Floyd's Algorithmus

Speicherbedarf sehr groß. Können auch noch besser vorgehen: Floyd's Algorithmus zum Finden von Zykeln.

$h : X \to Z$ Hashfunktion.
Annahme $Z \subseteq X$, $x_0 \in X \setminus Z$.

Definiere $x_{i+1} = h(x_i)$.
Es gibt minimale $\alpha < \beta$ mit $x_\alpha = x_\beta$ und $x_{\alpha-1} \neq x_{\beta-1}$.
Dann auch $x_{\alpha+i} = x_{\beta+i}$ für alle i, damit Zykellänge $\delta = \beta - \alpha$.
Also $x_{i+\delta} = x_i$ für $i \geq \alpha$ und $x_i = x_{2i} \Leftrightarrow 2i = i + r\delta \geq 2\alpha \Leftrightarrow i = r\delta \geq \alpha$.
Für $x_{\alpha-1}$ gilt $x_{\alpha-1} \neq x_{\alpha-1+r\delta}$ und $x_\alpha = x_{\alpha+r\delta}$.

Floyd's Algorithmus

Daraus ergibt sich folgendes Vorgehen:

- Berechne $(x_1, x_2), (x_2, x_4), \ldots$ bis $x_i = x_{2i}$. Setze $\delta' \leftarrow i$.
- Berechne $(x_0, x_{\delta'}), (x_1, x_{\delta'+1}), \ldots$ bis $x_i = x_{i+\delta'}$.
- Dann $x_{i-1} \neq x_{i+\delta'-1}$ und $x_i = x_{i+\delta'}$, also $h(x_{i-1}) = h(x_{i+\delta'-1})$.

Sei $n = \#Z$.

Im Zufallsorakelmodell erfolgt eine Kollision nach $1.18\sqrt{n}$ vielen Hashwerten x_i mit Wahrscheinlichkeit $\geq 1/2$.

Folglich $\alpha, \delta \leq 1.18\sqrt{n}$. Außerdem $\delta' \leq \max\{\delta, 2\alpha\}$ wegen der Minimalität von $\delta' = r\delta \geq \alpha$. Daher ergeben sich ungefähr $3.54\sqrt{n}$ Orakelaufrufe und konstanter Speicherbedarf.

Folgerung

Sei $h : X \to Z$ eine Hashfunktion mit $\#Z = n = 2^k$.

Im Zufallsorakelmodell kann man also eine Kollision nach ca. $s = \sqrt{n}$ Anfragen an das Hashorakel mit guter Wahrscheinlichkeit finden.

Auf der anderen Seite ist diese Wahrscheinlichkeit durch s^2/n nach oben beschränkt. Benutzt man weniger Orakelanfragen, nimmt die Wahrscheinlichkeit einer Kollision zügig (quadratisch) ab.

Ein in k polynomieller Angreifer kann folglich auf eine Kollision nur mit vernachlässigbarer Wahrscheinlichkeit poly$(k)/2^k$ hoffen. Seine Erfolgswahrscheinlichkeit ist damit insgesamt ebenfalls vernachlässigbar.

Folgerung

Damit sind Hashfunktionen im Zufallsorakelmodell auch kollisionsresistent, wenn gleich der Aufwand zum Finden einer Kollision wesentlich geringer ist als der, Urbilder zu berechnen.

Im Zufallsorakelmodell ergibt sich zusammenfassend:

- k Bit Sicherheit bezüglich der Einweg-Eigenschaft.
- Nur $k/2$ Bit Sicherheit bezüglich Kollisionsresistenz.

Von einer „guten" Hashfunktion fordert man daher im Standardmodell (d.h. nicht im Zufallsorakelmodell), daß Urbilder und Kollisionen nur mit Aufwand ungefähr 2^k bzw. $2^{k/2}$ berechnet werden können sollen.

In der Praxis fordert man zur Zeit $k \geq 160$.

Vergleich der Sicherheitseigenschaften

Die folgenden zwei Reduktionen sind im Standardmodell gültig.

Sei $h : X \to Z$ eine Hashfunktion.

Können wir zweite Urbilder berechnen, so können wir Kollisionen berechnen:

- Wähle $x \in X$ zufällig.
- Berechne ein zweites Urbild $x' \neq x$ mit $h(x') = h(x)$.
- Ausgabe von x, x'.

Resistenz gegen Kollisionen impliziert also Resistenz gegen schwache Kollisionen.

Vergleich der Sicherheitseigenschaften

Sei $h : X \to Y$ eine Kompressionsfunktion mit $\#X \geq 2\#Y$.

