Humboldt-Universität zu Berlin
Institut für Informatik

Seminar: Analyse kryptographischer Algorithmen
Leitung: Prof. Dr. Johannes Köbler, Matthias Schwan

CAST-128/256

Roland Stigge
eMail: stigge@informatik.hu-berlin.de
http://www.informatik.hu-berlin.de/~stigge/

8. Mai 2002

Inhaltsverzeichnis

1 Einleitung

Das die Vorlesung "Kryptologie 1" im Sommersemester 2002 begleitende Seminar "Analyse kryptographischer Algorithmen" analysiert die weniger bekannten, jedoch vielversprechenden Algorithmen der modernen Kryptologie.

Hierzu wählte ich den CAST-128- bzw. CAST-256-Algorithmus, der in der letztgenannten Form als Kandidat für den AES (Advanced Encryption Standard) vorgeschlagen wurde, jedoch nicht in die Endauswahl der 5 geeignetsten Verfahren kam.

Da CAST-256 auf CAST-128 aufbaut, werde ich, um entsprechende Redundanz auszuschließen, im folgenden immer zuerst bestimmte Eigenschaften des CAST-128 erläutern, um dann nur noch die Erweiterungen zum CAST-256 erklären zu müssen.

2 Geschichte

Als "Erfinder" des CAST gelten Carlisle Adams (Entrust Technologies, Inc., Kanada — "CA") und Stafford Tavares (Queens University Kingston, Kanada — "ST"). 1992 wurde der Name CAST erstmals benutzt und verschiedene Eigenschaften von CAST-1 und CAST-2 definiert. Nach verschiedenen Entwicklungsstufen wurde 1995 CAST-5 als CAST-128 veröffentlicht.

1997 wurde schließlich CAST-128 zu CAST-256 (= CAST-6) erweitert und in dieser erweiterten Form 1998 als AES-Kandidat vorgeschlagen.

Beide Algorithmen sind international für kommerziellen und nichtkommerziellen Gebrauch freigegeben und offengelegt.

CAST-128 wird z.B. ab PGP 5 eingesetzt. Bruce Schneier (ein Konkurrent!) schrieb 1998 in einer Newsgroup: "I give a big yuk to CAST-128."

3 Allgemeine Eigenschaften

3.1 CAST-128

Dieser Algorithmus ist dem DES und Blowfish ähnlich und kann als SPN (Substitution-Permutation-Network) und Produktalgorithmus (rundenbasierte Blockchiffre) klassifiziert werden. Er wird als resistent gegenüber differentieller, linearer und related-key Kryptoanalysis angesehen. Weitere Eigenschaften umfassen:

- Blockgröße (Klartext/Kryptotext): 64 bits
- Schlüsselgröße: 40 bis 128 bits (in 8-bit Schritten)
- Runden: 16 (12 bei kleinen Schlüsselgrößen)

- SAC (Strict Avalanche Criterion): Durch die Änderung eines Eingabebits ändert sich jedes einzelne Ausgabebit mit der Wahrscheinlichkeit von genau 1/2.
- BIC (Bit Independence Criterion): Keine Korrelation zwischen 2 Ausgabebitänderungen bei Änderung eines Eingabebits
- "No-Complementation"-Eigenschaft: Komplemente von Schlüssel und Klartext führen nicht zum Komplement des Kryptotextes. Da dies beim DES nicht gegeben ist, wird für eine Brute-Force Attacke die Anzahl der Iterationen reduziert.
- keine schwachen / semi-schwachen Schlüssel
- Algorithmus, Analyse und Konstruktion sind frei verfügbar
- Gleiche Struktur für Ver- und Entschlüsselung (bis auf (statisch) unterschiedliche Permutation der Rundenschlüssel)
- Performanz: 3,3 MBytes/sec (150MHz Pentium)

Die enthaltenen 8 Substitution-Boxes (S-Boxes) haben u.a. folgende Eigenschaften:

- Hohe Nicht-Linearität: 74 (Anzahl der Bits, die in der Wahrheitstabelle der Booleschen Funktion geändert werden müssen, um eine affine Boolesche Funktion zu erhalten. Affine Boolesche Funktion: $f(x_1, \ldots, x_n) = a_n x_n \oplus \ldots \oplus a_1 x_1 \oplus a_0$)
- Maximum bei Differenzen-Verteilung von 2

Ein besonderes Feature des Algorithmus ist die Konzentration auf den Operations-Mix, wobei mehrfach in verschiedener Reihenfolge XOR, Addition und Subtraktion miteinander verbunden werden. Dies erhöht die Sicherheit gegen lineare und differentielle Angriffe (Runden-Wahrscheinlichkeit verringert) und schließt differentielle Angriffe höherer Ordnung nahezu aus.

Mehrere verschiedene Rundenfunktionen erhöhen die Komplexität der Konstruktion linearer und differentieller Angriffe erheblich.

