

Kryptologie

Johannes Köbler



Institut für Informatik
Humboldt-Universität zu Berlin

WS 2020/21

Aktuelle Infos auf der VL-Webseite unter

- <https://hu.berlin/vlkrypto>

bzw.

- <https://www.informatik.hu-berlin.de/de/forschung/gebiete/algorithmenII/Lehre/ws20/krypto>

Skript, Folien und Aufgabenblätter

- Skript, Folien und Aufzeichnung werden jeweils nach der Vorlesung ins Netz (Webseite bzw. Moodle) gestellt
- Übungsblätter werden in der Regel dienstags veröffentlicht
- Die Besprechung der mündlichen Aufgaben erfolgt am Freitag der Folgewoche. Lösungen dazu können bis zum Tag davor in Moodle hochgeladen werden, Details siehe dort
- Die schriftlichen Aufgaben sind bis Dienstag zwei Wochen nach Ausgabe um 23:59 Uhr abzugeben
- Fragen zu Übung und Vorlesung können im Moodle-Forum auch **asynchron** gestellt und diskutiert werden

Anmeldung

- über Agnes
- und bei Moodle (wegen Punktevergabe und Bildung von Abgabegruppen)
- Mails von Agnes und von Moodle werden standardmäßig an den HU-Account gesendet (bitte regelmäßig checken)

Ausgabe der Aufgabenblätter

- über Moodle und auf der VL-Webseite

Abgabe von Lösungen

- digital über Moodle

- in Gruppen von bis zu drei Teilnehmern
- Lösungen für die schriftlichen Aufgaben sollten als PDF abgegeben werden
- die Abgabe von Lösungsvorschlägen für die mündlichen Aufgaben ist freiwillig und geht nicht in die Punktwertung ein
- Lösungsvorschläge für die mündlichen Aufgaben können auch per Texteingabe gemacht werden
- besonders gut gelungene Lösungen werden mit Zustimmung der/des Abgebenden im Forum veröffentlicht

Scheinkriterien

- Lösen von mindestens 50% der schriftlichen Aufgaben

Prüfungsform

- voraussichtlich mündlich
- Der Übungsschein ist **nicht** Prüfungsvoraussetzung

Gibt es zum organisatorischen Ablauf noch Fragen?

- Kryptografische Verfahren schaffen Vertrauen in ungeschützten Umgebungen
- Sie ermöglichen sichere Kommunikation über unsichere Kanäle und können verhindern, dass sich ein Kommunikationspartner unfair verhält
- In unsicheren Umgebungen wie dem Internet können sie die aus direkter Interaktion gewohnte Sicherheit herstellen
- Und auch die Interaktion in sicheren Umgebungen wird um Möglichkeiten erweitert, die ohne Kryptografie nicht denkbar wären
- Im Bachelormodul **Einführung in die Kryptologie** haben wir uns mit den mathematischen Grundlagen von kryptografischen Verfahren beschäftigt, wobei (symmetrische und asymmetrische) Verschlüsselungsverfahren im Vordergrund standen
- Im aktuellen Mastermodul **Kryptologie** werden wir dagegen kryptografische Verfahren und Protokolle für andere Schutzziele betrachten wie z.B. Hashverfahren und digitale Signaturen sowie Pseudozufallsgeneratoren

- Kryptosysteme (Verschlüsselungsverfahren) dienen der Geheimhaltung von Nachrichten bzw. Daten
- Hierzu gibt es auch andere Methoden wie z.B.
 - Physikalische Maßnahmen: Tresor etc.
 - Organisatorische Maßnahmen: einsamer Waldspaziergang etc.
 - Steganografische Maßnahmen: unsichtbare Tinte etc.

Überblick weiterer Schutzziele

Andererseits können durch kryptografische Verfahren weitere **Schutzziele** realisiert werden wie z.B.

- **Vertraulichkeit**
 - Geheimhaltung
 - Anonymität (z.B. Mobiltelefon)
 - Unbeobachtbarkeit (von Transaktionen)
- **Integrität**
 - von Nachrichten und Daten
- **Zurechenbarkeit**
 - Authentikation
 - Unabstreitbarkeit
 - Identifizierung
- **Verfügbarkeit**
 - von Daten
 - von Rechenressourcen
 - von Informationsdienstleistungen

In das Umfeld der Kryptologie fallen die folgenden Begriffe

- **Kryptografie:**

Lehre von der Geheimhaltung von Informationen durch Verschlüsselung
Im weiteren Sinne: Wissenschaft von der Übermittlung, Speicherung und Verarbeitung von Daten in einer von potentiellen Gegnern bedrohten Umgebung

- **Kryptoanalysis:**

Erforschung der Methoden eines unbefugten Angriffs gegen ein Kryptoverfahren
Zweck: Vereitelung der mit seinem Einsatz verfolgten Ziele

