liefert der erweiterte Euklid'sche Algorithmus (2.3.2):
  $$(-19) \cdot 11 + 2 \cdot 105 = 1$$
  Daher ist $e_3 = 2 \cdot 105 = 210$
- **Berechnung der Lösung**
  $n = 5 \cdot 21 \cdot 11 = 1155$:
  $$\begin{aligned} a =& (2 \cdot e_1 + 7 \cdot e_2 + 6 \cdot e_3) \mod 1155 \\ =& (2 \cdot 231 - 7 \cdot 440 + 6 \cdot 210) \mod 1155 \\ =& -1358 \mod 1155 \\ = 952 \end{aligned}$$

9.4 Elemente gerader und ungerader Ordnung in $\mathbb{Z}_n$**

siehe Skript 3 Kapitel 1.4 auf Seite 17(23).

Kapitel 10

Das Diffie-Hellman-Schlüsselaustauschverfahren

In allen Fällen, in denen ein symmetrisches Schlüsselverfahren (inkl. MAC-Verfahren) verwendet wird ist es nötig den Schlüssel sicher auszutauschen. Eine Möglichkeit ist es diese Übertragung über einen sicheren Kanal (z.B. persönliche Übergabe) zu organisieren. Dies ist allerdings mit einem hohen Aufwand verbunden und erlaubt nicht einen sicheren Kanal in Echtzeit (z.B. für HTTPS) aufzubauen. Hierfür wurde von W.Diffie udn M.E.Hellman auf Basis der Arbeit von R.C.Merkle ein asymmetrisches Schlüsselaustauschverfahren entwickelt, welches als DH-Verfahren bekannt ist. Asymmetrische Verschlüsselungsverfahren zeichnen sich durch die Verwendung von **Schlüsselpaaren** ($k = (k_{\text{priv}}, k_{\text{pub}})$) aus. Diese Schlüsselpaare sollten folgende Eigenschaften haben:

- Ein Schlüsselpaar ist genau einem Besitzer zugeordnet
- Niemals hat jemand anderes als der Besitzer zugriff auf den privaten Schlüssel k_{priv}
- Der private Schlüssel wird vom Besitzer selber generiert
- Der öffentliche Schlüssel k_{pub} lässt sich effizient aus k_{priv} errechnen
- Es gibt keinen effizienten Algorithmus um mithilfe von k_{pub} k_{priv} zu bestimmen

Vor dem Austausch eines symmetrischen Schlüssels besitzen beide Seiten jeweils ein unabhängiges asymmetrisches Schlüsselpaar.

10.1 Das DH-Verfahren in Einzelschritten

Im Folgenden werden Alice und Bob als Synonyme für zwei beliebige Sender/Empfänger genutzt:

- Alice und Bob vereinbaren zwei Parameter (meist durch einen Standard spezifiziert):
  - eine große Zahl $n \in \mathbb{N}$
  - eine Zahl $g \in \mathbb{Z}_n^*$ deren Ordnung $o(g)$ mindestens durch eine sehr große Primzahl geteilt wird.
- Alice bestimmt eine zufällig Zahl $e_A \in \mathbb{Z}_n^*$ als privaten Schlüssel:

  $$k_{A,\text{priv}} = e_A$$

  und errechnet den öffentlichen Schlüssel $k_{A,\text{pub}} = g^{e_A} \in \mathbb{Z}_n$
- Bob bestimmt ebenfalls eine Zufallszahl $e_B \in \mathbb{Z}_n^*$ und damit ein Schlüsselpaar:

  $$(k_{B,\text{priv}}, k_{B,\text{pub}}) = (e_B, g^{e_B})$$

- Alice und Bob tauschen ihre öffentlichen Schlüssel aus.
- Alice berechnet in $\mathbb{Z}n$:
$$s_A := (g^{e_B})^{e_A}$$
Bob berechnet in $\mathbb{Z}n$:
$$s_B := (g^{e_A})^{e_B}$$
- Der gemeinsame Schlüssel (shared secret) ergibt sich durch:
$$s_A := (g^{e_B})^{e_A} = g^{e_B \cdot e_A} = g^{e_A \cdot e_B} = (g^{e_A})^{e_B} = s_B$$

10.2 Das Diskrete-Logarithmus-Problem (DL-Problem)

Das Diskrete-Logarithmus-Problem bildet die mathematische Grundlage dafür, dass Schlüssel mittels der DH-Verfahrens über einen potentiell unsicheren Kanal übertragen werden können. Das Problem definiert sich wie folgt:

DL-Problem
Gegeben seien $n \in \mathbb{N}, g \in \mathbb{Z}_n^*$ und eine Potenz von g in $\mathbb{Z}_n$:
$$a \in \langle g \rangle := \{g^e \mid e \in \mathbb{N}\}$$
Als DL-Problem wird die Aufgabe bezeichnet nur mittels a und g einen passenden Exponenten e für $a = g^e$ zu finden.
Es wird vermutet, dass es keinen effizienten Algorithmus gibt.

