IT-Sicherheit

- Sicherheit vernetzter Systeme -

Kapitel 7: Kryptographische Hash-Funktionen

Rootkit in der Netzwerkkarte

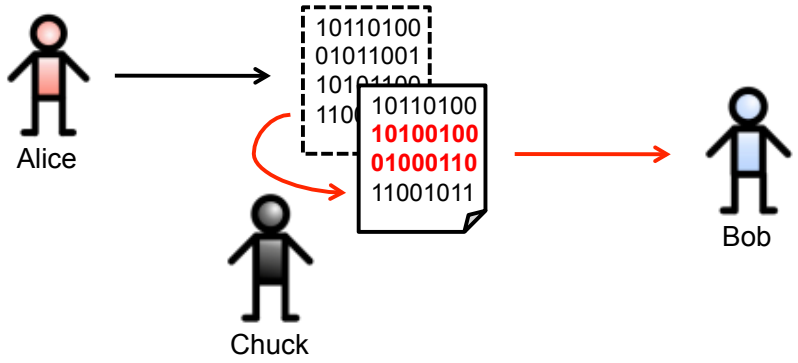
- Rootkit als Firmware für Broadcom NetXtreme Netzwerkkarten
- Speicherzugriff (direct memory access DMA) über PCI-Schnittstelle möglich
- Kann von Virenscannern nicht gefunden werden
- Bislang nur für (Server-)Netzwerkkarten mit größerem eigenen Speicher geeignet
- Angriffsszenario seit 2006
  - Aber erste Implementierung eines vollständigen Rootkits (kein Nachladen von Code)
  - Ähnlich: Verstecken von Rootkits im PC-BIOS oder Grafikkartenspeicher
- Weitere Infos:
  http://esec-lab.sogeti.com/dotclear/index.php?post%2F2010%2F11%2F21%2FPresentation-at-Hack.lu-%3A-Reversing-the-Broacom-NetExtreme-s-firmware

Inhalt

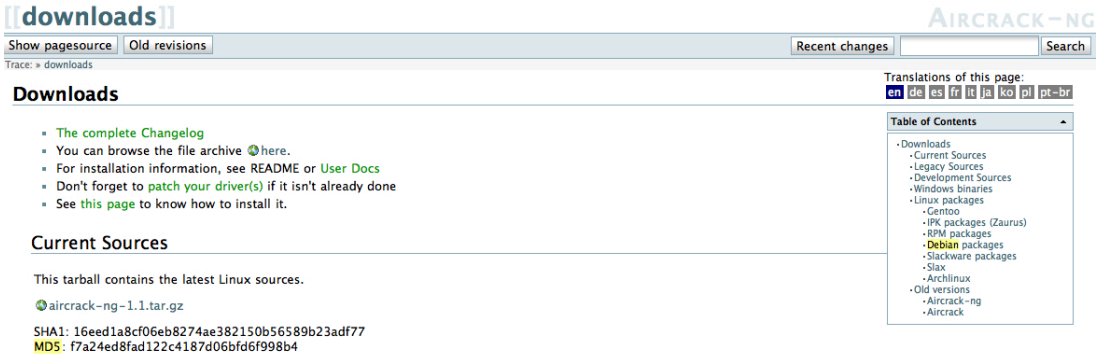
- Def.: Kryptographische Hash-Verfahren
- Angriffe gegen One-Way-Hash-Funktionen
- Konstruktion von Hash-Funktionen
- Algorithmen:
  - MD4
  - MD5
  - Whirlpool

Hash-Funktionen zur Integritätssicherung

- Ziel: Sicherstellen, dass Manipulationen an einer übertragenen Nachricht erkannt werden.

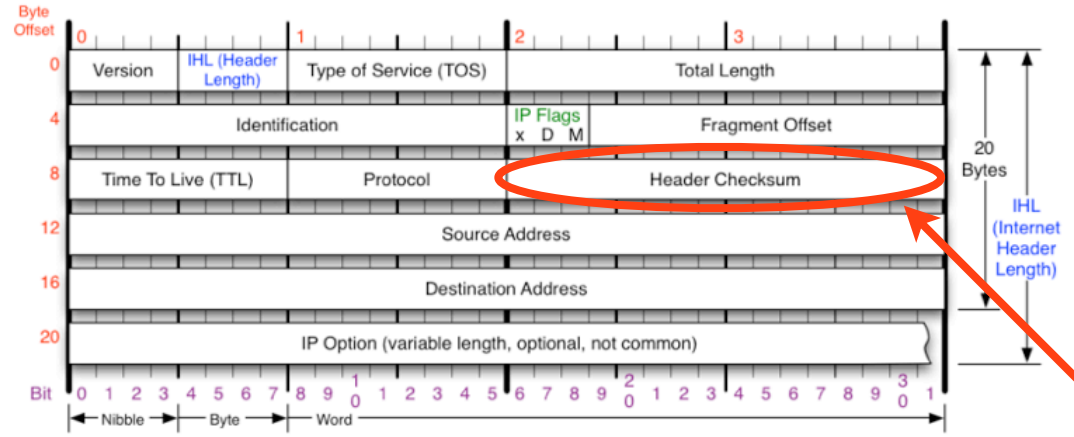


- Beispiel Software-Distribution:



Herkömmliche vs. kryptographische Hash-Funktionen

- Prüfsummen dienen der Erkennung von (unbeabsichtigten) Übertragungsfehlern, z.B. beim IPv4-Header:



Quelle: nmap.org

16-bit Addition / Einerkomplement

Version

Version of IP Protocol. 4 and 6 are valid. This diagram represents version 4 structure only.

Header Length

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

Protocol

IP Protocol ID. Including (but not limited to):

1 ICMP	17 UDP	57 SKIP
2 IGMP	47 GRE	88 EIGRP
6 TCP	50 ESP	89 OSPF
9 IGRP	51 AH	115 L2TP

Total Length

Total length of IP datagram, or IP fragment if fragmented. Measured in Bytes.

Fragment Offset

Fragment offset from start of IP datagram. Measured in 8 byte (2 words, 64 bits) increments. If IP datagram is fragmented, fragment size (Total Length) must be a multiple of 8 bytes.

Header Checksum

Checksum of entire IP header

IP Flags

x D M

x 0x80 reserved (evil bit)
D 0x40 Do Not Fragment
M 0x20 More Fragments follow

RFC 791

Please refer to RFC 791 for the complete Internet Protocol (IP) Specification.

- *Kryptographische* Prüfsummen sollen auch absichtliche Manipulationen erschweren

Kryptographische Hash-Funktionen: Grundlagen

- Hash-Funktionen
  - bilden „Universum“ auf endlichen Bildbereich ab
  - sind **nicht** injektiv
  - Bildbereich i.d.R. sehr viel kleiner als Universum
  - Kollisionen möglich:

$$\exists x, y \in U : x \neq y \quad \wedge \quad h(x) = h(y)$$

- *Kryptographische* Hash-Funktion H:
  - Eingabe: beliebig langes Wort m aus dem Universum U
  - Ausgabe: Hashwert H(m) mit fester Länge
  - H soll möglichst kollisionsresistent sein

Beispiel

- MD5-Hashwerte sind immer 128 Bits lang
  - egal, wie lange die Eingabe ist

```
829c11ba6dcdfe045dd1e5a77b34c05e  OOo-SDK_3.2.1_Linux_x86-64_install-deb_en-US.tar.gz
0f8abee370438e49e7ea0c2287589760  OOo-SDK_3.2.1_Linux_x86-64_install-rpm_en-US.tar.gz
35e8406c95c58b0087b9ad964faa13b8  OOo-SDK_3.2.1_Linux_x86_install-deb_en-US.tar.gz
ecc8271619ad788203cc61c7d9930522  OOo-SDK_3.2.1_Linux_x86_install-rpm_en-US.tar.gz
dddab486fd466bb1fc1a126d75919a3f  OOo-SDK_3.2.1_MacOS_x86_install_en-US.dmg
c4eb9f161736ba64933bdc81c274d8bc  OOo-SDK_3.2.1_Solaris_Sparc_install_en-US.tar.gz
3e8c88a645a8706e6c1bf7ef103bb993  OOo-SDK_3.2.1_Solaris_x86_install_en-US.tar.gz
7a7f4ea9173f9b8d9ae71cdf65c328f0  OOo-SDK_3.2.1_Win_x86_install_en-US.exe
```