3.2 CAST-256

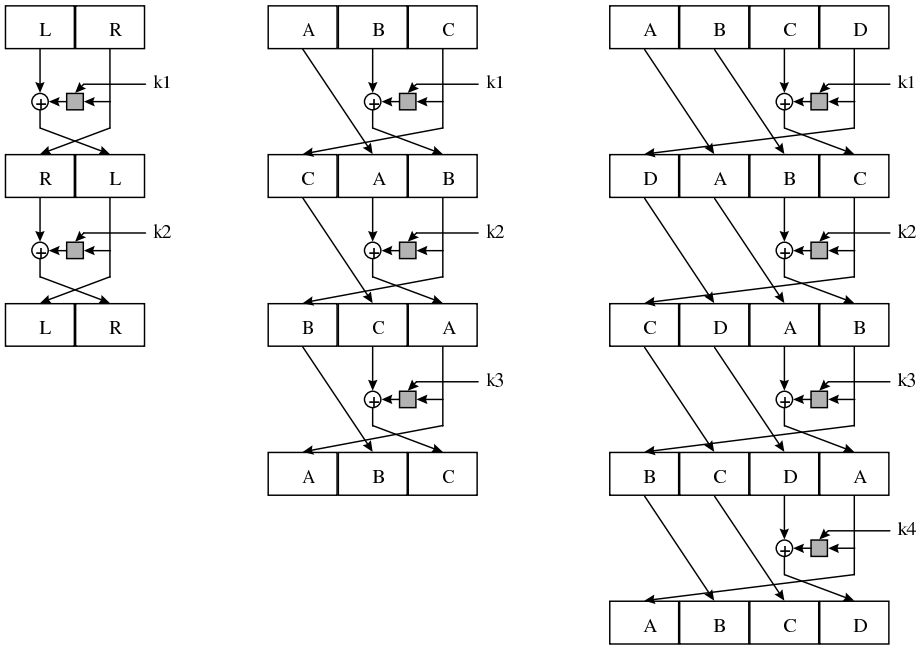
Dieser Algorithmus besitzt folgende Eigenschaften:

- Er basiert auf CAST-128: Rundenfunktionen, S-Boxen, Schlüsselmix, Operations-Mix
- Schlüsselgrößen: 128, 160, 192, 224, 256 bits
- Blockgröße: 128 bits, weitere sollten laut Autor möglich sein

- Für die Rundenschlüsselgenerierung wird eine ähnliche Verschlüsselungsfunktion wie für die Verschlüsselung ansich verwendet. Daraus folgt die Unabhängigkeit der Runden-Schlüssel

- 48 Runden insgesamt (12 Runden mit je 4 Teilrunden)

- Im Gegensatz zum DES wird ein erweitertes ("unvollständiges") Feistel-Netzwerk verwendet:



Während links ein gewöhnliches Feistel-Netzwerk (DES, CAST-128) dargestellt ist, nimmt nach rechts hin die Anzahl der Blöcke, die verschlüsselt werden soll, und damit auch die Anzahl der Runden, weil in jeder Runde effektiv nur je einer dieser Blöcke chiffriert wird, zu. Ganz rechts ist das Schema dargestellt, welches das Verfahren beim CAST-256 beschreibt.

4 Algorithmus

4.1 CAST-128

Zur Definition des Algorithmus ist die Definition folgender Funktion f: $\mathbb{Z}_{2^{32}} \rightarrow \mathbb{Z}_{2^{32}}$ notwendig:

$$
\begin{array}{lrcl}
\text{Typ 1:} & I & = & ((k_{m_i} + D) \circlearrowleft k_{r_i}) \\
 & f_1 & = & ((S_1[I_a] \oplus S_2[I_b]) - S_3[I_c]) + S_4[I_d] \\
\text{Typ 2:} & I & = & ((k_{m_i} \oplus D) \circlearrowleft k_{r_i}) \\
 & f_2 & = & ((S_1[I_a] - S_2[I_b]) + S_3[I_c]) \oplus S_4[I_d] \\
\text{Typ 3:} & I & = & ((k_{m_i} - D) \circlearrowleft k_{r_i}) \\
 & f_3 & = & ((S_1[I_a] + S_2[I_b]) \oplus S_3[I_c]) - S_4[I_d]
\end{array}
$$

Hierbei ist

- $\oplus$ die XOR-Verknüepfung,
- $+$ und $-$ die Addition bzw. Subtraktion mod 2^{32},
- $\circlearrowleft$ die Bitrotation nach links um die angegebene Stellenanzahl,
- $I_a, \ldots, I_d$ sind höchstwertigstes bis niederwertigstes Byte (in dieser Reihenfolge) von I,
- $S_1, \ldots, S_4$ sind S-Boxen,
- D ist die 32-Bit-Eingabe,
- k_{m_i} ist die 32-Bit-Maske, die als Rundenschlüssel (Runde i) verwendet wird, und
- k_{r_i} ist die 5-Bit-Rotationsangabe, die als Rundenschlüssel (Runde i) verwendet wird, um um die angegebene Anzahl an Stellen zu rotieren.

Die verschiedenen "Typen" kommen in verschiedenen Runden zur Anwendung: Typ 1 in den Runden 1, 4, 7, 10, 13 und 16, Typ 2 in den Runden 2, 5, 8, 11, 14 und Typ 3 in den Runden 3, 6, 9, 12 und 15.