- **Kryptoanalyse:**

Analyse eines Kryptoverfahrens zum Zweck der Bewertung seiner kryptografischen Stärken und Schwächen

- **Kryptologie:**

Wissenschaft vom Entwurf, der Anwendung und der Analyse von kryptografischen Verfahren (umfasst Kryptografie und Kryptoanalyse)

- sind ein wirksames Werkzeug zur Sicherstellung der Integrität von Nachrichten oder generell von digitalisierten Daten
- Sie nehmen somit beim Schutz der Datenintegrität eine ähnlich herausragende Stellung ein wie sie Kryptosystemen bei der Wahrung der Vertraulichkeit zukommt
- Daneben finden kryptografische Hashfunktionen aber auch vielfach als Bausteine von komplexeren Systemen Verwendung
- Wie wir noch sehen werden, sind kryptografische Hashfunktionen etwa bei der Erstellung von digitalen Signaturen sehr nützlich
- Auf weitere Anwendungsmöglichkeiten werden wir später eingehen

- Vielen Anwendungen von kryptografischen Hashfunktionen h liegt die Idee zugrunde, dass sie zu einem vorgegebenen Text x eine zwar kompakte aber dennoch repräsentative Darstellung $h(x)$ liefern, die unter praktischen Gesichtspunkten als eine eindeutige Identifikationsnummer von x fungieren kann
- Die Berechnungsvorschrift für h muss somit „charakteristische Merkmale“ von x in den Hashwert $h(x)$ einfließen lassen
- Da der Fingerabdruck eines Menschen ganz ähnliche Eigenschaften besitzt (was ihn für Kriminalisten bekanntlich so wertvoll macht), wird der Hashwert $h(x)$ auch oft als ein **digitaler Fingerabdruck** von x bezeichnet
- Gebräuchlich sind auch die Bezeichnungen **kryptografische Prüfsumme** oder **message digest** (englische Bezeichnung für „Nachrichtenextrakt“)

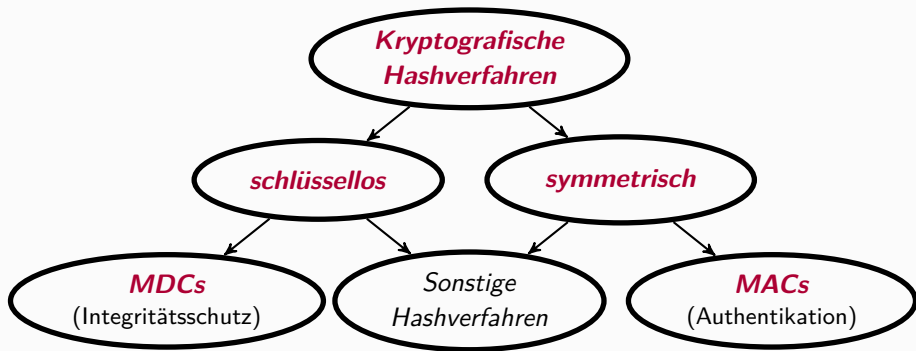
Typische Schutzziele, die sich mittels Hashfunktionen realisieren lassen:

Nachrichtenauthentikation (message authentication)

- Wie lässt sich sicherstellen, dass eine Nachricht (oder eine Datei) während einer (räumlichen oder auch zeitlichen) Übertragung nicht verändert wurde?
- Wie lässt sich der Urheber (oder Absender) einer Nachricht zweifelsfrei feststellen?

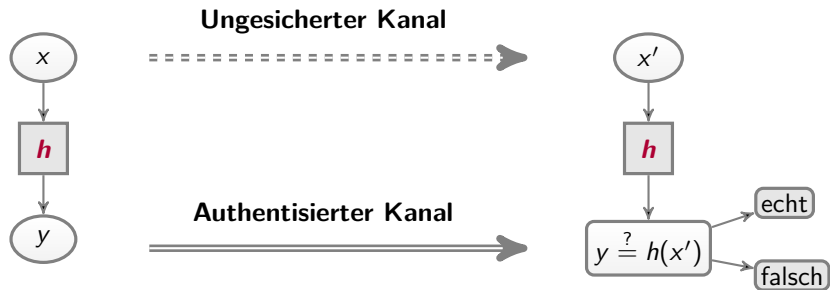
Teilnehmerauthentikation (entity authentication, identification)

- Wie kann sich eine Person (oder ein Gerät) anderen gegenüber zweifelsfrei ausweisen?