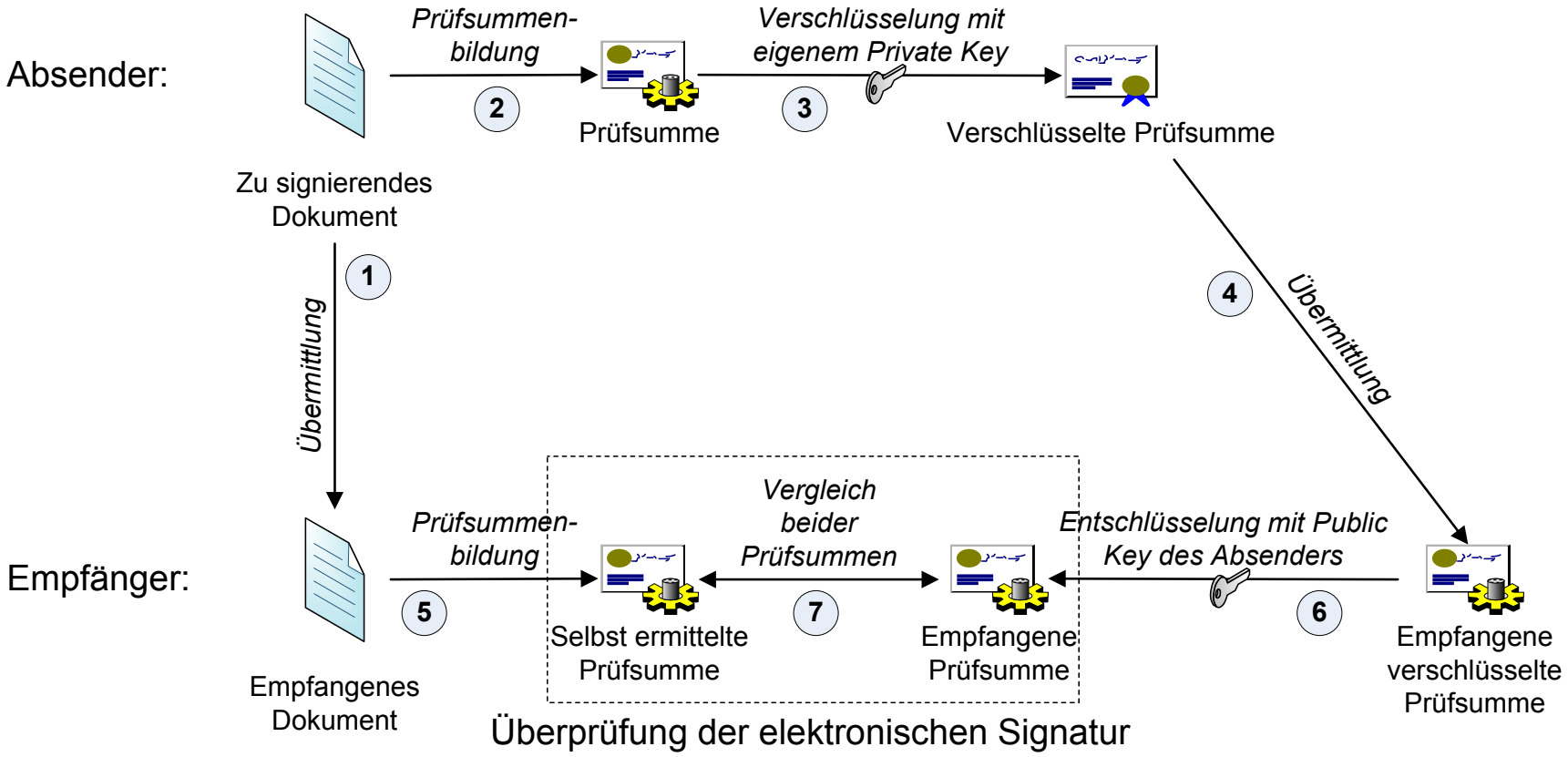
- Weil es nur 2^{128} verschiedene MD5-Hashwerte gibt, existieren beliebig viele Dateien mit demselben MD5-Hashwert
  - = Kollision
- Zwei sehr ähnliche, aber nicht identische Eingaben sollen nicht denselben MD5-Hashwert haben
  - = Kollisionsresistenz
- Angreifer versucht, die Nachricht *m* „sinnvoll“ in *m‘* abzuändern, so dass *md5(m) = md5(m‘)*

Einsatz kryptographischer Hash-Funktionen

- Integritätssicherung („Digitaler Fingerabdruck“):
  1. Alice erzeugt Nachricht m, berechnet H(m)=h und überträgt (m,h) an Bob (mindestens h muss gesichert werden, z.B. durch Verschlüsselung)
  2. Bob empfängt (m‘,h) und berechnet h‘=H(m‘)
  3. Falls h=h‘ kann davon ausgegangen werden, dass m=m‘, d.h. m wurde **nicht** verändert

- Digitale Signatur:
  - In der Praxis wird nicht die Nachricht m digital signiert
  - Stattdessen wird H(m) digital signiert {H(m)}
  - Übertragen wird dann (m,{H(m)})
  - Empfänger kann Quelle der Nachricht zweifelsfrei feststellen
  - Empfänger kann Integrität der Nachricht belegen

Signieren von Nachrichten und Signaturprüfung



Überprüfung der elektronischen Signatur

Def. Kryptographische Hashfunktion

- *Schwache* Hash-Funktion H:
  - H besitzt die Eigenschaften einer Einwegfunktion
  - Hashwert H(m) = h mit |h|=k (z.B. k = 128 Bits) ist bei gegebenem m einfach zu berechnen
  - Bei gegebenem h = H(m) für $m \in A_1^*$ ist es praktisch unmöglich, ein m‘ zu finden mit:

$$m' \neq m,\ m' \in A_1^* \ \wedge\ H(m') = h$$

- *Starke* Hash-Funktion H:
  - H hat alle Eigenschaften einer schwachen Hash-Funktion
  - Es ist zusätzlich praktisch unmöglich, eine Kollision zu finden, d.h. ein Paar verschiedene Eingabewerte m und m‘ mit:

$$m' \neq m,\ m, m' \in A_1^* \ \wedge\ H(m) = H(m')$$

Birthday Attack auf One-Way-Hash-Funktionen

- Wie viele Personen brauchen Sie, damit mit Wahrscheinlichkeit p > 0,5 eine weitere Person mit Ihnen Geburtstag hat?
  - Antwort: 253

$$P = 1 - \left(1 - \frac{1}{365}\right)^n$$

(ab n=253 ist P > 0,5)

- Wie viele Personen brauchen Sie, damit mit Wahrscheinlichkeit p > 0,5 zwei Personen am selben Tag Geburtstag haben?
  - Antwort: 23

$$P = 1 - \frac{n! \cdot \binom{365}{n}}{365^n}$$

(ab n=23 ist P > 0,5)

- Wie können Sie dieses Wissen für Angriffe gegen Hash-Funktionen nutzen?

Eine Kollision zu finden ist deutlich einfacher

als zu einem gegebenen Hash-Wert einen passenden Text!

Birthday Attack: Vorgehensweise

1. Alice sichert mit einem *k* Bits langen Hash eine Nachricht *M*
2. Mallet erzeugt $2^{k/2}$ Variationen der Nachricht *M*

- Die Wahrscheinlichkeit für eine Kollision ist größer 0,5.

- Wie können $2^{k/2}$ Variationen erzeugt werden?
  - Z.B. Einfügen von „Space – Backspace – Space“ Zeichen zwischen Wörtern
  - Wörter durch Synonyme ersetzen
  - .....