Der Algorithmus selbst läuft folgendermaßen ab:

1. Eingabe: Klartext $m_1 \cdots m_{64}$, Schlüssel $k = k_1 \cdots k_{128}$
2. Rundenschlüsselgenerierung: k_{m_i} (Maske, 32 bits), k_{r_i} (Rotation, 5 bits) für jede Runde i (abh. von k)
3. $(L_0, R_0) \leftarrow (m_1 \cdots m_{64})$
   d.h. $L_0 \leftarrow (m_1 \cdots m_{32})$ und $R_0 \leftarrow (m_{33} \cdots m_{64})$
4. 16 Runden: $L_i = R_{i-1}$ und $R_i = L_{i-1} \oplus f(R_{i-1}, k_{m_i}, k_{r_i})$ für alle $i \in \{1, \ldots, 16\}$ (Typ von f hängt von i ab)
5. Ausgabe: Kryptotext $c_1 \cdots c_{64} \leftarrow (R_{16}, L_{16})$
   d.h. noch eine Vertauschung

Zur Entschlüsselung wird der gleiche Algorithmus verwendet, wobei der einzige Unterschied in der umgekehrten Reihenfolge der generierten Schlüssel besteht. (Es erfolgt eine Berechnung von (L_0, R_0) aus (R_{16}, L_{16}).)

Die 8 S-Boxen sind unterteilt in 4 S-Boxen $(S_1, \ldots, S_4)$ für die Rundenfunktionen und 4 S-Boxen $(S_5, \ldots, S_8)$ für die Schlüsselgenerierung. Letztere werden nur einmalig zur Initialisierung einer Ver-/Entschlüsselung benötigt. Organisiert sind die S-Boxen als Lookup-Tables von 2^8 x 32bits ("8x32"). Als Beispiel sei hier nur die S-Box S_1 aufgeführt:

```
30fb40d4 9fa0ff0b 6beccd2f 3f258c7a 1e213f2f 9c004dd3 6003e540 cf9fc949
bfd4af27 88bbbdb5 e2034090 98d09675 6e63a0e0 15c361d2 c2e7661d 22d4ff8e
28683b6f c07fd059 ff2379c8 775f50e2 43c340d3 df2f8656 887ca41a a2d2bd2d
a1c9e0d6 346c4819 61b76d87 22540f2f 2abe32e1 aa54166b 22568e3a a2d341d0
66db40c8 a784392f 004dff2f 2db9d2de 97943fac 4a97c1d8 527644b7 b5f437a7
b82cbaef d751d159 6ff7f0ed 5a097a1f 827b68d0 90ecf52e 22b0c054 bc8e5935
4b6d2f7f 50bb64a2 d2664910 bee5812d b7332290 e93b159f b48ee411 4bff345d
fd45c240 ad31973f c4f6d02e 55fc8165 d5b1caad a1ac2dae a2d4b76d c19b0c50
882240f2 0c6e4f38 a4e4bfd7 4f5ba272 564c1d2f c59c5319 b949e354 b04669fe
b1b6ab8a c71358dd 6385c545 110f935d 57538ad5 6a390493 e63d37e0 2a54f6b3
3a787d5f 6276a0b5 19a6fcdf 7a42206a 29f9d4d5 f61b1891 bb72275e aa508167
38901091 c6b505eb 84c7cb8c 2ad75a0f 874a1427 a2d1936b 2ad286af aa56d291
d7894360 425c750d 93b39e26 187184c9 6c00b32d 73e2bb14 a0bebc3c 54623779
64459eab 3f328b82 7718cf82 59a2cea6 04ee002e 89fe78e6 3fab0950 325ff6c2
81383f05 6963c5c8 76cb5ad6 d49974c9 ca180dcf 380782d5 c7fa5cf6 8ac31511
35e79e13 47da91d0 f40f9086 a7e2419e 31366241 051ef495 aa573b04 4a805d8d
548300d0 00322a3c bf64cddf ba57a68e 75c6372b 50afd341 a7c13275 915a0bf5
6b54bfab 2b0b1426 ab4cc9d7 449ccd82 f7fbf265 ab85c5f3 1b55db94 aad4e324
cfa4bd3f 2deaa3e2 9e204d02 c8bd25ac eadf55b3 d5bd9e98 e31231b2 2ad5ad6c
954329de adbe4528 d8710f69 aa51c90f aa786bf6 22513f1e aa51a79b 2ad344cc
7b5a41f0 d37cfbad 1b069505 41ece491 b4c332e6 032268d4 c9600acc ce387e6d
bf6bb16c 6a70fb78 0d03d9c9 d4df39de e01063da 4736f464 5ad328d8 b347cc96
75bb0fc3 98511bfb 4ffbcc35 b58bcf6a e11f0abc bfc5fe4a a70aec10 ac39570a
3f04442f 6188b153 e0397a2e 5727cb79 9ceb418f 1cacd68d 2ad37c96 0175cb9d
c69dff09 c75b65f0 d9db40d8 ec0e7779 4744ead4 b11c3274 dd24cb9e 7e1c54bd
f01144f9 d2240eb1 9675b3fd a3ac3755 d47c27af 51c85f4d 56907596 a5bb15e6
580304f0 ca042cf1 011a37ea 8dbfaadb 35ba3e4a 3526ffa0 c37b4d09 bc306ed9
98a52666 5648f725 ff5e569d 0ced63d0 7c63b2cf 700b45e1 d5ea50f1 85a92872
af1fbda7 d4234870 a7870bf3 2d3b4d79 42e04198 0cd0ede7 26470db8 f881814c
474d6ad7 7c0c5e5c d1231959 381b7298 f5d2f4db ab838653 6e2f1e23 83719c9e
bd91e046 9a56456e dc39200c 20c8c571 962bda1c e1e696ff b141ab08 7cca89b9
1a69e783 02cc4843 a2f7c579 429ef47d 427b169c 5ac9f049 dd8f0f00 5c8165bf
```