Kryptografische Hashverfahren lassen sich grob danach klassifizieren, ob der Hashwert lediglich in Abhängigkeit vom Eingabetext berechnet wird oder zusätzlich von einem symmetrischen Schlüssel abhängt

- Kryptografische Hashfunktionen, bei deren Berechnung keine Schlüssel benutzt werden, dienen vornehmlich der Erkennung von unbefugt vorgenommenen Manipulationen an Dateien oder Nachrichten
- Daher werden sie auch als **MDC** (*M*anipulation *D*etection *C*ode) bezeichnet
- Zuweilen wird das Kürzel **MDC** auch als eine Abkürzung für *M*odification *D*etection *C*ode verwendet
- Seltener ist dagegen die Bezeichnung **MIC** (*m*essage *i*ntegrity *c*odes)

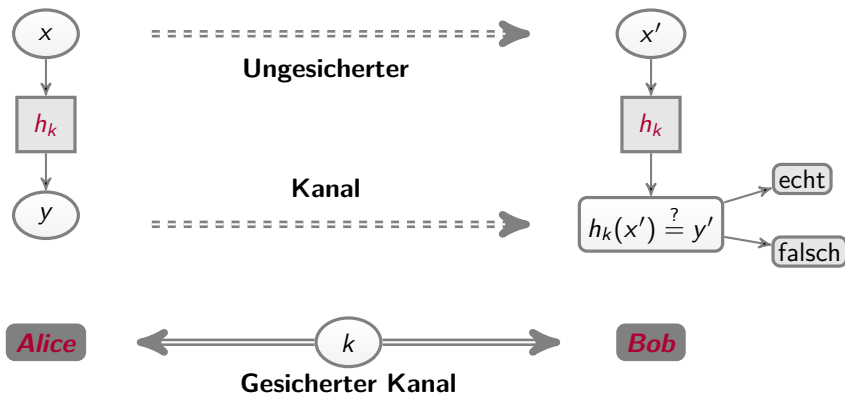


Um die Integrität eines Datensatzes x sicherzustellen, der über einen ungesicherten Kanal gesendet (bzw. auf einem vor Manipulationen nicht sicheren Webserver abgelegt) wird, kann man wie folgt vorgehen

- Der **MDC**-Hashwert $y = h(x)$ von x wird auf einem authentisierten Kanal übertragen
- Nach der Übertragung wird geprüft, ob der Datensatz noch den Hashwert y liefert

- Kryptografische Hashverfahren mit symmetrischen Schlüsseln finden hauptsächlich bei der Authentifizierung von Nachrichten Verwendung
- Diese werden daher auch als **MAC** (*message authentication code*) oder als **Authentikationscode** bezeichnet
- Daneben gibt es auch Hashverfahren mit asymmetrischen Schlüsseln
- Diese werden jedoch der Rubrik der Signaturverfahren zugeordnet, da mit ihnen ausschließlich digitale Signaturen gebildet werden

- Die Abbildung auf der nächsten Folie zeigt, wie sich Nachrichten mit einem MAC authentisieren lassen
- Man beachte, dass nun auch der Hashwert über den unsicheren Kanal gesendet wird
- Möchte Alice eine Nachricht x an Bob übermitteln, so berechnet sie den zugehörigen **MAC**-Wert $y = h_k(x)$ und fügt diesen der Nachricht x hinzu
- Bob überprüft die Echtheit der empfangenen Nachricht (x', y') , indem er seinerseits den zu x' gehörigen Hashwert $h_k(x')$ berechnet und das Ergebnis mit y' vergleicht
- Der geheime Authentifikationsschlüssel k muss hierbei genau wie bei einem symmetrischen Kryptosystem über einen gesicherten Kanal vereinbart werden



- Hierbei ist k der symmetrische Authentifikationsschlüssel und $y = h_k(x)$ der **MAC**-Wert für x unter k
- Indem Alice ihre Nachricht x um den Hashwert $y = h_k(x)$ ergänzt, hat Bob nicht nur die Möglichkeit, anhand von y die empfangene Nachricht x' auf Manipulationen, sondern auch ihre Herkunft zu überprüfen

- Wir betrachten nun verschiedene Sicherheitsanforderungen an MDCs h
- Dabei nehmen wir an, dass $h: X \rightarrow Y$ öffentlich bekannt ist
- Ein Paar $(x, y) \in X \times Y$ heißt **gültig** für h , falls $h(x) = y$ ist
- Ein Paar (x, x') mit $x \neq x'$ und $h(x) = h(x')$ heißt **Kollisionspaar** für h
- Die Anzahl $\|Y\|$ der Hashwerte bezeichnen wir mit m
- Ist auch der Textraum X endlich, $\|X\| = n$, so heißt h eine **(n, m) -Hashfunktion**
- In diesem Fall verlangen wir meist, dass $n \geq 2m$ ist, und wir nennen h dann eine **Kompressionsfunktion** (compression function)

Schlüssellose Hashfunktionen (MDCs)