Beispiel für einen Brief mit 2^37 Variationen

- [Stal 98]

Dear Anthony,

{This letter is / I am writing} to introduce {you to / to you} {Mr. / --} Alfred {P. / --}

Barton, the {new / newly appointed} {chief / senior} jewellery buyer for {our / the}

Northern {European / Europe} {area / division}. He {will take / has taken} over {the / --}

responsibility for {all / the whole of} our interests in {watches and jewellery / jewellery and watches}

in the {area / region}. Please {afford / give} him {every / all the} help he {may need / needs}

to {seek out / find} the most {modern / up to date} lines for the {top / high} end of the

market. He is {empowered / authorized} to receive on our behalf {samples / specimens} of the

{latest / newest} {watch and jewellery / jewellery and watch} products, {up / subject} to a {limit / maximum}

of ten thousand dollars. He will {carry / hold} a signed copy of this {letter / document}

as proof of identity. An order with his signature, which is {appended / attached}

{authorizes / allows} you to charge the cost to this company at the {above / head office}

address. We {fully / --} expect that our {level / volume} of orders will increase in

the {following / next} year and {trust / hope} that the new appointment will {be / prove}

{advantageous / an advantage} to both our companies.

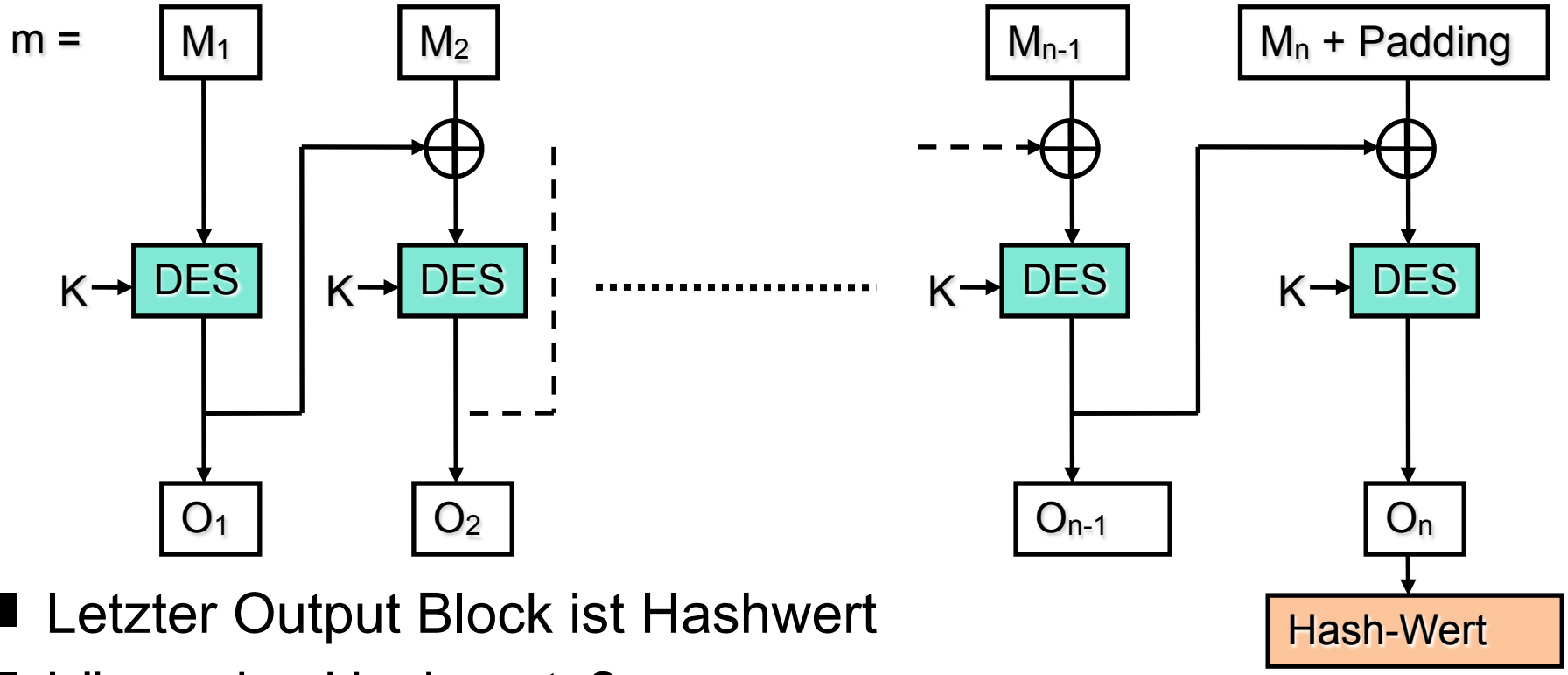
Konstruktion kryptographischer Hash-Funktionen

- Folge von Kompressionsfunktionen G
- Nachricht m wird in Blöcke M_i mit fester Länge y zerlegt
- Hash-Verfahren wird mit Initialisierungswert IV vorbelegt

- Letzter Block M_n muss ggf. auf vorgegebene Länge y „aufgefüllt“ werden (Padding)
- Als Kompressionsfunktion G können verwendet werden:
  - Hash-Funktionen auf der Basis symmetrischer Blockchiffren
  - Dedizierte Hash-Funktionen

DES als Kompressionsfunktion

- DES im Cipher Block Chaining (CBC) Mode



- Letzter Output Block ist Hashwert
- Länge des Hashwerts?

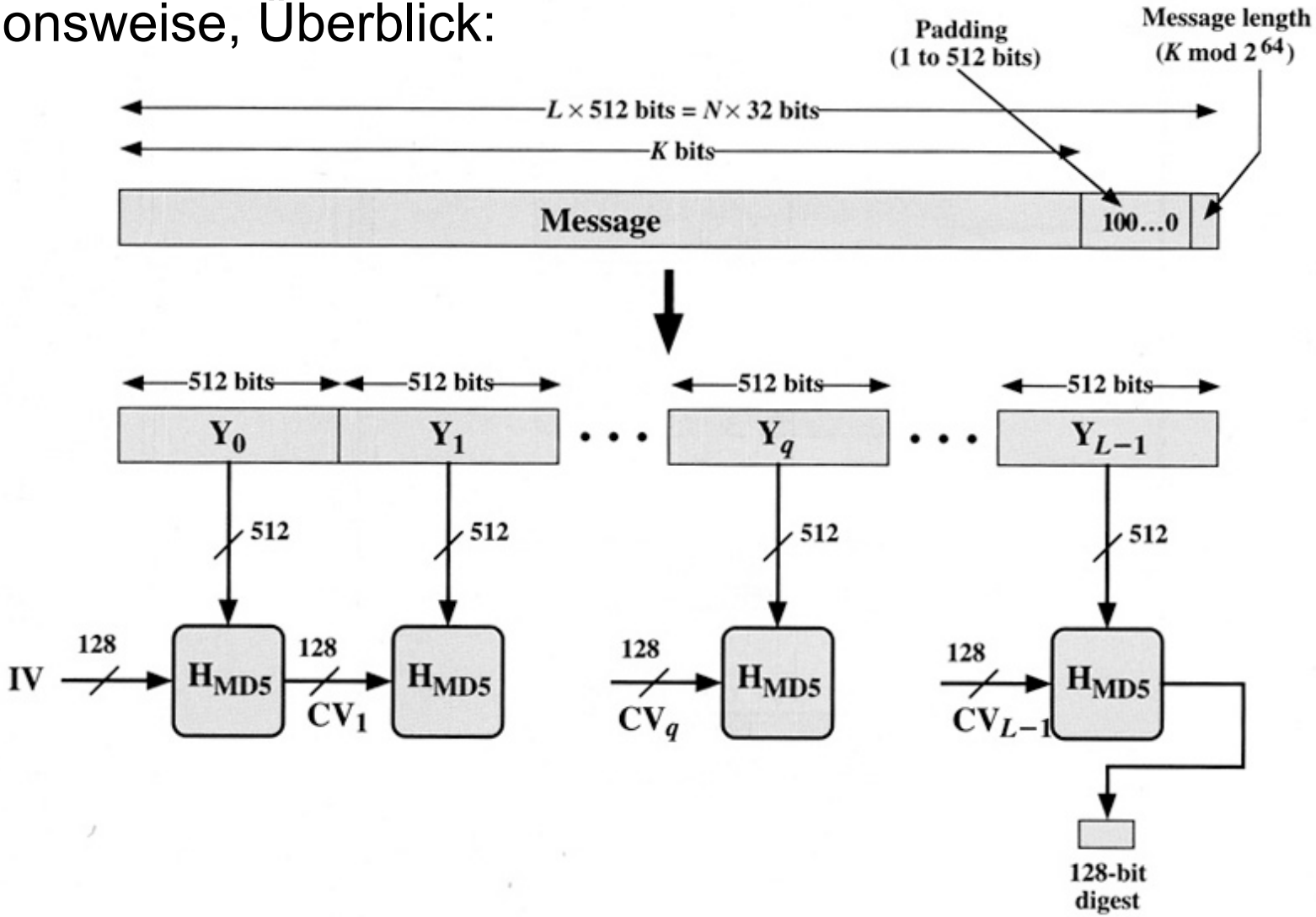
  64 Bits

Hash-Funktionen: MD4 (1990)