Nun bleibt nur noch die Schlüsselgenerierung zu klären. Dazu betrachten wir eine *Kopie* des 128-Bit-Schlüssels als Folge von 16 Bytes: $x_0, \ldots, x_{15}$ mit

x_0 als höchstwertigstem Byte. Zusätzlich (und analog zu $x_0, \ldots, x_{15}$) wird $z_0, \ldots, z_{15}$ als temporärer 128-Bit-Wert definiert. Jetzt können k_{m_i} und k_{r_i} wie folgt definiert werden:

$$\begin{aligned}
z_0z_1z_2z_3 &= x_0x_1x_2x_3 \oplus S_5[x_{13}] \oplus S_6[x_{15}] \oplus S_7[x_{12}] \oplus S_8[x_{14}] \oplus S_7[x_8] \\
z_4z_5z_6z_7 &= x_8x_9x_{10}x_{11} \oplus S_5[z_0] \oplus S_6[z_2] \oplus S_7[z_1] \oplus S_8[z_3] \oplus S_8[x_{10}] \\
z_8z_9z_{10}z_{11} &= x_{12}x_{13}x_{14}x_{15} \oplus S_5[z_7] \oplus S_6[z_6] \oplus S_7[z_5] \oplus S_8[z_4] \oplus S_5[x_9] \\
z_{12}z_{13}z_{14}z_{15} &= x_4x_5x_6x_7 \oplus S_5[z_{10}] \oplus S_6[z_9] \oplus S_7[z_{11}] \oplus S_8[z_8] \oplus S_6[x_{11}]
\end{aligned}$$

$$\begin{aligned}
k_{m_1} &= S_5[z_8] \oplus S_6[z_9] \oplus S_7[z_7] \oplus S_8[z_6] \oplus S_5[z_2] \\
k_{m_2} &= S_5[z_{10}] \oplus S_6[z_{11}] \oplus S_7[z_5] \oplus S_8[z_4] \oplus S_6[z_6] \\
k_{m_3} &= S_5[z_{12}] \oplus S_6[z_{13}] \oplus S_7[z_3] \oplus S_8[z_2] \oplus S_7[z_9] \\
k_{m_4} &= S_5[z_{14}] \oplus S_6[z_{15}] \oplus S_7[z_1] \oplus S_8[z_0] \oplus S_8[z_{12}]
\end{aligned}$$

$$\begin{aligned}
x_0x_1x_2x_3 &= z_8z_9z_{10}z_{11} \oplus S_5[z_5] \oplus S_6[z_7] \oplus S_7[z_4] \oplus S_8[z_6] \oplus S_7[z_0] \\
x_4x_5x_6x_7 &= z_0z_1z_2z_3 \oplus S_5[x_0] \oplus S_6[x_2] \oplus S_7[x_1] \oplus S_8[x_3] \oplus S_8[z_2] \\
x_8x_9x_{10}x_{11} &= z_4z_5z_6z_7 \oplus S_5[x_7] \oplus S_6[x_6] \oplus S_7[x_5] \oplus S_8[x_4] \oplus S_5[z_1] \\
x_{12}x_{13}x_{14}x_{15} &= z_{12}z_{13}z_{14}z_{15} \oplus S_5[x_{10}] \oplus S_6[x_9] \oplus S_7[x_{11}] \oplus S_8[x_8] \oplus S_6[z_3]
\end{aligned}$$

$$\begin{aligned}
k_{m_5} &= S_5[x_3] \oplus S_6[x_2] \oplus S_7[x_{12}] \oplus S_8[x_{13}] \oplus S_5[x_8] \\
k_{m_6} &= S_5[x_1] \oplus S_6[x_0] \oplus S_7[x_{14}] \oplus S_8[x_{15}] \oplus S_6[x_{13}] \\
k_{m_7} &= S_5[x_7] \oplus S_6[x_6] \oplus S_7[x_8] \oplus S_8[x_9] \oplus S_7[x_3] \\
k_{m_8} &= S_5[x_5] \oplus S_6[x_4] \oplus S_7[x_{10}] \oplus S_8[x_{11}] \oplus S_8[x_7]
\end{aligned}$$

$$\begin{aligned}
z_0z_1z_2z_3 &= x_0x_1x_2x_3 \oplus S_5[x_{13}] \oplus S_6[x_{15}] \oplus S_7[x_{12}] \oplus S_8[x_{14}] \oplus S_7[x_8] \\
z_4z_5z_6z_7 &= x_8x_9x_{10}x_{11} \oplus S_5[z_0] \oplus S_6[z_2] \oplus S_7[z_1] \oplus S_8[z_3] \oplus S_8[x_{10}] \\
z_8z_9z_{10}z_{11} &= x_{12}x_{13}x_{14}x_{15} \oplus S_5[z_7] \oplus S_6[z_6] \oplus S_7[z_5] \oplus S_8[z_4] \oplus S_5[x_9] \\
z_{12}z_{13}z_{14}z_{15} &= x_4x_5x_6x_7 \oplus S_5[z_{10}] \oplus S_6[z_9] \oplus S_7[z_{11}] \oplus S_8[z_8] \oplus S_6[x_{11}]
\end{aligned}$$

$$\begin{aligned}
k_{m_9} &= S_5[z_3] \oplus S_6[z_2] \oplus S_7[z_{12}] \oplus S_8[z_{13}] \oplus S_5[z_9] \\
k_{m_{10}} &= S_5[z_1] \oplus S_6[z_0] \oplus S_7[z_{14}] \oplus S_8[z_{15}] \oplus S_6[z_{12}] \\
k_{m_{11}} &= S_5[z_7] \oplus S_6[z_6] \oplus S_7[z_8] \oplus S_8[z_9] \oplus S_7[z_2] \\
k_{m_{12}} &= S_5[z_5] \oplus S_6[z_4] \oplus S_7[z_{10}] \oplus S_8[z_{11}] \oplus S_8[z_6]
\end{aligned}$$