Können wir Urbilder berechnen, so können wir Kollisionen berechnen:

- Wähle $x \in X$ zufällig.
- Berechne ein Urbild x' von $h(x)$.
- Ausgabe von x, x', wenn $x \neq x'$. Sonst Fehler.

Sei $C = \{h^{-1}(\{h(x)\}) \mid x \in X\}$. Die Erfolgswahrscheinlichkeit ist

$$\begin{aligned}
P &= (1/\#X) \sum_{x \in X} (\#h^{-1}(\{h(x)\}) - 1)/\#h^{-1}(\{h(x)\}) \\
&= (1/\#X) \sum_{c \in C} \sum_{x \in c} (\#c - 1)/\#c = (1/\#X) \sum_{c \in C} (\#c - 1) \\
&\geq (\#X - \#Y)/\#X \geq (\#X - \#X/2)/\#X \geq 1/2.
\end{aligned}$$

Resistenz gegen Kollisionen impliziert also die Einweg-Eigenschaft.

Konstruktion von Hashfunktionen

Neben Einwegeigenschaft und Kollisionsresistenz sollen Hashwerte von langen Nachrichten effizient ohne großen Speicheraufwand berechnet werden können (Magnetband, ...).

Allgemeines Prinzip: Iterierung.

Nachricht mit Bitstring 0^d oder 10^{d-1} und Nachrichtenlänge padden, dann in geeignete Blöcke m_i aufteilen.

Bei der Berechnung des Hashwerts eine Zustandsvariable h_i mitführen. Erster Zustand ist $h_0 = IV$, letzter Zustand h_n ist Hashwert.

In jedem Schritt eine Kompressionsfunktion auf m_i und h_i anwenden, liefert h_{i+1}.

Ähnlich wie im CBC Mode für Blockchiffren.

Davies-Meyer Kompressionsfunktion

Man benutzt einen Blockchiffre, um eine Kompressionsfunktion zu erhalten.

Blockchiffre $\mathcal{E} : \{0,1\}^k \times \{0,1\}^b \to \{0,1\}^b$ mit Schlüssellänge k und Blocklänge b.

Liefert:
Kompressionsfunktion $f : \{0,1\}^{k+b} \to \{0,1\}^b$ mit $f(K||m) = \mathcal{E}(K,m) \oplus m$.

Ist nicht sehr effizient ($\mathcal{E}$ bietet zuviel Funktionalität, z.B. $\mathcal{D}$).
Man verwendet daher meist andere Kompressionsfunktionen.

Ansonsten gibt es noch andere Varianten. Man kann auch die Blocklänge verdoppeln etc.

Merkle-Damgard Konstruktion

Hashfunktion $h : \{0,1\}^* \to \{0,1\}^n$ aus Kompressionsfunktion bauen.

Sei $f : \{0,1\}^m \to \{0,1\}^n$ eine Kompressionsfunktion und $r = m - n \geq 2$.

Der Hashwert von $x \in \{0,1\}^*$ wird dann wie folgt ausgerechnet:

- x hinten mit Nullen und der Bitlänge von x binär geschrieben padden und in s Blöcke $x_i \in \{0,1\}^r$ mit $0 \leq i \leq s-1$ aufteilen.
- $h_0 = IV$ für einen festen Initialwert.
- $h_{i+1} = f(h_i || x_i)$ für $0 \leq i \leq s-1$.
- Der Hashwert ist h_s.

Sollte das Padding mehrere Blöcke beanspruchen, so fügen wir ein weiteres Padding an, welches die Anzahl der im vorhergehenden Padding verwendeten Blöcke enthält usw., bis das letzte Padding in einen Block paßt. Es existieren hier auch andere Varianten.

Merkle-Damgard Konstruktion

Thm: Für eine kollisionsresistente Kompressionsfunktion f ist auch die durch die Merkle-Damgard erhaltene Hashfunktion h kollisionsresistent.

Bew: Sei $h(x) = h(x')$ mit $x \neq x'$. Definiere $x_i, x'_i, s, s', h_i, h'_i$ wie in der Konstruktion. Wir können $s \leq s'$ annehmen. Es gilt $h_s = h'_{s'}$. Gilt $(h_{s-j}, x_{s-j}) = (h'_{s'-j}, x'_{s'-j})$ für alle $1 \leq j \leq s$, so folgt $s = s'$ wegen dem Padding und dann $x = x'$ im Widerspruch zur Annahme. Sei also j minimal mit $1 \leq j \leq s$ und $(h_{s-j}, x_{s-j}) \neq (h'_{s'-j}, x'_{s'-j})$. Dann gilt $h_{s-j+1} = h'_{s'-j+1}$ und eine Kollision der Kompressionsfunktion ist gefunden. $\square$

Effiziente Hashfunktionen

MD5 (Message Digest 5):

- Von Ron Rivest, 1992. Recht weit verbreitet.
- 128 Bit Hashwerte, ist für heute etwas knapp bemessen.
- Wurde kürzlich gebrochen, Kollisionen können gefunden werden, daher unsicher!