- Entwickelt von Ron Rivest: MD4 = Message Digest Nr. 4
- Design-Kriterien:
  - Kollisionsresistenz: Es gibt kein besseres Verfahren als Brute Force, um zwei Nachrichten mit demselben MD4-Hash zu finden
  - Direkte Sicherheit: MD4 basiert auf keinerlei (Sicherheits-)Annahmen wie z.B. dem Faktorisierungsproblem
  - Geschwindigkeitsoptimiert für Software-Implementierungen
  - Bevorzugt Little Endian 32-Bit-Architekturen (Intel)
  - Einfach und kompakt
- Erfolgreiche Angriffe zeigen mangelnde Kollisionsresistenz:
  - Boer und Bosselaers brechen die beiden letzten Runden der insges. drei
  - Merkle greift erfolgreich die ersten beiden Runden an
  - Angriff auf alle 3 Runden gelingt nicht
  - Konsequenz: Rivest verbessert MD4; Ergebnis ist MD5
- MD4 ist Public Domain (IETF RFC1320);
  sein Prinzip ist Basis für viele weitere Hash-Funktionen

Hash-Funktionen: MD5 (1991)

- Länge 128 Bit, arbeitet auf 512 Bit Blöcken
- Funktionsweise, Überblick:

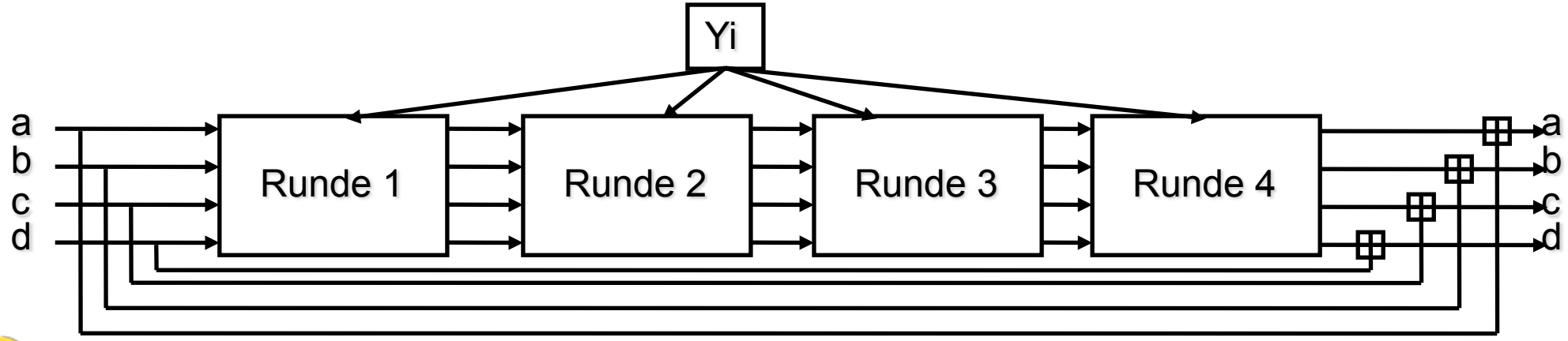


MD5 Ablauf

1. Padding Bits der Nachricht hinzufügen
2. Länge der Originalnachricht (mod 2^{64}) anfügen
3. Nachricht in 512-Bit-Blöcke aufteilen
4. Initialisierung von 32-Bit-Variablen:

   A = 0x01234567 C = 0xFEDCBA98

   B = 0x89ABCDEF D = 0x76543210
5. Zuweisung a=A, b=B, c=C, d=D
6. Kompressionsfunktion H_{MD5} angewendet auf jeden (Teil-)Block



MD5 Kompressionsfunktion (1)

- 4 Runden mit je einer nichtlinearen Funktion

$$F(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge (\neg Z))$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee (\neg Z))$$

  - Funktionen so gewählt, dass korrespondierende Bits von X, Y, Z und dem Ergebnis unabhängig voneinander sind

- In jeder Runde wird die Funktion 16 mal auf einen 32-Bit-Teilblock M_j von Y_i wie folgt angewendet

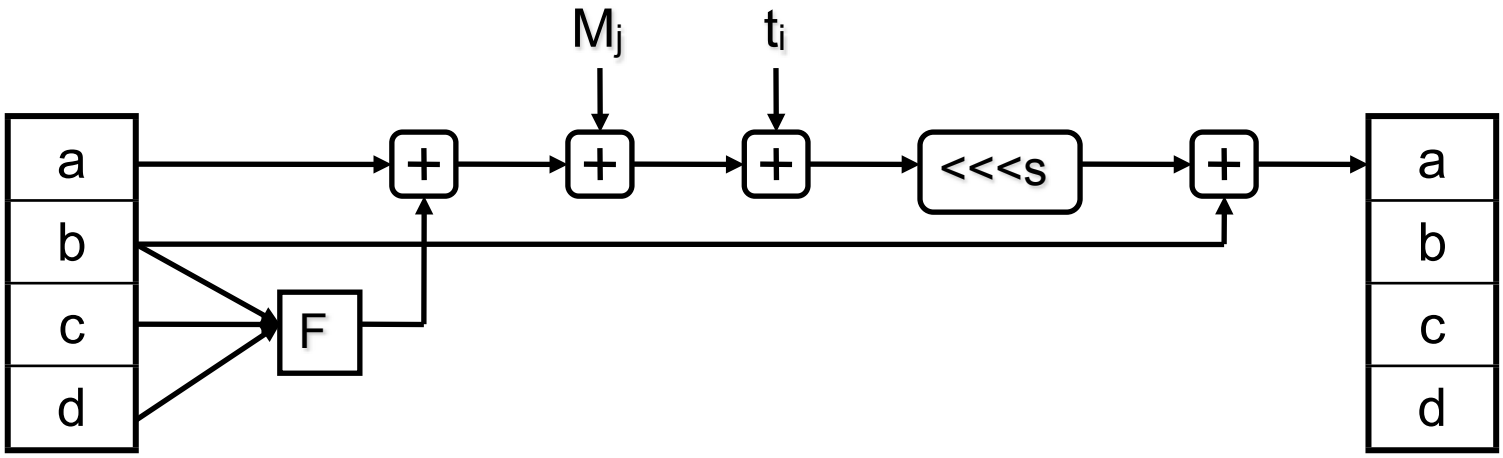
$$FF(a,b,c,d,M_j,s,t_i): \quad a = b + ((a + F(b,c,d) + M_j + t_i) <<< s)$$
$$GG(a,b,c,d,M_j,s,t_i): \quad a = b + ((a + G(b,c,d) + M_j + t_i) <<< s)$$
$$HH(a,b,c,d,M_j,s,t_i): \quad a = b + ((a + H(b,c,d) + M_j + t_i) <<< s)$$
$$II(a,b,c,d,M_j,s,t_i): \quad a = b + ((a + I(b,c,d) + M_j + t_i) <<< s)$$

MD5 Kompressionsfunktion (2)

- $FF(a,b,c,d,M_j,s,t_i): \quad a = b + ((a + F(b,c,d) + M_j + t_i) <<< s)$
  - + bezeichnet Addition modulo 2^{32}
  - $t_i = 2^{32} abs(\sin(i))$ mit i Grad im Bogenmaß; 0 <= i < 64 (i über 4 Runden)
  - <<< s bezeichnet zirkulären Shift um *s* Bits (s variiert pro Operation)
  - Auswahl des Teilblocks M_j (Runde 1: i = 0..15, ..., Runde 4: i = 48..63)

Runde 1	Natürliche Ordnung	Runde 3	$(5 + 3i) \bmod 16$
Runde 2	$(1 + 5i) \bmod 16$	Runde 4	$7i \bmod 16$