$$
\begin{aligned}
x_0x_1x_2x_3 &= z_8z_9z_{10}z_{11} \oplus S_5[z_5] \oplus S_6[z_7] \oplus S_7[z_4] \oplus S_8[z_6] \oplus S_7[z_0] \\
x_4x_5x_6x_7 &= z_0z_1z_2z_3 \oplus S_5[x_0] \oplus S_6[x_2] \oplus S_7[x_1] \oplus S_8[x_3] \oplus S_8[z_2] \\
x_8x_9x_{10}x_{11} &= z_4z_5z_6z_7 \oplus S_5[x_7] \oplus S_6[x_6] \oplus S_7[x_5] \oplus S_8[x_4] \oplus S_5[z_1] \\
x_{12}x_{13}x_{14}x_{15} &= z_{12}z_{13}z_{14}z_{15} \oplus S_5[x_{10}] \oplus S_6[x_9] \oplus S_7[x_{11}] \oplus S_8[x_8] \oplus S_6[z_3]
\end{aligned}
$$

$$
\begin{aligned}
k_{m_{13}} &= S_5[x_8] \oplus S_6[x_9] \oplus S_7[x_7] \oplus S_8[x_6] \oplus S_5[x_3] \\
k_{m_{14}} &= S_5[x_{10}] \oplus S_6[x_{11}] \oplus S_7[x_5] \oplus S_8[x_4] \oplus S_6[x_7] \\
k_{m_{15}} &= S_5[x_{12}] \oplus S_6[x_{13}] \oplus S_7[x_3] \oplus S_8[x_2] \oplus S_7[x_8] \\
k_{m_{16}} &= S_5[x_{14}] \oplus S_6[x_{15}] \oplus S_7[x_1] \oplus S_8[x_0] \oplus S_8[x_{13}]
\end{aligned}
$$

$$
\begin{aligned}
z_0z_1z_2z_3 &= x_0x_1x_2x_3 \oplus S_5[x_{13}] \oplus S_6[x_{15}] \oplus S_7[x_{12}] \oplus S_8[x_{14}] \oplus S_7[x_8] \\
z_4z_5z_6z_7 &= x_8x_9x_{10}x_{11} \oplus S_5[z_0] \oplus S_6[z_2] \oplus S_7[z_1] \oplus S_8[z_3] \oplus S_8[x_{10}] \\
z_8z_9z_{10}z_{11} &= x_{12}x_{13}x_{14}x_{15} \oplus S_5[z_7] \oplus S_6[z_6] \oplus S_7[z_5] \oplus S_8[z_4] \oplus S_5[x_9] \\
z_{12}z_{13}z_{14}z_{15} &= x_4x_5x_6x_7 \oplus S_5[z_{10}] \oplus S_6[z_9] \oplus S_7[z_{11}] \oplus S_8[z_8] \oplus S_6[x_{11}]
\end{aligned}
$$

$$
\begin{aligned}
k_{r_1} &= S_5[z_8] \oplus S_6[z_9] \oplus S_7[z_7] \oplus S_8[z_6] \oplus S_5[z_2] \\
k_{r_2} &= S_5[z_{10}] \oplus S_6[z_{11}] \oplus S_7[z_5] \oplus S_8[z_4] \oplus S_6[z_6] \\
k_{r_3} &= S_5[z_{12}] \oplus S_6[z_{13}] \oplus S_7[z_3] \oplus S_8[z_2] \oplus S_7[z_9] \\
k_{r_4} &= S_5[z_{14}] \oplus S_6[z_{15}] \oplus S_7[z_1] \oplus S_8[z_0] \oplus S_8[z_{12}]
\end{aligned}
$$

$$
\begin{aligned}
x_0x_1x_2x_3 &= z_8z_9z_{10}z_{11} \oplus S_5[z_5] \oplus S_6[z_7] \oplus S_7[z_4] \oplus S_8[z_6] \oplus S_7[z_0] \\
x_4x_5x_6x_7 &= z_0z_1z_2z_3 \oplus S_5[x_0] \oplus S_6[x_2] \oplus S_7[x_1] \oplus S_8[x_3] \oplus S_8[z_2] \\
x_8x_9x_{10}x_{11} &= z_4z_5z_6z_7 \oplus S_5[x_7] \oplus S_6[x_6] \oplus S_7[x_5] \oplus S_8[x_4] \oplus S_5[z_1] \\
x_{12}x_{13}x_{14}x_{15} &= z_{12}z_{13}z_{14}z_{15} \oplus S_5[x_{10}] \oplus S_6[x_9] \oplus S_7[x_{11}] \oplus S_8[x_8] \oplus S_6[z_3]
\end{aligned}
$$

$$
\begin{aligned}
k_{r_5} &= S_5[x_3] \oplus S_6[x_2] \oplus S_7[x_{12}] \oplus S_8[x_{13}] \oplus S_5[x_8] \\
k_{r_6} &= S_5[x_1] \oplus S_6[x_0] \oplus S_7[x_{14}] \oplus S_8[x_{15}] \oplus S_6[x_{13}] \\
k_{r_7} &= S_5[x_7] \oplus S_6[x_6] \oplus S_7[x_8] \oplus S_8[x_9] \oplus S_7[x_3] \\
k_{r_8} &= S_5[x_5] \oplus S_6[x_4] \oplus S_7[x_{10}] \oplus S_8[x_{11}] \oplus S_8[x_7]
\end{aligned}
$$

