CRYPTO2-Übersicht: VL10+11: Hashfunktionen

↳ Def. Einwegfunktion, kollisionsfrei, schwach kollisionsfrei/zweiturbildfrei; Merkle-Damgård; Keccak und Zusammenhänge

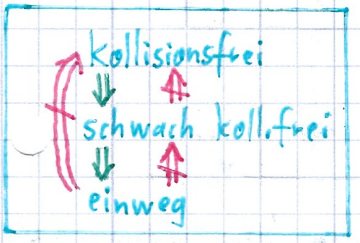
- **Def. Hash:** $h: \mathbb{B}^\infty \to \mathbb{B}^n$ Hashfunktion der Länge n
  √ h effektiv berechenbar √ Urbild „schwer" bestimmbar konstruierbar
  √ keine 2 Dokumente mit selbem Fingerabdruck/kein 2. Dok. mit selbem FA

- **Def. Einwegfkt:** zu $z \in_R \mathbb{B}^n$ ist es „praktisch unmöglich" ein $x \in \mathbb{B}^\infty$ mit $h(x)=z$ zu finden

- **Def. kollisionsfrei:** es ist „praktisch unmöglich" $x \neq x' \in \mathbb{B}^\infty$ mit $h(x)=h(x')$ zu finden

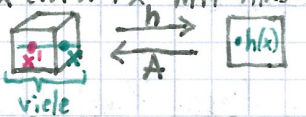
- **Def. schwach kollisionsfrei/2nd preimage resistant/zweiturbildfrei:** (...ist schwieriger als...)
  es ist „praktisch unmöglich" zu geg. $x \in \mathbb{B}^\infty$ ein $x' \neq x \in \mathbb{B}^\infty$ mit $h(x)=h(x')$ zu finden

- **Zusammenhänge:**



(1) kollisionsfrei ⇒ schwach kollisionsfrei:
Kontraposition: nicht schwach kollisionsfrei ⇒ ich finde zu einem geg. $x \in \mathbb{B}^\infty$ relativ leicht ein $x' \neq x$ mit $h(x)=h(x')$ ⇒ nicht kollisionsfrei ■

(2) schwach kollisionsfrei ⇒ einweg:
Kontraposition: nicht einweg ⇒ es existiert ein effektiver Algorithmus A mit $A(z)=x$ sodass $h(x)=z$ ⇒ da für sehr viele $x_i \in \mathbb{B}^\infty$ $h(x_i)=h(x)$ gilt, wird für $x'=A(h(x))$ höchstwahrscheinlich $x' \neq x$ gelten ⇒ habe für ein geg. x ein $x' \neq x$ mit $h(x)=h(x')$ gefunden ⇒ nicht schwach kollisionsfrei ■



$A(h(x))=x' \neq x$ (Zeichng. aus VL)

(A) einweg ⇏ kollisionsfrei:
Sei $h: \mathbb{B}^\infty \to \mathbb{B}^n$ eine Einwegfunktion.
Baue $h': \mathbb{B}^\infty \to \mathbb{B}^n$, $(x_0, x_1, x_2, \dots) \mapsto h(x_0+x_1, x_2, \dots)$
h' ist immer noch einweg, aber nicht kollisionsfrei:
$h'(1, 0, x_2, \dots) = h'(0, 1, x_2, \dots) = h(1+0=0+1, x_2, \dots)$

UE 10: (B) schwach kollisionsfrei ⇏ kollisionsfrei:
Sei $h: \mathbb{B}^\infty \to \mathbb{B}^n$ schwach kollisionsfrei.
Wähle $x_0 \neq x_1 \in \mathbb{B}^\infty$ und $w \in \mathbb{B}^{n+1} \setminus \mathbb{B}^n$
Baue $h': \mathbb{B}^\infty \to \mathbb{B}^{n+1}$, $h'(x) := \begin{cases} w & \text{wenn } x=x_0 \text{ oder } x=x_1 \\ h(x) & \text{sonst} \end{cases}$
h' ist immer noch schwach kollisionsfrei, aber nicht kollisionsfrei:
$h'(x_0) = h'(x_1) = w$

(C) einweg ⇏ schwach kollisionsfrei:
Sei h eine Einwegfunktion.
Baue h' mit $h'(0,x) = h(x)$ und $h'(1,x) = h(x)$.
h' ist immer noch einweg, aber nicht schwach kollisionsfrei:
Sei $x \in_R \mathbb{B}^\infty$, $\text{len}(x) \geq 2$ ist dabei so gut wie sicher, $x' = \bar{x}$ ist dann ein zweites Urbild, d.h. $h'(x)=h'(\bar{x})$, wobei $\bar{x} := (1-x_0, x_1, x_2, \dots)$