- Beispiel: Elementarer Schritt in Runde 1 auf 32-Bit-Block



MD5: Rundenfunktion; 4 Runden mit 64 Schritten

- Runde 1:
  1. FF(a, b, c, d, M0, 7, 0xd76aa478)
  2. FF(d, a, b, c, M1, 12, 0xe8c7b756)
  3. FF(c, d, a, b, M2, 17, 0x242070db)
  4. FF(b, c, d, a, M3, 22, 0xc1bdceee)

- Runde 4:
  60. II(a, b, c, d, M4, 6, 0xf7537e82)
  61. II(d, a, b, c, M11, 10, 0xbd3af235)
  62. II(c, d, b, a, M2, 15, 0x2ad7d2bb)
  63. II(b, c, d, a, M9, 21, 0xeb86d391)

Sicherheit von MD5

- Differentielle Kryptanalyse auf MD5 mit nur einer Runde [Bers 92]:
  - Für jede der 4 Runden einzeln möglich
  - Angriff auf alle 4 Runden konnte nicht gezeigt werden
- Pseudokollision [BoBo 93]:
  - Zwei verschiedene Variablenbelegungen von a,b,c,d führen für verschiedene Inputblöcke zum gleichen Outputblock
  - Damals schien eine Erweiterung des Ansatzes zu einem allgemeinen Angriff nicht möglich
- Erzeugung einer Kollision in der Kompressionsfunktion [Dobb 96]:
  - Zwei 512 Bit Blöcke produzieren den selben 128 Bit Output
  - Bis dahin gefährlichster bekannter Angriff
  - Bisher kein Mechanismus zur Generalisierung des Angriffs auf gesamten MD5 mit IV gefunden

Sicherheit von MD5 (Forts.)

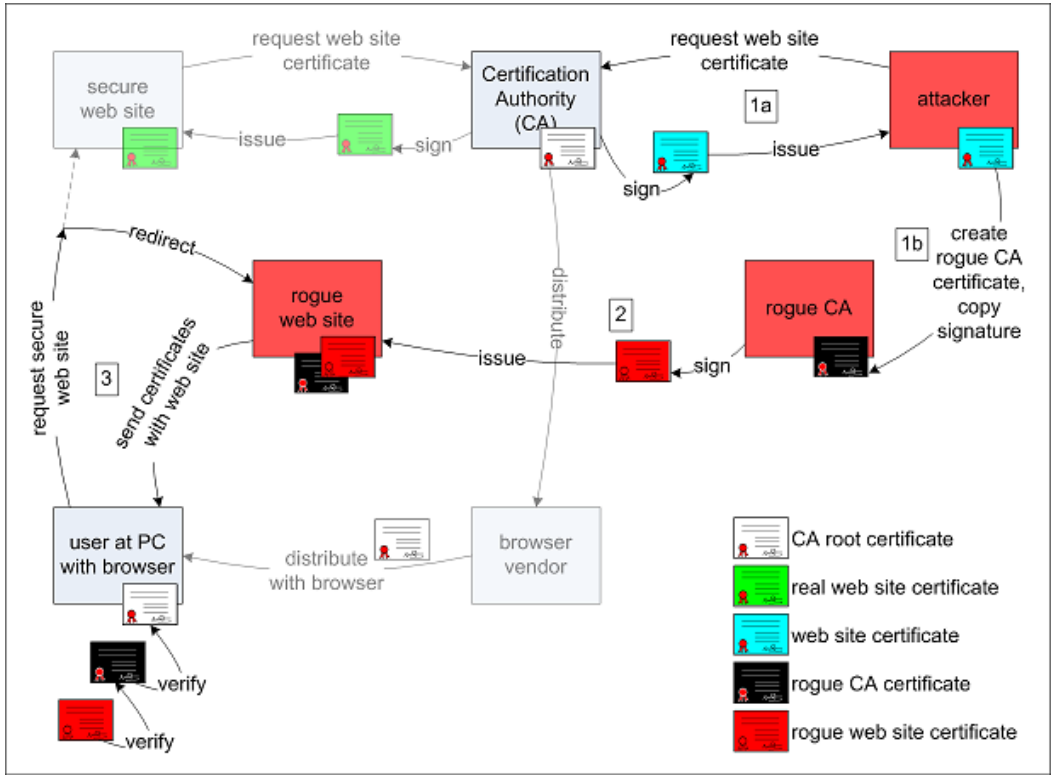
- Kollision gefunden [Wang,Feng,Lai,Yu 2004]:
  - MD5(M,N_i) = MD5(M‘,N_i‘)
  - M und M‘ zu finden dauert ca. eine Stunde (IBM P690 Cluster)
  - danach N_i und N_i‘ zu finden 15 Sek. bis 5 Minuten
  - funktioniert mit beliebigen Initialisierungsvektor IV
  - In der Arbeit werden auch Kollisionen für MD4, HAVAL-128 und RIPEMD-128 angegeben
  - Ende des MD5CRK-Projekts (distributed birthday attack)
- Kollision in X.509 Zertifikat gefunden (Kollision in den Schlüsseln) [de Weger 2005]
- Kollision in X.509 Zertifikat mit unterschiedlichen Identitäten [Stevens, Lenstra, de Wegener 2006/2007]

➡ MD5 (und SHA-1) nicht mehr verwenden!

➡ Algorithmen mit längeren Hash-Werten verwenden:
z.B. SHA-224, SHA-256, SHA-384, SHA-512, Whirlpool, o.ä.

Sicherheit von MD5 (Forts.)

- Bislang umfangreichster, praktisch relevanter Angriff [SSALMOW08]: http://www.win.tue.nl/hashclash/rogue-ca/



- Alle Browser, die RapidSSL-Zertifikaten vertrauten, vertrauten auch den mit dem „rogue CA certificate“ ausgestellten Zertifikaten
- Man-in-the-Middle Angriffe: Browser kann bei SSL-Zertifikaten, die MD5-Hashsummen verwenden, die Server-Authentizität nicht mehr zuverlässig prüfen

Backdoor in FTP-Server ProFTPD (02.12.2010)

- Sicherheitsloch in FTP-Serversoftware ProFTPD ermöglicht Angreifern Zugang zum ProFTPD-Projektserver
- Angreifer bauen Backdoor in ProFTPD ein:
  Spezieller FTP-Befehl startet Root-Shell
- Manipulierter Quelltext wird an alle Mirrors übertragen
- Entwickler bemerken das Problem drei Tage später und entfernen das Backdoor

- Offizielle Empfehlung: Anhand der MD5-Hashwerte prüfen, ob man eine Quelltext-Version ohne Backdoor hat

- Weitere Infos: http://www.proftpd.org/md5_pgp.html

GNU-Server gehackt (29.11.2010)

- Angreifer mit Georgischer IP-Adresse nutzte SQL Injection, um MD5-Passwort-Hashes von savannah.gnu.org zu erhalten (24.11.2010)
- Passwörter von Accounts einiger GNU-Projekte und mind. ein Admin-Account vermutlich per Brute Force geknackt (26.11.2010)
- Passwörter für „Vandalismus“ missbraucht (Defacement von www.gnu.org)
- Betreiber spielen am 29.11.2010 das Backup vom 23.11.2010 ein
- Alle Benutzer müssen ihre Passwörter ändern

- Weitere Infos:
  http://www.fsf.org/blogs/sysadmin/savannah-and-www.gnu.org-downtime

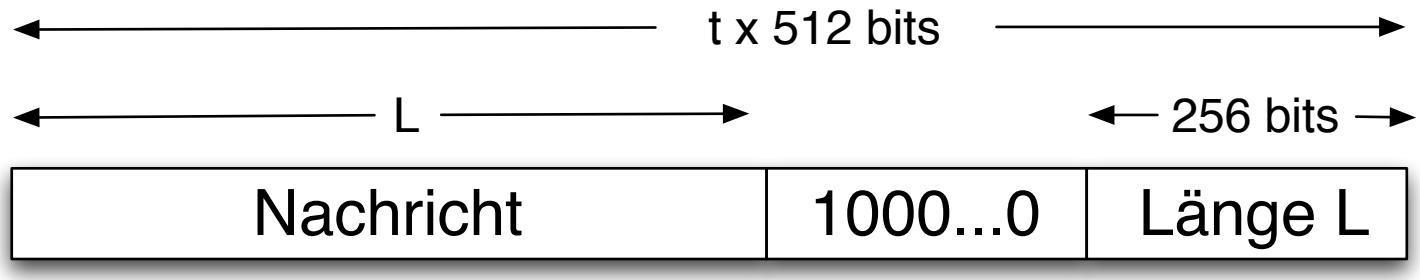
Whirlpool Hash-Funktion (2000)