$$
\begin{aligned}
z_0z_1z_2z_3 &= x_0x_1x_2x_3 \oplus S_5[x_{13}] \oplus S_6[x_{15}] \oplus S_7[x_{12}] \oplus S_8[x_{14}] \oplus S_7[x_8] \\
z_4z_5z_6z_7 &= x_8x_9x_{10}x_{11} \oplus S_5[z_0] \oplus S_6[z_2] \oplus S_7[z_1] \oplus S_8[z_3] \oplus S_8[x_{10}] \\
z_8z_9z_{10}z_{11} &= x_{12}x_{13}x_{14}x_{15} \oplus S_5[z_7] \oplus S_6[z_6] \oplus S_7[z_5] \oplus S_8[z_4] \oplus S_5[x_9] \\
z_{12}z_{13}z_{14}z_{15} &= x_4x_5x_6x_7 \oplus S_5[z_{10}] \oplus S_6[z_9] \oplus S_7[z_{11}] \oplus S_8[z_8] \oplus S_6[x_{11}]
\end{aligned}
$$

$$
\begin{aligned}
k_{r_9} &= S_5[z_3] \oplus S_6[z_2] \oplus S_7[z_{12}] \oplus S_8[z_{13}] \oplus S_5[z_9] \\
k_{r_{10}} &= S_5[z_1] \oplus S_6[z_0] \oplus S_7[z_{14}] \oplus S_8[z_{15}] \oplus S_6[z_{12}] \\
k_{r_{11}} &= S_5[z_7] \oplus S_6[z_6] \oplus S_7[z_8] \oplus S_8[z_9] \oplus S_7[z_2] \\
k_{r_{12}} &= S_5[z_5] \oplus S_6[z_4] \oplus S_7[z_{10}] \oplus S_8[z_{11}] \oplus S_8[z_6]
\end{aligned}
$$

$$
\begin{aligned}
x_0x_1x_2x_3 &= z_8z_9z_{10}z_{11} \oplus S_5[z_5] \oplus S_6[z_7] \oplus S_7[z_4] \oplus S_8[z_6] \oplus S_7[z_0] \\
x_4x_5x_6x_7 &= z_0z_1z_2z_3 \oplus S_5[x_0] \oplus S_6[x_2] \oplus S_7[x_1] \oplus S_8[x_3] \oplus S_8[z_2] \\
x_8x_9x_{10}x_{11} &= z_4z_5z_6z_7 \oplus S_5[x_7] \oplus S_6[x_6] \oplus S_7[x_5] \oplus S_8[x_4] \oplus S_5[z_1] \\
x_{12}x_{13}x_{14}x_{15} &= z_{12}z_{13}z_{14}z_{15} \oplus S_5[x_{10}] \oplus S_6[x_9] \oplus S_7[x_{11}] \oplus S_8[x_8] \oplus S_6[z_3]
\end{aligned}
$$

$$
\begin{aligned}
k_{r_{13}} &= S_5[x_8] \oplus S_6[x_9] \oplus S_7[x_7] \oplus S_8[x_6] \oplus S_5[x_3] \\
k_{r_{14}} &= S_5[x_{10}] \oplus S_6[x_{11}] \oplus S_7[x_5] \oplus S_8[x_4] \oplus S_6[x_7] \\
k_{r_{15}} &= S_5[x_{12}] \oplus S_6[x_{13}] \oplus S_7[x_3] \oplus S_8[x_2] \oplus S_7[x_8] \\
k_{r_{16}} &= S_5[x_{14}] \oplus S_6[x_{15}] \oplus S_7[x_1] \oplus S_8[x_0] \oplus S_8[x_{13}]
\end{aligned}
$$

Von kr_i werden jeweils nur die 5 niederwertigsten Bits benutzt (der Rest sei daher ausmaskiert).

Da der Algorithmus auf 128-Bit Schlüssel ausgelegt ist, muß der angegebene Schlüssel bei kleineren Schlüsselgrößen auf 128 bit rechts (niederwertigste Bits) mit Nullen aufgefüllt werden. Desweiteren muß bei Schlüsselgrößen bis 80 bits die Anzahl der Runden von 16 auf 12 verringert werden.

4.2 CAST-256

Wie bereits angedeutet, werden hierfür folgende Elemente des CAST-128 weiter verwendet:

- Prinzip der Rundenschlüssel (k_{m_i}, k_{r_i}), deren Berechnung ist jedoch neu
- Die Rundenfunktionen $f_1, \ldots, f_3$