10.2.1 Das Diffie-Hellman-Problem (DH-Problem)

Das DH-Problem ist eine Variante des DL-Problems, bei der zwei Potenzen $g^{e_A}, g^{e_B} \in \langle g \rangle$ gegeben sind. Hierbei wird die Zahl $g^{e_A \cdot e_B}$ gesucht. Auch hier wird vermutet, dass es keinen effizienten Algorithmus gibt.

10.3 Beispielanwendung des DH-Verfahrens

Realistische Beispiele mit Zahlen, die mehrere hundert Dezimalstellen besitzen, können natürlich nur mit einem Computer durchgerechnet werden. Prinzipiell lassen sich die Rechnungen jedoch mit kleinen Zahlen nachvollziehen:

Alice und Bob wollen einen Schlüssel (shared secret) mithilfe des DH-Verfahrens vereinbaren. Hierzu verabreden sie, die „DH-Parameter“

$$(n,g) = (47,2)$$

zu verwenden. (Beachten Sie, dass n eine sichere Primzahl ist.)

Alice benutzt einen Zufallszahlengenerator um ihren privaten Schlüsselwert zu erzeugen. Dieser sei beispielsweise:

$$k_{A,priv} = e_A = 17$$

Alice berechnet nun ihren zu $k_{A,priv}$ gehörenden öffentlichen Schlüssel:

$$k_{A,pub} = g^{e_A} \bmod n = 2^{17} \bmod 47 = 131072 \bmod 47 = 36$$

Alice besitzt somit das Schlüsselpaar:

$$\left(k_{A,priv}, k_{A,pub}\right) = (17,36)$$

Ebenso erzeugt sich Bob ein Schlüsselpaar. Dieses sei:

$$\left(k_{B,priv}, k_{B,pub}\right) = (13,14)$$

(Überprüfen Sie, dass dieses Zahlenpaar tatsächlich ein Schlüsselpaar ist!)

Nachdem Alice und Bob ihre öffentlichen Schlüssel ausgetauscht haben, können sie ihren gemeinsamen Schlüssel $k_{A,B}$ berechnen. Alice rechnet hierzu

$$\begin{aligned} k_{A,B} &= {k_{B,pub}}^{e_A} \bmod n = 14^{17} \bmod 47 \\ &= 30491346729331195904 \bmod 47 = 28 \end{aligned}$$

und Bob:

$$\begin{aligned} k_{A,B} &= {k_{A,pub}}^{e_B} \bmod n = 36^{13} \bmod 47 \\ &= 170581728179578208256 \bmod 47 = 28 \end{aligned}$$

10.4 Angriffe auf das DH-Verfahren

10.4.1 Pohlig-Hellman-Reduktion

Eine Brute-Force-Suche für das DL-Problem wird dadurch vereinfacht, dass sich die Brute-Force-Suche auf eine Brute-Force-Suche auf die Untergruppen von $\langle g \rangle$ mit maximaler Primzahlpotenz reduzieren lässt. Ein optimaler Schutz bietet die Wahl einer Primzahl n bei der $n-1$ die Primfaktorzerlegung $n-1 = 2 \cdot q$ hat (q ist eine Primzahl).

10.4.2 Shanks' „Baby Steps Giant Steps“-Verfahren (BSGS-Verfahren)

Falls die Möglichkeit besteht $\tilde{m} := \lceil\sqrt{m}\rceil$ viele Zahlen von $\mathbb{Z}n$ in einer Tabelle zu speichern, lässt sich ein gesuchter Exponent x mit dem BSGS-Verfahren bereits mit der Berechnung von höchsten $2\tilde{m}$ vielen Potenzen finden. Das Verfahren ähnelt dabei dem Meet-in-the-middle-Angriff (siehe 6.5).

10.4.3 Pollard's Rho-Methode

siehe Skript 3 Kapitel 2.2.3 auf Seite 26(42)

10.4.4 Zahlkörpersieb

Zu Beginn der 1990er Jahre wurde das Zahlkörpersieb (Number Field Sieve (NFS)) als Methode zur Faktorisierung großer Zahlen n und zur Bestimmung diskreter Logarithmen in $\mathbb{Z}_n^*$ entwickelt. Die Laufzeit zur Bestimmung diskreter Logarithmen in $\mathbb{Z}_n^*$ mithilfe des Zahlkörpersiebs kann durch

$$O\left(e^{\left(\left(\frac{64}{9}\right)^{1/3}+o(1)\right)(\ln(n))^{1/3}(\ln(\ln(n)))^{2/3}}\right)$$

von oben abgeschätzt werden.