- Entwickelt von P. Barreto und V. Rijmen
- im Rahmen des europäischen NESSI (**N**ew **E**uropean **S**chemes for **S**ignatures, **I**ntegrity, and **E**ncryption) entwickelt
- Struktur ähnlich zu AES, bzw. Rijndael
- Inzwischen Industriestandard im Rahmen von ISO/IEC 10118-3
  - Whirlpool ist nicht patentiert, die Referenzimplementierung ist Open Source
- 512 Bit lange Hashwerte bei max. Nachrichtenlänge 2^{256} Bits
- Design-Ziele:
  - Kollision zu finden benötigt $2^{n/2}$ Whirlpool-Operationen
  - Zu gegebenem Hash h eine Nachricht x zu finden mit h = H(x) benötigt 2^n Whirlpool-Operationen
  - Zu gegebenen Hash h und geg. Nachricht m eine Nachricht x mit selbem Hashwert zu finden benötigt 2^n Whirlpool-Operationen
- Weit verbreitete Nutzung z.B. im Rahmen von TrueCrypt (2005)

Whirlpool: Überblick

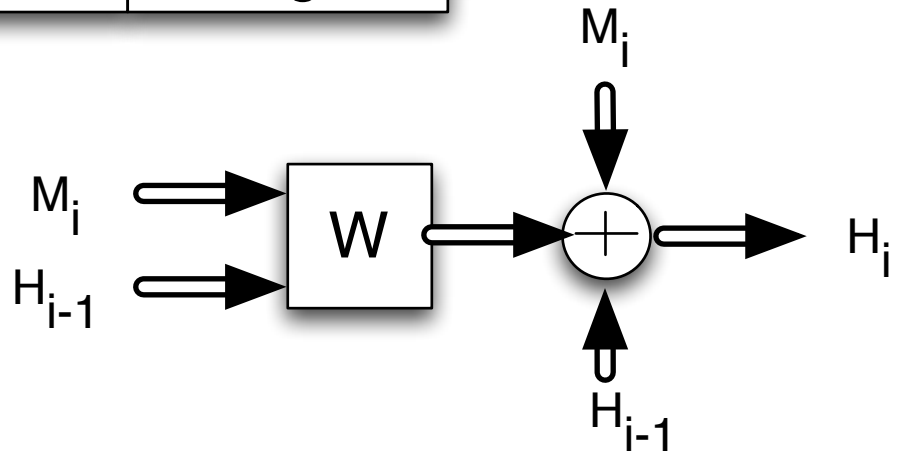
- Whirlpool (WP(m)) arbeitet auf 512 Bit langen Teilblöcken M_i
- Verwendet Block Chiffre W
- Arbeitet intern mit 8x8 Byte Matrix CState

1. Expansion von m auf ein Vielfaches von 512 Bit; Aufteilen in Nachrichtenblöcke M_0 bis M_{t-1}



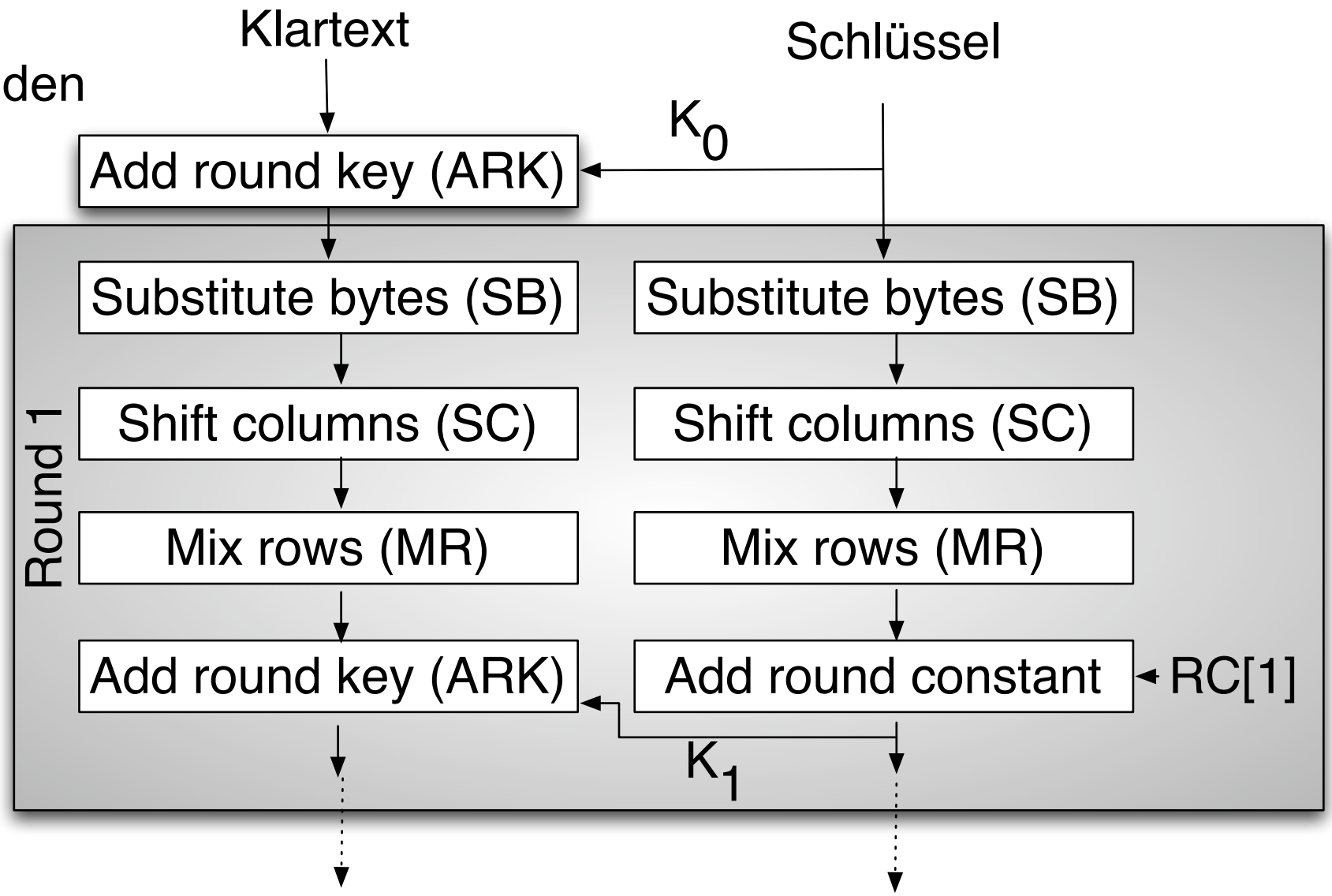
2. Initialisierung von $H_o = 0$
3. For $0 \le i \le t-1$