- Die S-Boxen $S_1, \ldots, S_4$

Im Speziellen wird der Algorithmus wie folgt beschrieben: Sei $\beta = (ABCD)$ ein 128-Bit-Block mit $A, \ldots, D$ jew. 32-Bit und sei

$$\beta \leftarrow Q_i(\beta)$$

die Kurznotation für

$$\begin{aligned} C &= C \oplus f_1(D, k_{r_0}^{(i)}, k_{m_0}^{(i)}) \\ B &= B \oplus f_2(C, k_{r_1}^{(i)}, k_{m_1}^{(i)}) \\ A &= A \oplus f_3(B, k_{r_2}^{(i)}, k_{m_2}^{(i)}) \\ D &= D \oplus f_1(A, k_{r_3}^{(i)}, k_{m_3}^{(i)}) \end{aligned}$$

und

$$\beta \leftarrow \overline{Q_i}(\beta)$$

die Kurznotation für

$$\begin{aligned} D &= D \oplus f_1(A, k_{r_3}^{(i)}, k_{m_3}^{(i)}) \\ A &= A \oplus f_3(B, k_{r_2}^{(i)}, k_{m_2}^{(i)}) \\ B &= B \oplus f_2(C, k_{r_1}^{(i)}, k_{m_1}^{(i)}) \\ C &= C \oplus f_1(D, k_{r_0}^{(i)}, k_{m_0}^{(i)}) \end{aligned}$$

wobei Q als "Vorwärts-Vierer-Runde" und $\overline{Q}$ als "Rückwärts-Vierer-Runde" bezeichnet wird.

Sei weiterhin $k_r^{(i)} = \{k_{r_0}^{(i)}, k_{r_1}^{(i)}, k_{r_2}^{(i)}, k_{r_3}^{(i)}\}$ die Menge der 5-Bit-Rotations-Schlüssel für die i-te Vierer-Runde. Analog sei $k_m^{(i)} = \{k_{m_0}^{(i)}, k_{m_1}^{(i)}, k_{m_2}^{(i)}, k_{m_3}^{(i)}\}$ die Menge der 32-Bit-Masken-Schlüssel für die i-te Vierer-Runde.

Sei $\kappa = (ABCDEFGH)$ ein 256-Bit-Block mit $A, \ldots, H$ jew. 32-Bit und sei

$$\kappa \leftarrow \omega_i(\kappa)$$

die Kurzform für

$$
\begin{aligned}
G &= G \oplus f_1(H, t_{r_0}^{(i)}, t_{m_0}^{(i)}) \\
F &= F \oplus f_2(G, t_{r_1}^{(i)}, t_{m_1}^{(i)}) \\
E &= E \oplus f_3(F, t_{r_2}^{(i)}, t_{m_2}^{(i)}) \\
D &= D \oplus f_1(E, t_{r_3}^{(i)}, t_{m_3}^{(i)}) \\
C &= C \oplus f_2(D, t_{r_4}^{(i)}, t_{m_4}^{(i)}) \\
B &= B \oplus f_3(C, t_{r_5}^{(i)}, t_{m_5}^{(i)}) \\
A &= A \oplus f_1(B, t_{r_6}^{(i)}, t_{m_6}^{(i)}) \\
H &= H \oplus f_2(A, t_{r_7}^{(i)}, t_{m_7}^{(i)})
\end{aligned}
$$

wobei ω auch als "Vorwärts-Oktave" bezeichnet wird.
Sei desweiteren

$$k_r^{(i)} \leftarrow \kappa$$

die Kurznotation für

$$
\begin{aligned}
k_{r_0}^{(i)} &= LSB(5, A) \\
k_{r_1}^{(i)} &= LSB(5, C) \\
k_{r_2}^{(i)} &= LSB(5, E) \\
k_{r_3}^{(i)} &= LSB(5, G)
\end{aligned}
$$

wobei $LSB(n, x)$ die n niederwertigsten Bits von x darstellt. Analog sei

$$k_m^{(i)} \leftarrow \kappa$$

die Kurznotation für

$$
\begin{aligned}
k_{m_0}^{(i)} &= H \\
k_{m_1}^{(i)} &= F \\
k_{m_2}^{(i)} &= D \\
k_{m_3}^{(i)} &= B
\end{aligned}
$$

Nun kann die Verschüsselung selbst definiert werden:

1: Eingabe: β = 128-Bit-Klartextblock

2: **for** $i = 0$ to 5 **do**
3: $\beta \leftarrow Q_i(\beta)$
4: **end for**
5: **for** $i = 6$ to 11 **do**
6: $\beta \leftarrow \overline{Q_i}(\beta)$
7: **end for**
8: Ausgabe: 128-Bit-Kryptotextblock = β

Zur Entschlüsselung wird der gleiche Algorithmus verwendet werden, nur muß hierbei (vorher) die Reihenfolge der Vierer-Rundenschlüssel umgekehrt werden:

1: **for** $i = 0$ to 11 **do**
2: $k_{r_{new}}^{(i)} = k_r^{(11-i)}$
3: $k_{m_{new}}^{(i)} = k_m^{(11-i)}$
4: **end for**

Wie schon angedeutet, wird für den CAST-256 ein neuer Algorithmus für die Berechnung der Rundenschlüssel angegeben:

Initialisierung:

1: $c_m = 2^{30}\sqrt{2} = 5A827999_{16}$
2: $m_m = 2^{30}\sqrt{3} = 6ED9EBA1_{16}$
3: $c_r = 19$
4: $m_r = 17$
5: **for** $i = 0$ to 23 **do**
6: **for** $j = 0$ to 7 **do**
7: $t_{m_j}^{(i)} = c_m$
8: $c_m = (c_m + m_m) \mod 2^{32}$
9: $t_{r_j}^{(i)} = c_r$
10: $c_r = (c_r + m_r) \mod 32$
11: **end for**
12: **end for**

Rundenschlüsselberechnung:

1: $\kappa = ABCDEFGH$ = 256-Bit-Primär-Schüssel
2: **for** $i = 0$ to 11 **do**
3: $\kappa \leftarrow \omega_{2i}(\kappa)$
4: $\kappa \leftarrow \omega_{2i+1}(\kappa)$
5: $k_r^{(i)} \leftarrow \kappa$
6: $k_m^{(i)} \leftarrow \kappa$
7: **end for**

Falls der "Primärschlüssel" kleiner als 256 Bit sein sollte, wird dieser bis 256 Bits links (niederwertig) mit Nullen aufgefüllt.