- Da h öffentlich bekannt ist, ist es sehr einfach, für einen vorgegebenen Text x ein gültiges Paar (x, y) zu erzeugen
- Für bestimmte kryptografische Anwendungen ist es wichtig, dass dies bei vorgegebenem Hashwert y dagegen nicht möglich ist

Problem P1 (Bestimmung eines Urbilds)

Gegeben: Eine Hashfunktion $h: X \rightarrow Y$ und ein Hashwert $y \in Y$

Gesucht: Ein Text $x \in X$ mit $h(x) = y$

- Falls es einen immensen Aufwand erfordert, bei gegebenem Hashwert y einen Text x mit $h(x) = y$ zu finden, so heißt h **Einweg-Hashfunktion** (*one-way hash function bzw. preimage resistant hash function*)
- Diese Eigenschaft wird beispielsweise benötigt, wenn die Hashwerte der Benutzerpasswörter in einer öffentlich zugänglichen Datei abgespeichert werden, wie es bei manchen Unix-Systemen der Fall ist

- Für andere Anwendungen ist es dagegen wichtig, dass es für einen gegebenen Text x praktisch unmöglich ist, einen weiteren Text $x' \neq x$ mit dem gleichen Hashwert $h(x') = h(x)$ zu finden

Problem P2 (Bestimmung eines zweiten Urbilds)

Gegeben: Eine Hashfunktion $h: X \rightarrow Y$ und ein Text $x \in X$

Gesucht: Ein Text $x' \in X \setminus \{x\}$ mit $h(x') = h(x)$

- Falls Problem P2 einen immensen Aufwand erfordert, heißt h **schwach kollisionsresistent** (*weakly collision resistant bzw. second preimage resistant*)
- Diese Eigenschaft wird beim Integritätsschutz durch einen MDC benötigt

- Für bestimmte Anwendungen ist es sogar nötig, dass sich überhaupt kein Kollisionspaar finden lässt
- Diese Eigenschaft ist bspw. beim Einsatz von MDCs bei der Erstellung von digitalen Signaturen erforderlich

Problem P3 (Bestimmung einer Kollision)

Gegeben: Eine Hashfunktion $h: X \rightarrow Y$

Gesucht: Zwei Texte $x \neq x' \in X$ mit $h(x') = h(x)$

- Falls Problem P3 einen immensen Aufwand erfordert, heißt h (**stark**) **kollisionsresistent** (*collision resistant*)

- Falls Problem P3 einen immensen Aufwand erfordert, heißt h (**stark**) **kollisionsresistent** (*collision resistant*)
- Obwohl die schwache Kollisionsresistenz eine gewisse Ähnlichkeit mit der Einweg-Eigenschaft aufweist, sind diese beiden Eigenschaften im allgemeinen unvergleichbar:
 - Eine schwach kollisionsresistente Funktion muss nicht notwendigerweise eine Einwegfunktion sein, da die Bestimmung eines Urbildes gerade für diejenigen Funktionswerte einfach sein kann, die nur ein einziges Urbild besitzen
 - Umgekehrt impliziert die Einweg-Eigenschaft auch nicht die schwache Kollisionsresistenz, da die Kenntnis eines Urbildes das Auffinden weiterer Urbilder sehr stark erleichtern kann

- Wir zeigen nun, dass stark kollisionsresistente Hashfunktionen sowohl schwach kollisionsresistent als auch Einweghashfunktionen sind
- Hierzu reduzieren wir das Kollisionsproblem auf das Problem, ein zweites Urbild zu bestimmen

Satz

- Sei $h: X \rightarrow Y$ eine (n, m) -Hashfunktion
- Dann ist das Problem P3, ein Kollisionspaar für h zu bestimmen, auf das Problem P2, ein zweites Urbild zu bestimmen, reduzierbar
- Folglich sind stark kollisionsresistente Hashfunktionen auch schwach kollisionsresistent

Satz

- Sei $h: X \rightarrow Y$ eine (n, m) -Hashfunktion
- Dann ist das Problem P3, ein Kollisionspaar für h zu bestimmen, auf das Problem P2, ein zweites Urbild zu bestimmen, reduzierbar
- Folglich sind stark kollisionsresistente Hashfunktionen auch schwach kollisionsresistent

Beweis.

- Sei A ein Las-Vegas Algorithmus, der für ein zufällig aus X gewähltes x mit Erfolgswahrscheinlichkeit ε ein zweites Urbild x' für h liefert und andernfalls ? ausgibt
- Dann ist klar, dass folgender Las-Vegas Algorithmus mit Wahrscheinlichkeit ε ein Kollisionspaar findet:

-
- 1 wähle zufällig $x \in X$
 - 2 $x' := A(x)$
 - 3 **if** $x' \neq ?$ **then return** (x, x') **else return** ?
-

Als nächstes reduzieren wir das Kollisionsproblem auf das Urbildproblem

Satz

- Sei $h: X \rightarrow Y$ eine (n, m) -Hashfunktion mit $n \geq 2m$
- Dann ist das Problem P3, ein Kollisionspaar für h zu bestimmen, auf das Problem P1, ein Urbild zu bestimmen, reduzierbar

Beweis.

