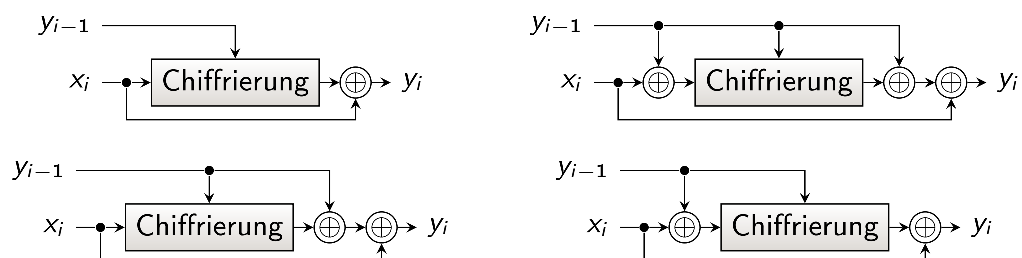
11 Einweg-Hash-Funktionen

- bildet eine Nachricht $\boldsymbol{x}$ beliebiger Länge auf einen Wert $\boldsymbol{y} = H(\boldsymbol{x})$ fester (kürzerer) Länge ab
- zu gegebenem $\boldsymbol{y}$ ist es schwer, ein $\boldsymbol{x}$ zu finden, sodass $H(\boldsymbol{x}) = \boldsymbol{y}$
- zu gegebenem $\boldsymbol{x}$ ist es schwer, ein $\boldsymbol{x}'$ zu finden, sodass $H(\boldsymbol{x}') = H(\boldsymbol{x})$
- es ist es schwer, ein Paar $\boldsymbol{x}$ und $\boldsymbol{x}'$ ($\boldsymbol{x} \neq \boldsymbol{x}'$) zu finden, sodass $H(\boldsymbol{x}') = H(\boldsymbol{x})$
- Anwendung: digtale Signaturen
  - statt Nachricht wird Hash der Nachricht signiert
  - schneller
  - Klartext-Nachricht + kurze Signatur
- Anwendung: Passwort-Datenbank
  - nur Hashes der Passwörter werden gespeichert
  - vom Benutzer eingegebenes Passwort wird gehasht und dann mit Datenbank verglichen
  - erbeutet ein Angreifer die Passwort-Datenbank, kann er die Passwörter nicht rekonstruieren
  - "Salz" zur Erhöhung der Sicherheit: Passwort wird um zufälliges "Salz" erweitert, bevor es (und das Salz) in Datenbank gespeichert wird $\Rightarrow$ gleiche Passwörter liefern unterschiedliche Hashes; keine Tabelle von Hashes zu häufigen Passwörtern aufstellbar

11.1 Hash-Funktionen aus Blockchiffren

- diverse Konstellationen möglich, folgende vier gelten für geeigenete Blockchiffren mit Blocklänge=Schlüssellänge als sicher:



- $x_1, x_2, \ldots, x_K$: Blöcke der Nachricht, y_0: Initialisierungsvektor, y_K: berechneter Hash
- sicher, aber spezielle Hash-Funktionen deutlich schneller

11.2 MD5

- Nachricht wird in 512 bit-Blöcke zerlegt, diese jeweils in 16 Teilblöcke zu 32 bit
- zu bildender Hash von 128 bit wird in 4 Teilblöcke A, B, C, D zu 32 bit zerlegt
- jeder Block wird in 4 Runden verarbeitet
- jede Runde besteht aus 16 Schritten der Form

$$a \leftarrow b + ((a + f(b, c, d) + x_j + t) <<< s)$$

  - in jedem Schritt andere Zuordnung zwischen A, B, C, D und a, b, c, d
  - in jedem Schritt anderer Nachrichten-Teilblock x_j
  - in jedem Schritt andere Konstanten t und s
  - in jeder Runde andere bitweise angewendete Funktion f:
    1. Runde: $f(b, c, d) = (b \wedge c) \vee (\neg b \wedge d)$
    2. Runde: $f(b, c, d) = (b \wedge d) \vee (c \wedge \neg d)$
    3. Runde: $f(b, c, d) = b \oplus c \oplus d$
    4. Runde: $f(b, c, d) = c \oplus (b \vee \neg d)$
- das Ergebnis nach allen vier Runden wird zu den ursprünglichen A, B, C, D addiert

11.3 SHA-1

- verwandt mit MD5
- Nachricht wird in 512 bit-Blöcke zerlegt, diese jeweils in 16 Teilblöcke zu 32 bit
- zu bildender Hash von 160 bit wird in 5 Teilblöcke A, B, C, D, E zu 32 bit zerlegt
- jeder Block wird in 4 Runden verarbeitet
- jede Runde besteht aus 20 Schritten der Form

$$\begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} \leftarrow \begin{pmatrix} (a <<< 5) + f_t(b, c, d) + e + W_t + K_t \\ a \\ b <<< 30 \\ c \\ d \end{pmatrix}$$