5 Analyse, Angriffe

5.1 Allgemein

Zusätzlich zu den o.g. Eigenschaften der Algorithmen kann rückblickend auf die 10 Jahre Entwicklungsarbeit "leider" nur gesagt werden, daß es offenbar keine ernst zu nehmenden veröffentlichten Angriffe gibt, die den Algorithmus in der Praxis aushebeln könnten.

Vereinfachte Versionen der Algorithmen haben zu Analysezwecken jedoch geholfen, die Sicherheit besser zu bewerten, z.B. ist bei jew. bis zu 6 Runden unter vereinfachten Bedingungen (keine Rotation, vereinfachter Operationen-Mix, ...) von "Non-Surjective Attacks" und "HOD Attacks" berichtet worden. Dies liegt an der Natur des Algorithmus: Erst nach 7 Runden ist jedes der 128 Ausgabebits von jedem Eingabebit abhängig [6].

Desweiteren werde ich im folgenden der Vollständigkeit halber einige mögliche (hier bislang erfolglose) Angriffe aufzählen:

Nur-Kryptotext-Angriff (Ciphertext Only Attack): Nach $2^{n/2} = 2^{64}$ Blöcken (mit n = Blockgröße) im CBC-Modus liegt die Wahrscheinlilchkeit bereits bei 1/2, daß es einen XOR-Zusammenhang zwischen zwei Klartexten gibt.

Bekannter-Klartext-Angriff (Known Plaintext Attack): Lineare Kryptoanalysis — verwendet Häufungen im Auftreten linearer Zusammenhänge zwischen Eingabe, Schlüssel und Ausgabe der Rundenfunktion einer Produktchiffre

Gewählter-Klartext-Angriff (Chosen Plaintext Attack): Differentielle Kryptoanalysis — verwendet Häufungen bei der Differenzenverteilung in der Ausgabe bei bestimmten Differenzen in der Eingabe der Rundenfunktion einer Produktchiffre

Gewählter-Schlüssel-Angriff (Chosen Key Attack): Verwendet Zusammenhänge zwischen "Primary Key" und Rundenschlüsseln

Ähnlicher-Schlüssel-Angriff (Related Key Attack): Nutzt Zusammenhang zwischen aufeinanderfolgenden Rundenschlüsseln aus

Angriff höheren Grades (Higher Order (HOD) Attack): Verallgemeinerung differentieller Angriffe

Nicht-Surjektivitäts-Angriff (Non Surjective Attack): Statistischer Angriff, der nicht-surjektive Rundenfunktionen ausnutzt

Kombinierter Angriff (Combination Attack): Kombination aus o.g. Verfahren

Da auch keine schwachen Schlüssel oder semi-schwachen (als Paar verwendeten) Schlüssel bekannt sind, können alle bekannten Schlüssel verwendet werden.

5.2 CAST-128

Speziell bei diesem Algorithmus läßt sich die gute Speicher- und Zeitkomplexität hervorheben. Da alle Schlüsselgrößen auf 128 Bit erweitert werden, ergeben sich hier für alle Schlüsselgrößen gleiche Werte.

5.3 CAST-256

Durch das Erbe vom CAST-128 erbt dieser Algorithmus die Sicherheit seines Vorgängers, spart sogar Speicherplatz (nur halb so viele S-Boxen) und erreicht eine Geschwindigkeit, die der Autor mit 2/3 von CAST-128 angibt. Damit ist er leider nicht einmal halb so schnell wie Rijndael.

Durch die allgemein gute Verständlichkeit des verallgemeinerten Feistel-Netzwerkes gibt es einen Konsenz bei der Transparenz dieses (natürlich offengelegten) Algorithmus.

Die Anzahl der Runden wurde vom DES (8 Doppel-Runden) abgeleitet. Folglich müßte die gesuchte Anzahl also mindestens 8 Vierer-Runden betragen.

Literatur

[1] Carlisle Adams: “The CAST-256 Encryption Algorithm” (Teil der AES-Bewerbung), Juni 1998
`http://www.entrust.com/resources/pdf/cast-256.pdf`

[2] C. Adams, Entrust Technologies: RFC2144 - “The CAST-128 Encryption Algorithm”, Mai 1997
`http://www.ietf.org/rfc/rfc2144.txt`

[3] C. Adams, J. Gilchrist: RFC2612 - “The CAST-256 Encryption Algorithm”, Juni 1999
`http://www.ietf.org/rfc/rfc2612.txt`

[4] C. Adams: Constructing Symmetric Ciphers using the CAST Design Procedure
`http://www.entrust.com/resources/pdf/cast.pdf`

[5] C. Adams: “CAST-256 - A Submission for the Advanced Encryption Standard” (Folien), August 1998
`http://csrc.nist.gov/encryption/aes/round1/conf1/cast-slides.pdf`

[6] Adams, Heys, Tavares, Wiener: "An Analysis of the CAST-256 Cipher"
http://www.engr.mun.ca/~howard/PAPERS/cast256.ps