   $H_i = W_{Hi-1}(M_i) \oplus M_i \oplus H_{i-1}$
4. $WP(m) = H_t$

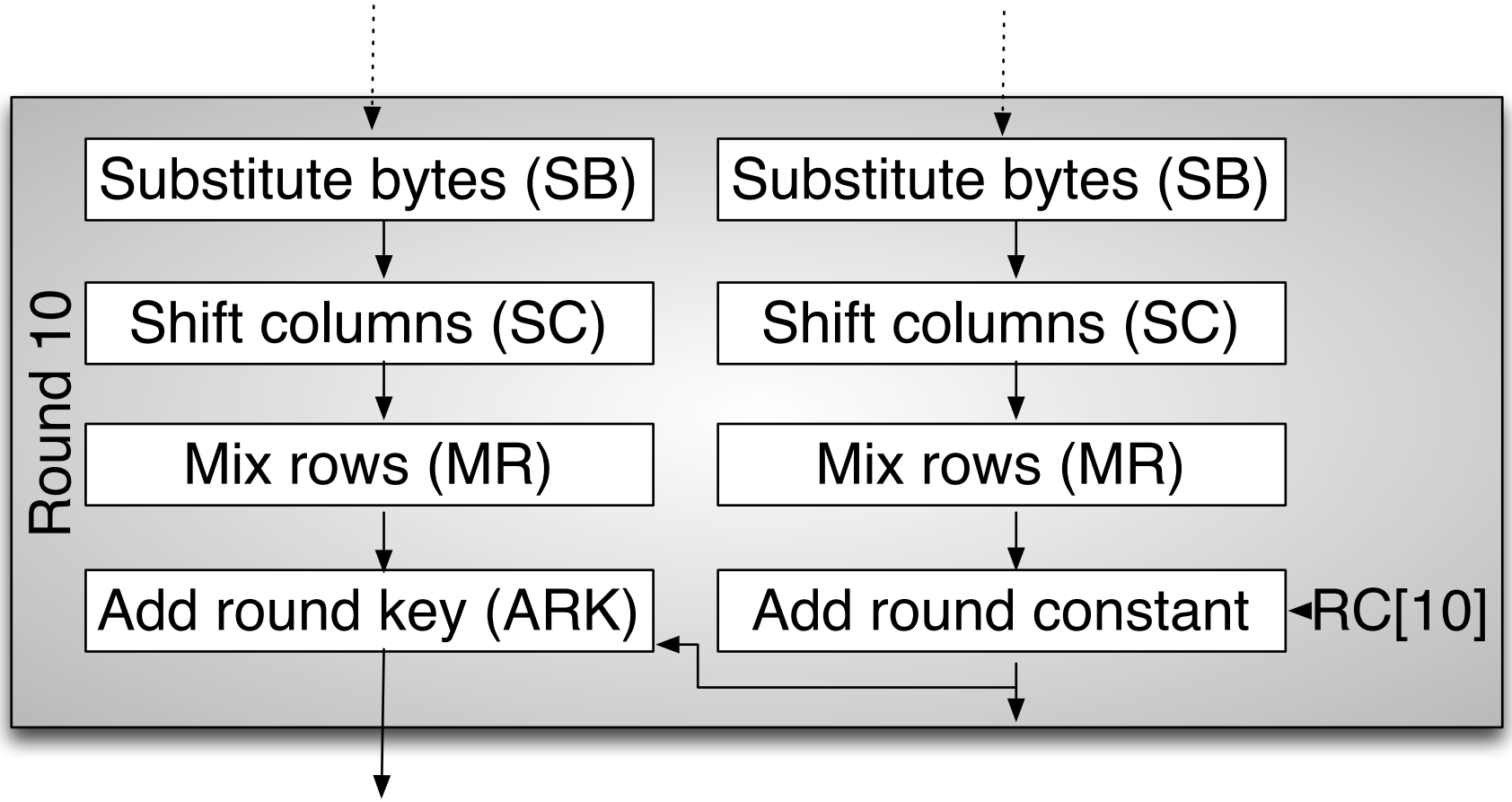


Whirlpool: Block Chiffre W

- Immer 10 Runden



Whirlpool: Block Chiffre W; Fortsetzung



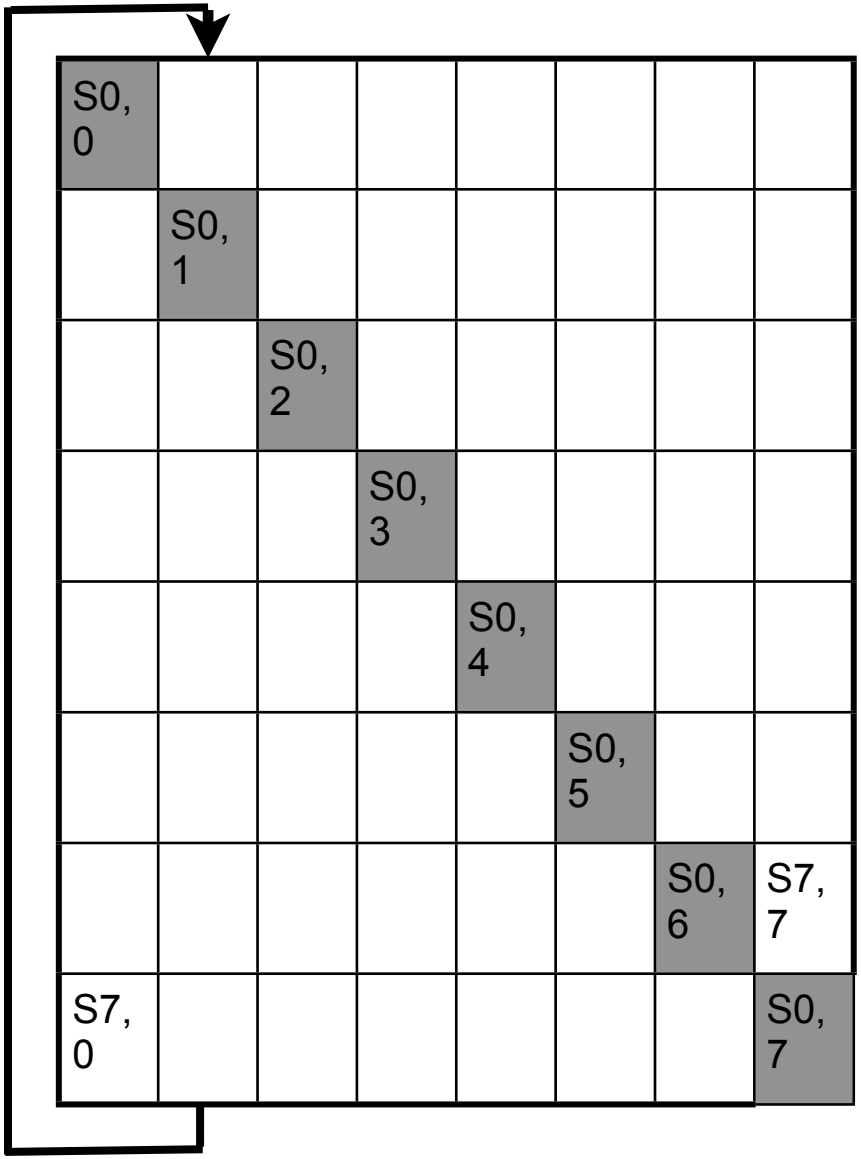
Whirlpool: Byte Substitution

- Mittels S-Box: auf Matrix-Elementen von CState
- 4 linken Bit bestimmen Spalte; 4 rechten Bit die Zeile

	00_x	01_x	02_x	03_x	04_x	05_x	06_x	07_x	08_x	09_x	$0A_x$	$0B_x$	$0c_x$	$0d_x$	$0E_x$	$0F_x$
00_x	18_x	23_x	$c6_x$	$E8_x$	87_x	$B8_x$	01_x	$4F_x$	36_x	$A6_x$	$d2_x$	$F5_x$	79_x	$6F_x$	91_x	52_x
10_x	60_x	Bc_x	$9B_x$	$8E_x$	$A3_x$	$0c_x$	$7B_x$	35_x	$1d_x$	$E0_x$	$d7_x$	$c2_x$	$2E_x$	$4B_x$	FE_x	57_x
20_x	15_x	77_x	37_x	$E5_x$	$9F_x$	$F0_x$	$4A_x$	dA_x	58_x	$c9_x$	29_x	$0A_x$	$B1_x$	$A0_x$	$6B_x$	85_x
30_x	Bd_x	$5d_x$	10_x	$F4_x$	cB_x	$3E_x$	05_x	67_x	$E4_x$	27_x	41_x	$8B_x$	$A7_x$	$7d_x$	95_x	$d8_x$
40_x	FB_x	EE_x	$7c_x$	66_x	dd_x	17_x	47_x	$9E_x$	cA_x	$2d_x$	BF_x	07_x	Ad_x	$5A_x$	83_x	33_x
50_x	63_x	02_x	AA_x	71_x	$c8_x$	19_x	49_x	$d9_x$	$F2_x$	$E3_x$	$5B_x$	88_x	$9A_x$	26_x	32_x	$B0_x$
60_x	$E9_x$	$0F_x$	$d5_x$	80_x	BE_x	cd_x	34_x	48_x	FF_x	$7A_x$	90_x	$5F_x$	20_x	68_x	$1A_x$	AE_x
70_x	$B4_x$	54_x	93_x	22_x	64_x	$F1_x$	73_x	12_x	40_x	08_x	$c3_x$	Ec_x	dB_x	$A1_x$	$8d_x$	$3d_x$
80_x	97_x	00_x	cF_x	$2B_x$	76_x	82_x	$d6_x$	$1B_x$	$B5_x$	AF_x	$6A_x$	50_x	45_x	$F3_x$	30_x	EF_x
90_x	$3F_x$	55_x	$A2_x$	EA_x	65_x	BA_x	$2F_x$	$c0_x$	dE_x	$1c_x$	Fd_x	$4d_x$	92_x	75_x	06_x	$8A_x$
$A0_x$	$B2_x$	$E6_x$	$0E_x$	$1F_x$	62_x	$d4_x$	$A8_x$	96_x	$F9_x$	$c5_x$	25_x	59_x	84_x	72_x	39_x	$4c_x$
$B0_x$	$5E_x$	78_x	38_x	$8c_x$	$d1_x$	$A5_x$	$E2_x$	61_x	$B3_x$	21_x	$9c_x$	$1E_x$	43_x	$c7_x$	Fc_x	04_x
$c0_x$	51_x	99_x	$6d_x$	$0d_x$	FA_x	dF_x	$7E_x$	24_x	$3B_x$	AB_x	cE_x	11_x	$8F_x$	$4E_x$	$B7_x$	EB_x
$d0_x$	$3c_x$	81_x	94_x	$F7_x$	$B9_x$	13_x	$2c_x$	$d3_x$	$E7_x$	$6E_x$	$c4_x$	03_x	56_x	44_x	$7F_x$	$A9_x$
$E0_x$	$2A_x$	BB_x	$c1_x$	53_x	dc_x	$0B_x$	$9d_x$	$6c_x$	31_x	74_x	$F6_x$	46_x	Ac_x	89_x	14_x	$E1_x$
$F0_x$	16_x	$3A_x$	69_x	09_x	70_x	$B6_x$	$d0_x$	Ed_x	cc_x	42_x	98_x	$A4_x$	28_x	$5c_x$	$F8_x$	86_x