- **Präzisere Definitionen (nach von zur Gathen, CryptoSchool):** (d.h. k=fest!)
  $h: \mathbb{B}^{n+k} \to \mathbb{B}^n$ mit $k>0$ ist eine Kompressionsfunktion der Länge n
  Ein effektiver Algorithmus A heißt...
  - „Inverter", wenn für ein $z \in_R \mathbb{B}^n$ $A(z) \in \{x, \text{FAIL}\}$ wobei $h(x)=z$.
  - „Zweiturbildfinder", wenn für ein $x \in_R \mathbb{B}^{n+k}$ $A(x) \in \{x', \text{FAIL}\}$ wobei $x' \neq x$ und $h(x')=h(x)$ (...hat es leichter als ein...)
  - „Kollisionsfinder", wenn für ein Input Φ $A(\Phi) \in \{(x,x'), \text{FAIL}\}$ wobei $h(x)=h(x')$

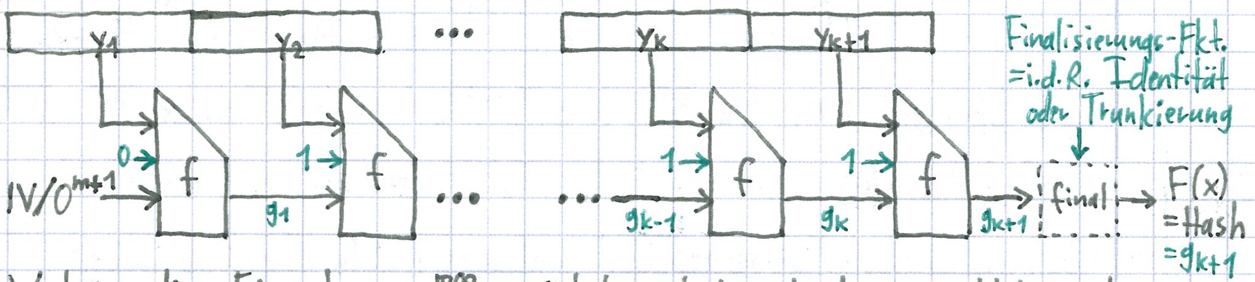
  Dabei ist σ_A die Erfolgswk. von A.
  zu (1): Satz 1: Exist. ein Zweiturbildfinder mit $\sigma_A = \varepsilon$, dann exist. ein Koll.finder $\mathcal{A}$ mit $\sigma_{\mathcal{A}} = \varepsilon$. (klar.)
  zu (2): Satz 2: Exist. ein Inverter mit $\sigma_A = \varepsilon$, dann exist. ein Zweiturbildfinder $\mathcal{A}$ mit $\sigma_{\mathcal{A}} \geq \sigma_A \cdot (1-2^{-k})$.

VL 11:

de.wiki: $\mathbb{B}^\infty = \{0,1\}^*$

(vgl. de.wiki ≟ VL)

• **Die Merkle-Damgård-Konstruktion (genutzt von MD5, SHA1, SHA2):**
... baut aus einer kollisionssicheren/-resistenten/-„freien" (↯ VL 4) **Kompressionsfunktion** $f: \mathbb{B}^{m+t} \to \mathbb{B}^m$, $t \geq 2$ eine kollisionssichere **Hash**funktion (d.h. beliebige Länge der Eingabe) $F: \mathbb{B}^\infty \to \mathbb{B}^m$



Wobei die Eingabe $x \in \mathbb{B}^\infty$ zunächst zerlegt und dann gepaddet wird:

Zerlegung: $x = x_1 \| x_2 \| \dots \| x_k$ mit $|x_i| = t-1$ für $i \in \{1, \dots, k-1\}$; evtl. ist x_k kürzer

Padding: $y_1 = x_1; \dots; y_{k-1} = x_{k-1}$; $y_k = x_k \| 0^d$ sodass $|y_k| = t-1$; y_{k+1} = Binärdarstellung von d, $|y_{k+1}| = t-1$
↳ auf de.wiki stattdessen $|m| = |x|$