- Sei A ein Invertierungsalgorithmus für h , d.h. A berechnet für jeden Hashwert y in $W(h) = \{h(x) \mid x \in X\}$ ein Urbild x mit $h(x) = y$
- Betrachte folgenden Las-Vegas Algorithmus B:

 - 1 wähle zufällig $x \in X$
 - 2 $y := h(x)$
 - 3 $x' := A(y)$
 - 4 **if** $x \neq x'$ **then return** (x, x') **else return** ?

Beweis.

- Sei A ein Invertierungsalgorithmus für h , d.h. A berechnet für jeden Hashwert y in $W(h) = \{h(x) \mid x \in X\}$ ein Urbild x mit $h(x) = y$
- Betrachte folgenden Las-Vegas Algorithmus B :

```
1 wähle zufällig  $x \in X$ 
2  $y := h(x)$ 
3  $x' := A(y)$ 
4 if  $x \neq x'$  then return  $(x, x')$  else return ?
```

- Sei $C = \{h^{-1}(y) \mid y \in W(X)\}$
- Dann hat B eine Erfolgswahrscheinlichkeit von

$$\sum_{C \in \mathcal{C}} \frac{\|C\|}{\|X\|} \cdot \frac{\|C\| - 1}{\|C\|} = \frac{1}{n} \sum_{C \in \mathcal{C}} (\|C\| - 1) = (n - m)/n \geq \frac{1}{2}$$

Das Zufallsorakelmodell (ZOM)

- Das ZOM dient dazu, den Aufwand verschiedener Angriffe auf eine Hashfunktion $h: X \rightarrow Y$ nach oben abzuschätzen
- Sind X und Y vorgegeben, so können wir eine Hashfunktion $h: X \rightarrow Y$ dadurch „konstruieren“, dass wir für jedes $x \in X$ zufällig ein $y \in Y$ wählen und $h(x) = y$ setzen
- Äquivalent hierzu ist, für h eine zufällige Funktion aus der Klasse $F(X, Y)$ aller m^n Funktionen von X nach Y zu wählen
- Dieses Verfahren ist auf Grund des hohen Aufwands zwar nicht mehr praktikabel, wenn $n = \|X\|$ eine bestimmte Größe übersteigt
- Es liefert uns aber ein theoretisches Modell für eine Hashfunktion mit „idealen“ kryptografischen Eigenschaften
- Offensichtlich kann ein Angreifer nur dadurch Informationen über h erhalten, dass er für eine Reihe von Texten x_i die zugehörigen Hashwerte $h(x_i)$ berechnet (was der Befragung eines funktionalen Zufallsorakels entspricht)

Eine Zufallsfunktion h eignet sich deshalb gut als kryptografische Hashfunktion, weil der Hashwert $h(x)$ für einen Text x auch dann noch schwer vorhersagbar ist, wenn der Angreifer bereits die Hashwerte einer beliebigen Zahl von anderen Texten $x_i \neq x$ kennt

Proposition

- Sei $X_0 = \{x_1, \dots, x_k\}$ eine beliebige Menge von k verschiedenen Texten $x_i \in X$ und seien $y_1, \dots, y_k \in Y$
- Dann gilt für eine zufällig aus $F(X, Y)$ gewählte Funktion h und für jedes Paar $(x, y) \in (X - X_0) \times Y$,

$$\Pr[h(x) = y \mid h(x_i) = y_i \text{ für } i = 1, \dots, k] = 1/m$$

- Um eine obere Komplexitätsschranke für das Urbildproblem P1 im ZOM zu erhalten, betrachten wir folgenden Algorithmus
- Hierbei gibt der Parameter q die Anzahl der Hashwertberechnungen (also die Anzahl der gestellten Orakelfragen an das Zufallsorakel h) an
- Die Laufzeit des Algorithmus ist also proportional zu q

Prozedur FindPreimage(h, y, q)

- 1 wähle eine beliebige Menge $X_0 = \{x_1, \dots, x_q\} \subseteq X$
 - 2 **for** each $x_i \in X_0$ **do**
 - 3 **if** $h(x_i) = y$ **then return**(x_i)
 - 4 **return** ?
-

Satz

FINDPREIMAGE(h, y, q) gibt im ZOM mit Wahrscheinlichkeit $\varepsilon = 1 - (1 - 1/m)^q$ ein Urbild von y aus (unabhängig von der Wahl der Menge X_0)

Beweis.