RIPEMD-160:

- Europäisches Design (K. U. Leven und BSI), 1996.
- 160 Bit Hashwerte (interne Blockgröße 512 Bit).
- Im ISO/IEC 10118-3 standardisiert.
- Ebenfalls unsicher.

Effiziente Hashfunktionen

SHA-1 (Secure Hash Algorithm):

- Von der NSA.
- 160 Bit Hashwerte (interne Blockgröße 512 Bit).
- Am weitesten verbreitete Hashfunktion.
- Im ISO/IEC 10118-3 und FIPS180-1 standardisiert.
- Sicherheit angeknackst, liegt bei ca. 2^{63} anstelle von 2^{80}.

SHA-256, SHA-384, SHA-512:

- Im FIPS180-2.
- 256, 384 und 512 Bit Hashwerte.
- Recht neu (2001).
- Scheinen zur Zeit die einzig sicheren (effizienten) Hashfunktionen zu sein ...

SHA-1

Eingabe x muß Bitlänge $|x| \leq 2^{64} - 1$ haben.

SHA-1 Padding:

- Eingabe x. Setze $d \leftarrow (447 - |x|) \bmod 512$.
- $l \leftarrow$ Binärdarstellung von $|x|$, wobei $|l| = 64$.
- $y \leftarrow x||1||0^d||l$. Ausgabe y.

$\vee$ logisches Oder, $\wedge$ logisches Und, $\bar{\cdot}$ Negation.

80 Funktionen:

$$f_i(B,C,D) = \begin{cases} (B \wedge C) \vee (\bar{B} \wedge D) & \text{für } 0 \leq i \leq 19 \\ B \oplus C \oplus D & \text{für } 20 \leq i \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{für } 40 \leq i \leq 59 \\ B \oplus C \oplus D & \text{für } 60 \leq i \leq 79. \end{cases}$$

SHA-1

80 Konstanten:

$$K_i = \begin{cases} \texttt{5A827999} & \text{für } 0 \leq i \leq 19 \\ \texttt{6ED9EBA1} & \text{für } 20 \leq i \leq 39 \\ \texttt{8F1BBCDC} & \text{für } 40 \leq i \leq 59 \\ \texttt{CA62C1D6} & \text{für } 60 \leq i \leq 79. \end{cases}$$

ROTL^x = zyklischer Shift um x Bits nach Links. + Addition modulo 2^{32}.

Kompressionsfunktion $f : \{0,1\}^{512} \times \{0,1\}^{160} \to \{0,1\}^{160}$:

- Eingabe $M \in \{0,1\}^{512}$, $H \in \{0,1\}^{160}$.
- Schreibe $M = W_0||\ldots||W_{15}$ mit $W_i \in \{0,1\}^{32}$.
- Schreibe $H = H_0||\ldots||H_4$ mit $H_i \in \{0,1\}^{32}$.
- Für $t \leftarrow 16,\ldots,79$: $W_t \leftarrow$ROTL$^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$.
- $A \leftarrow H_0, \ldots, E \leftarrow H_4$.

SHA-1

- Für $t \leftarrow 0,\ldots,79$:

  $y \leftarrow$ROTL$^5(A) + f_t(B,C,D) + E + W_t + K_t$.

  $E \leftarrow D$, $D \leftarrow C$, $C \leftarrow$ROTL$^{30}(B)$.

  $B \leftarrow A$, $A \leftarrow y$.
- $H_0 \leftarrow H_0 + A, \ldots, H_4 \leftarrow H_4 + E$.
- Ausgabe $H_0||\ldots||H_4$.

SHA-1:

- Eingabe $x \in \{0,1\}^*$.
- $x \leftarrow$ SHA-1 Padding(x).
- Schreibe $x = M_1||\ldots||M_n$ mit $M_i \in \{0,1\}^{512}$.
- $H \leftarrow$ `67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0`.
- Für $i \leftarrow 1,\ldots,n$: $H \leftarrow f(M_i, H)$.
- Ausgabe von H.