  - zu Beginn $a = A, b = B, \ldots, e = E$
  - in Schritten $t = 0$ bis 15: $W_t = x_t$
  - in Schritten $t = 16$ bis 79: $W_t = W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}$
  - in jeder Runde andere Konstante K_t
  - in jeder Runde andere bitweise angewendete Funktion f_t:
    1. Runde: $f_t(b, c, d) = (b \wedge c) \vee (\neg b \wedge d)$
    3. Runde: $f_t(b, c, d) = (b \wedge c) \vee (c \wedge d) \vee (d \wedge b)$
    2. und 4. Runde: $f_t(b, c, d) = b \oplus c \oplus d$
- das Ergebnis nach allen vier Runden wird zu den ursprünglichen A, B, C, D, E addiert

11.4 SHA-2

- ähnliche Struktur wie MD5 und SHA-1
- beseitigt (eher theoretische) Schwächen von SHA-1
- erlaubt Hashes der Längen 224, 256, 384 und 512 bit (auch SHA-224, SHA-256, SHA-384 und SHA-512 genannt)

11.5 Regenbogentabellen

- werden in einer Passwort-Datenbank Hashes ohne Salz gespeichert, so existiert ein effektiver Angriff der vorberechnete Daten nutzt
- beschränkt man sich auf kurze Passwörter, kann man eine Tabelle mit allen Passwörtern und den zugeordneten Hashes aufstellen (5 Kleinbuchstaben: $26^5 = 11\,881\,376$ Möglichkeiten)
- für längere Passwörter mit Sonderzeichen schlägt dieser Ansatz fehl, da die Tabelle zu groß würde (10 Buchstaben, Zahlen Sonderzeichen: $80^{10} = 1{,}0737 \cdot 10^{19}$ Möglichkeiten)
- eine Möglichkeit, die Tabellen zu verkleinern (aber nicht den Zeitaufwand zur Erzeugung!), sind Hashketten
- definiere eine Reduktionsfunktion R, die einen Hash-Wert auf ein mögliches Passwort abbildet

- Aufstellen der Tabelle:
  - wähle so viele mögliche Passwörter, wie die angepeilte Tabellengröße erlaubt
  - berechne zu jedem Passwort den Hash, davon die Reduktionsfunktion, davon wieder den Hash usw.
  - brich diese Ketten nach N Schritten ab; speichere nur urspüngliches Passwort (Startpunkt) und letzten Hash (Endpunkt)

$$\text{aaaaaa} \xrightarrow{H} \text{0xDE311F37} \xrightarrow{R} \text{baoirq} \xrightarrow{H} \text{0x2B89E789} \xrightarrow{R} \text{aeuklr} \dashrightarrow \text{0x30B9CA6A}$$
$$\text{aaaaab} \xrightarrow{H} \text{0xA9CD02EA} \xrightarrow{R} \text{weuirg} \xrightarrow{H} \text{0x0020539C} \xrightarrow{R} \text{vkazwk} \dashrightarrow \text{0x9E7B45DD}$$
$$\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots$$

- Suchen eines Passworts zu einem Hash:
  - wende solange wiederholt R und H an, bis Hashwert als Endpunkt in Tabelle zu finden (maximal N mal, mindestens 0 mal)
  - rekonstruiere Kette von zugehörigem Startpunkt, bis ursprünglicher Hash gefunden; Passwort aus Schritt davor ist Lösung
- im Idealfall Tabelle um Faktor N kleiner als vollständige Auslistung aller Passwort-Hash-Kombinationen und Passwort in N Schritten zu finden
- Problem: da weder H noch R injektiv können verschiedene Ketten zusammenlaufen
  - Tabelle mit M Startpunkt-Endpunkt-Paaren deckt deutlich weniger als $N \cdot M$ Passwörter ab
  - falsche Alarme bei Suche (Startpunkt führt nicht auf richtigen Hash)

- insbesondere das Zusammenlaufen verschiedener Ketten für größere N reduziert die Effektivität diese Verfahrens drastisch
- Abhilfe schaffen die Regenbogentabellen, die in jedem Schritt eine anderen Reduktionsfuktion R_n benutzen
- liefern zwei Ketten an unterschiedlichen Stellen den gleichen Wert, trennen sie sich wieder
- liefern zwei Ketten an der gleichen Stelle den gleichen Wert, führen sie auf den gleichen Endpunkt und lassen sich so leicht aufspüren und ersetzen
- bei der Suche müssen alle N möglichen Startstellen geprüft werden, also $\mathcal{O}(N^2)$ Hashes berechnet und auf ihre Existenz in der Tabelle geprüft werden