- Sei $y \in Y$ fest und sei $X_0 = \{x_1, \dots, x_q\}$
- Für $i = 1, \dots, q$ bezeichne E_i das Ereignis " $h(x_i) = y$ "
- Nach obiger Proposition sind diese Ereignisse stochastisch unabhängig und ihre Wahrscheinlichkeit ist

$$\Pr[E_i] = 1/m \text{ für } i = 1, \dots, q$$

- Also folgt

$$\Pr[E_1 \cup \dots \cup E_q] = 1 - \Pr[\bar{E}_1 \cap \dots \cap \bar{E}_q] = 1 - (1 - 1/m)^q$$



Das Zufallsorakelmodell (ZOM)

Folgender Algorithmus liefert uns eine obere Schranke für die Komplexität des Problems P2, ein zweites Urbild für $h(x)$ zu bestimmen

Prozedur FindSecondPreimage(h, x, q)

```

1   $y := h(x)$ 
2  wähle eine beliebige Menge  $X_0 = \{x_1, \dots, x_{q-1}\} \subseteq X - \{x\}$ 
3  for each  $x_i \in X_0$  do
4      if  $h(x_i) = y$  then return( $x_i$ )
5  return ?
  
```

Satz

FINDSECONDPREIMAGE(h, x, q) gibt im ZOM mit Wahrscheinlichkeit $\varepsilon = 1 - (1 - 1/m)^{q-1}$ ein zweites Urbild $x_0 \neq x$ von $y = h(x)$ aus.

Der Beweis ist analog zum Beweis des vorherigen Satzes

Der Geburtstagsangriff

- Ist q vergleichsweise klein, so ist bei beiden bisher betrachteten Angriffen $\varepsilon \approx q/m$
- Um also auf eine Erfolgswahrscheinlichkeit von $1/2$ zu kommen, ist $q \approx m/2$ zu wählen
- Geht es lediglich darum, *irgendein* Kollisionspaar (x, x') aufzuspüren, so bietet sich ein sogenannter **Geburtstagsangriff** an
- Dieser lässt sich deutlich zeiteffizienter realisieren
- Wie der Name schon andeutet, basiert dieser Angriff auf dem sog. **Geburtstagsparadoxon**, welches in seiner einfachsten Form folgendes besagt

Geburtstagsparadoxon

Bereits in einer Klasse mit 23 Kindern ist die Wahrscheinlichkeit größer $1/2$, dass mindestens zwei Kinder am gleichen Tag Geburtstag haben

Der Geburtstagsangriff

- Der nächste Satz besagt, dass bei q -maligem Ziehen (mit Zurücklegen) aus einer Urne mit m Kugeln mit einer Wahrscheinlichkeit von

$$1 - (m-1)(m-2) \cdots (m-q+1)/m^{q-1}$$

mindestens eine Kugel mehrmals gezogen wird

- Für $m = 365$ und $q = 23$ ergibt dies einen Wert von ungefähr 0,507
- Da die Häufigkeiten der Geburtstage in einer Klasse nicht gleichverteilt sind, ist die Wahrscheinlichkeit, dass 2 Kinder am gleichen Tag Geburtstag haben, sogar noch etwas höher
- Zur Kollisionsbestimmung verwenden wir folgenden Algorithmus

Prozedur Collision(h, q)

-
- 1 wähle eine beliebige Menge $X_0 = \{x_1, \dots, x_q\} \subseteq X$
 - 2 **for** each $x_i \in X_0$ **do** $y_i := h(x_i)$
 - 3 **if** $\exists i \neq j : y_i = y_j$ **then return** (x_i, x_j) **else return** ?
-

Prozedur Collision(h, q)

- 1 wähle eine beliebige Menge $X_0 = \{x_1, \dots, x_q\} \subseteq X$
- 2 **for** each $x_i \in X_0$ **do** $y_i := h(x_i)$
- 3 **if** $\exists i \neq j : y_i = y_j$ **then return** (x_i, x_j) **else return** ?

- Bei einer naiven Implementierung würde zwar der Zeitaufwand für die Auswertung der if-Bedingung quadratisch von q abhängen
- Trägt man aber jeden Text x unter dem Suchwort $h(x)$ in eine Hash-tabelle der Größe q ein, so wird der Zeitaufwand für jeden einzelnen Text x im wesentlichen durch die Berechnung von $h(x)$ bestimmt

Satz

COLLISION(h, q) gibt im ZOM mit Erfolgswahrscheinlichkeit

$$\varepsilon = 1 - \frac{(m-1)(m-2) \cdots (m-q+1)}{m^{q-1}}$$

ein Kollisionspaar (x, x') für h aus

Beweis.