Hash:
$g_1 = f(0^m \| 0 \| y_1)$
$g_2 = f(g_1 \| 1 \| y_2)$
...
$g_k = f(g_{k-1} \| 1 \| y_k)$
$g_{k+1} = f(g_k \| 1 \| y_{k+1}) = F(x) = \text{Hashwert}$

m	1	t-1
0^m	0	y_1
g_1	1	y_2
...	...	...
g_k	1	y_{k+1}

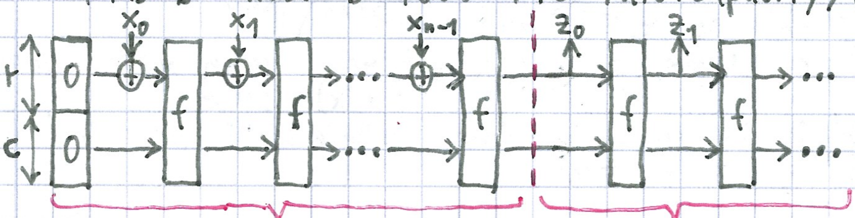
(f jeweils auf eine Zeile angewendet ergibt die nächste; $F(x) = g_{k+1}$)

Merkle-Damgård: Wenn man für F eine Kollision finden kann, dann auch für f.
D.h.: f ist kollisionssicher ⇒ F ist kollisionssicher

Schwäche dieser Konstruktion: "Length extension" (VL) / "Extension-Attack" (de.wiki):
Wenn die Finalisierungsfunktion umkehrbar ist, kann man aus dem Hash F(m) einer unbekannten Nachricht m leicht den Hash $F(\bar{m} \| y)$ für beliebige Erweiterungen y bestimmen (wobei $\bar{m}$ die gepaddete Version von m ist), da man den inneren Zustand von Merkle-Damgård kennt!
⇒ Hat man erstmal eine Kollision gefunden, hat man gleich Mehrfachkollisionen für den gleichen Hash!

SHA-3 verhindert dies, da internal state aus c zusätzl. Bits/Infos besteht, die nicht Teil des Outputs sind (vgl. wiki/UE):

• **SHA-3 (unter dem Namen „Keccak" entwickelt):** → u.a. John Daemen von AES!
Status: $\mathbb{B}^b$ wobei $b = 1600 = r + c$ = rate + capacity; (r,c) variabel; c höher = sicherer!

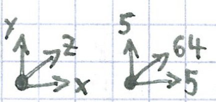


Schwamm-/Sponge-Konstruktion (VL)

→ genannt „Permutationsfunktion"

L=6 bei 64bit-Wörtern → in $12 + 2 \cdot L = 12 + 2 \cdot 6 = 24$ Runden! (siehe auch VL)

wobei f = Hintereinanderschaltung von Θ (theta), ρ (rho), π (pi), χ (chi), ι (iota)
dabei: $\mathbb{B}^{1600} \cong W^{5 \times 5}$ mit $W = \mathbb{B}^{64}$ (d.h. 64-bit-Wörter), sehe Status als 5×5×64-Array



de.wiki / VL: $x \lll 1 = rot(x, 1)$

Lineare Mischoperation:
→ Θ: Paritätsbits jeder 5-Wort-Spalte mit den Wörtern der 2 benachbarten Spalten XORen:
$p_j := a_{0j} \oplus a_{1j} \oplus a_{2j} \oplus a_{3j} \oplus a_{4j}$ $\bar{a}_{ij} := a_{ij} \oplus p_{j-1} \oplus (p_{j+1} \lll 1)$ (de.wiki)

Achtung: x,y/i,j in de.wiki & VL vertauscht; hier wie im de.wiki!

→ ρ: Wörter des Zustandsvektors rotieren: $\bar{a}_{ij} := a_{ij} \lll r[i,j]$ (de.wiki/VL)

→ π: Wörter des Zustandsvektors permutieren: $\bar{a}_{3i+2j,i} := a_{i,j}$ (de.wiki)

→ χ: nichtlineare Operation: $\bar{a}_{i,j} := a_{i,j} \oplus (\neg a_{i,j+1} \,\&\, a_{i,j+2})$ (de.wiki)
bitweise Negation ↑ ↑ bitweises Und (in VL: "·" für Und)

→ ι: XOR-Verknüpfen des Wortes $a_{0,0}$ mit einer rundenabhängigen Konstanten:
$\bar{a}_{0,0} := a_{0,0} \oplus RC[r]$ (de.wiki/VL)

Dabei sind r[x,y] & RC[r] per Tabelle gegeben, aber auch berechenbar!