Whirlpool: Shift Column

S0, 0	S0, 1	S0, 2	S0, 3	S0, 4	S0, 5	S0, 6	S0, 7
S7, 0							S7, 7



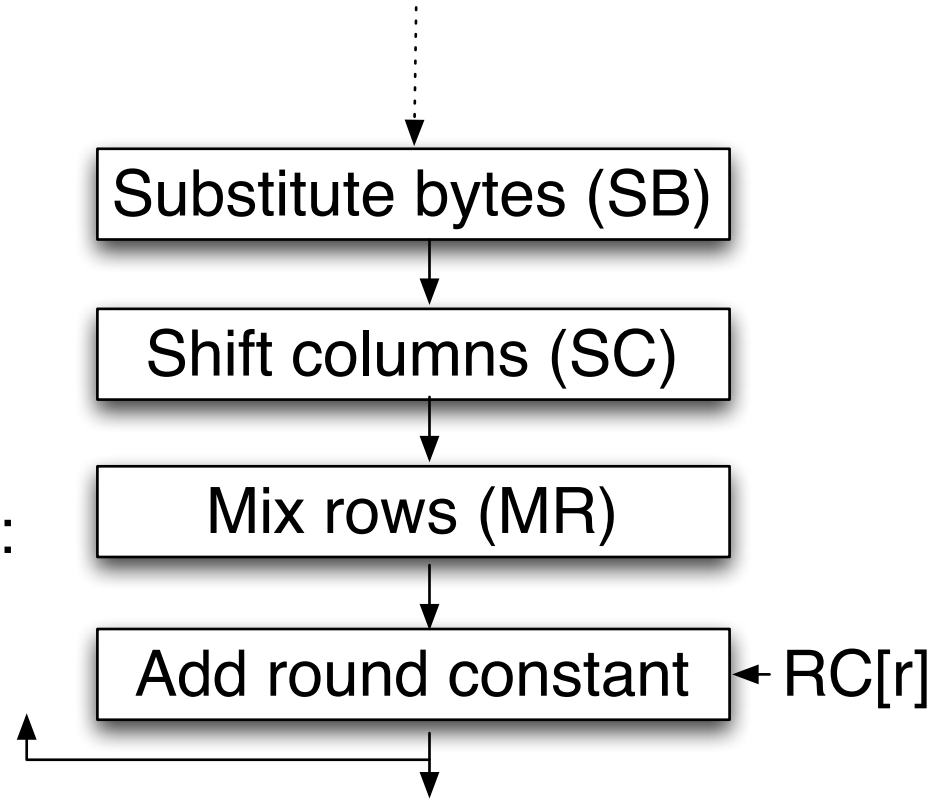
Whirlpool: Mix Row

- Matrixmultiplikation für jede Zeile von CState

$$\begin{bmatrix} s'_{i,0} \\ s'_{i,1} \\ s'_{i,2} \\ s'_{i,3} \\ s'_{i,4} \\ s'_{i,5} \\ s'_{i,6} \\ s'_{i,7} \end{bmatrix}^T = \begin{bmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,3} \\ s_{i,4} \\ s_{i,5} \\ s_{i,6} \\ s_{i,7} \end{bmatrix}^T \cdot A = \begin{bmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,3} \\ s_{i,4} \\ s_{i,5} \\ s_{i,6} \\ s_{i,7} \end{bmatrix}^T \cdot \begin{bmatrix} 01_x & 01_x & 03_x & 01_x & 05_x & 08_x & 09_x & 05_x \\ 05_x & 01_x & 01_x & 03_x & 01_x & 05_x & 08_x & 09_x \\ 09_x & 05_x & 01_x & 01_x & 03_x & 01_x & 05_x & 08_x \\ 08_x & 09_x & 05_x & 01_x & 01_x & 03_x & 01_x & 05_x \\ 05_x & 08_x & 09_x & 05_x & 01_x & 01_x & 03_x & 01_x \\ 01_x & 05_x & 08_x & 09_x & 05_x & 01_x & 01_x & 03_x \\ 03_x & 01_x & 05_x & 08_x & 09_x & 05_x & 01_x & 01_x \\ 01_x & 03_x & 01_x & 05_x & 08_x & 09_x & 05_x & 01_x \end{bmatrix}$$

Whirlpool: Add round key / constant

- CState wird mit K_i XOR-verknüpft
- Berechnung von K_i



- Berechnung von RC[r] für Runde r:

  RC[0] = K

  for $1 \le r \le 10$

  $RC[r]_{0,j} = Sbox[8(r-1)+j]$ für $0 \le j \le 7$ u. $1 \le r \le 10$

  $RC[r]_{i,j} = 0$ für $1 \le i \le 7$, $0 \le j \le 7$ u. $1 \le r \le 10$

Vergleich Rijndael und W

	Rijndael	W
Blockgröße [bits]	128, 160, 192, 224 oder 256	immer 512
Rundenanzahl	10, 11, 12, 13 oder 14	immer 10
Schlüsselauswahl	ausgezeichneter Algorithmus (a priori)	Rundenfunktion von W
Reduktionspolynom	$x^8+x^4+x^3+x+1$ (0x11b)	$x^8+x^4+x^3+x^2+1$ (0x11d)

Sicherheit von Whirlpool

- Algorithmus wurde im Rahmen des NESSIE-Projekts evaluiert
- Versteckte Schwächen wurden nicht gefunden
- Statistische Analyse: Whirlpool sollte stochastisch sein
- 512 Bit Länge bietet sonst nur SHA-512
- Resistent gegenüber bekannten Angriffen
- Bisher kein erfolgreicher Angriff

- ABER: Algorithmus noch relativ jung und bisher noch nicht so weit verbreitet

Banken sperren Kreditkarten nach Datenleck bei Onlinehändler (30.11.2010)

- Postbank und ING-DiBa sperren Visa- und Mastercard-Karten von ca. 1% ihrer Kunden; Austausch hat begonnen.
- Hintergrund ist ein Datendiebstahl bei einem nicht näher genannten (!) deutschen Online-Händler
- Vergleichbare Aktion im November 2009:
  - Volks- und Raiffeisenbanken, Sparkassen, Deutsche Bank und Sparkassen sperren mehr als 60.000 Kreditkarten
  - Datendiebstahl bei einem Zahlungs-Dienstleister in Spanien
- Positive Darstellung:
  Austausch der Karte ist für den Kunden kostenlos