- Sei $X_0 = \{x_1, \dots, x_q\}$ und für $i = 1, \dots, q$ bezeichne E_i das Ereignis $h(x_i) \notin \{h(x_1), \dots, h(x_{i-1})\}$
- Dann ist $E_1 \cap \dots \cap E_q$ das Ereignis “COLLISION(h, q) gibt ? aus”
- Für $i = 1, \dots, q$ gilt nun

$$\Pr[E_i | E_1 \cap \dots \cap E_{i-1}] = \frac{m - i + 1}{m}$$

- Dies führt auf die Erfolgswahrscheinlichkeit

$$\begin{aligned}\varepsilon &= 1 - \Pr[E_1 \cap \dots \cap E_q] \\ &= 1 - \Pr[E_1] \Pr[E_2 | E_1] \cdots \Pr[E_q | E_1 \cap \dots \cap E_{q-1}] \\ &= 1 - \left(\frac{m-1}{m}\right) \left(\frac{m-2}{m}\right) \cdots \left(\frac{m-q+1}{m}\right)\end{aligned}$$

- Mit der Approximation $1 - x \approx e^{-x}$ erhalten wir folgende Abschätzung für ε :

$$\begin{aligned}\varepsilon &= 1 - \prod_{i=1}^{q-1} \left(1 - \frac{i}{m}\right) \\ &\approx 1 - \prod_{i=1}^{q-1} e^{-\frac{i}{m}} = 1 - e^{-\frac{1}{m} \sum_{i=1}^{q-1} i} = 1 - e^{-\frac{q(q-1)}{2m}} \\ &\approx 1 - e^{-\frac{q^2}{2m}} \approx q^2/2m\end{aligned}$$

- Für q erhalten wir daraus die Abschätzung

$$q \approx c_\varepsilon \sqrt{m}$$

mit einer von ε abhängigen Konstante $c_\varepsilon = \sqrt{2\varepsilon}$

- Diese Abschätzung ist nur für ε -Werte nahe Null hinreichend genau

Der Geburtstagsangriff

- Aus der Abschätzung $\varepsilon \approx 1 - e^{-\frac{q^2}{2m}}$ für ε (siehe vorige Folie) erhalten wir insbesondere für größere Werte von ε eine bessere Abschätzung für q :

$$q \approx c'_\varepsilon \sqrt{m}$$

mit der Konstanten $c'_\varepsilon = \sqrt{2 \ln \frac{1}{1-\varepsilon}}$

- Für $\varepsilon = 1/2$ ergibt sich somit $q \approx \sqrt{(2 \ln 2)m} \approx 1,17\sqrt{m}$
- Besitzt also eine binäre Hashfunktion $h: \{0,1\}^n \rightarrow \{0,1\}^m$ die Hashwertlänge $m = 128$ Bit, so müssen im ZOM $q \approx 1,17 \cdot 2^{64}$ Texte gehasht werden, um mit einer Wahrscheinlichkeit von $1/2$ eine Kollision zu finden
- Um einem Geburtstagsangriff widerstehen zu können, sollte eine Hashfunktion mindestens eine Hashwertlänge von 128 oder besser 160 Bit haben

- Im Folgenden beschäftigen wir uns mit der Frage, wie sich aus einer kollisionsresistenten Kompressionsfunktion

$$h: \{0, 1\}^{m+t} \rightarrow \{0, 1\}^m$$

eine kollisionsresistente Hashfunktion

$$\hat{h}: \{0, 1\}^* \rightarrow \{0, 1\}^l$$

konstruieren lässt

- Hierzu betrachten wir folgende kanonische Konstruktionsmethode:

Iterierte Hashfunktionen

Preprocessing: Transformiere $x \in \{0, 1\}^*$ mittels einer Funktion

$$y: \{0, 1\}^* \rightarrow \bigcup_{r \geq 1} \{0, 1\}^{rt}$$

zu einem String $y(x)$ mit der Eigenschaft $|y(x)| \equiv_t 0$

Processing: Sei $IV \in \{0, 1\}^m$ ein öffentlich bekannter

Initialisierungsvektor und sei $y(x) = y_1 \cdots y_r$ mit $|y_i| = t$ für $i = 1, \dots, r$.

Berechne eine Folge $z_0, \dots, z_r$ von Strings $z_i \in \{0, 1\}^m$ wie folgt:

$$z_i = \begin{cases} IV, & i = 0, \\ h(z_{i-1}y_i), & i = 1, \dots, r \end{cases}$$

Optionale Ausgabetransformation: Berechne den Wert $\hat{h}(x) = g(z_r)$, wobei $g: \{0, 1\}^m \rightarrow \{0, 1\}^l$ eine öffentlich bekannte Funktion ist (meist wird für g die Identität verwendet)

Zur Berechnung von $\hat{h}(x)$ wird also die Funktion h genau r -mal aufgerufen

Wir formulieren nun eine für Preprocessing-Funktionen wünschenswerte Eigenschaft

Definition

- Eine Funktion $y: \{0,1\}^* \rightarrow \{0,1\}^*$ heißt **suffixfrei**, falls es keine Strings $x \neq \tilde{x}$ und z in $\{0,1\}^*$ mit $y(\tilde{x}) = zy(x)$ gibt
- Mit anderen Worten: kein Funktionswert $y(x)$ ist Suffix eines Funktionswertes $y(\tilde{x})$ an einer Stelle $\tilde{x} \neq x$

Man beachte, dass jede suffixfreie Funktion insbesondere injektiv ist

Iterierte Hashfunktionen

Satz

Falls die Preprocessing-Funktion y suffixfrei und die Ausgabetransformation g injektiv ist, so ist mit h auch $\hat{h}$ kollisionsresistent

Beweis.

- Wir nehmen an, dass es gelingt, ein Kollisionspaar $(x, \tilde{x})$ für $\hat{h}$ zu finden (d.h. $\hat{h}(x) = \hat{h}(\tilde{x})$ und $x \neq \tilde{x}$)
- Seien $y(x) = y_1 \dots y_r$ und $y(\tilde{x}) = \tilde{y}_1 \dots \tilde{y}_s$ mit $r \leq s$
- Da y suffixfrei ist, muss ein Index $i \in \{1, \dots, r\}$ mit $y_i \neq \tilde{y}_{s-r+i}$ existieren
- Weiter seien z_i ($i = 0, \dots, r$) und $\tilde{z}_j$ ($j = 0, \dots, s$) die in der Processing-Phase berechneten Hashwerte
- Da g injektiv ist, muss mit $g(z_r) = \hat{h}(x) = \hat{h}(\tilde{x}) = g(\tilde{z}_s)$ auch $z_r = \tilde{z}_s$ gelten

Iterierte Hashfunktionen

Beweis.

- Wir nehmen an, dass es gelingt, ein Kollisionspaar $(x, \tilde{x})$ für $\hat{h}$ zu finden (d.h. $\hat{h}(x) = \hat{h}(\tilde{x})$ und $x \neq \tilde{x}$)
- Seien $y(x) = y_1 \dots y_r$ und $y(\tilde{x}) = \tilde{y}_1 \dots \tilde{y}_s$ mit $r \leq s$
- Da y suffixfrei ist, muss ein Index $i \in \{1, \dots, r\}$ mit $y_i \neq \tilde{y}_{s-r+i}$ existieren
- Weiter seien z_i ($i = 0, \dots, r$) und $\tilde{z}_j$ ($j = 0, \dots, s$) die in der Processing-Phase berechneten Hashwerte
- Da g injektiv ist, muss mit $g(z_r) = \hat{h}(x) = \hat{h}(\tilde{x}) = g(\tilde{z}_s)$ auch $z_r = \tilde{z}_s$ gelten
- Sei $i_{\max}$ der größte Index $i \in \{1, \dots, r\}$ mit $z_{i-1}y_i \neq \tilde{z}_{s-r+i-1}\tilde{y}_{s-r+i}$
- Dann bilden $z_{i_{\max}-1}y_{i_{\max}}$ und $\tilde{z}_{s-r+i_{\max}-1}\tilde{y}_{s-r+i_{\max}}$ wegen

$$h(z_{i_{\max}-1}y_{i_{\max}}) = z_{i_{\max}} = \tilde{z}_{s-r+i_{\max}} = h(\tilde{z}_{s-r+i_{\max}-1}\tilde{y}_{s-r+i_{\max}})$$

ein Kollisionspaar für h

- Merkle und Damgaard schlugen 1989 folgende konkrete Realisierung ihrer Konstruktion vor
- Als Initialisierungsvektor wird der Nullvektor $IV = 0^m$ benutzt, die optionale Ausgabetransformation entfällt, und für $y(x)$ wird im Fall $t \geq 2$ die folgende Funktion verwendet (den Fall $t = 1$ betrachten wir später)

Die Merkle-Damgaard-Konstruktion

- Für $x = \varepsilon$ sei $y(x) = 0^t$
- Für $x \in \{0, 1\}^n$ mit $n > 0$ sei $r = \lceil \frac{n}{t-1} \rceil$ und $x = x_1 x_2 \dots x_{r-1} x_r$ mit $|x_1| = |x_2| = \dots = |x_{r-1}| = t - 1$ sowie $|x_r| = t - 1 - d$, wobei $0 \leq d < t - 1$
- Im Fall $r = 1$ ist dann $y(x) = y_1 y_2$ mit $y_1 = 0x0^d$ und $y_2 = 1\text{bin}_{t-1}(d)$
- Und für $r > 1$ ist $y(x) = y_1 \dots y_{r+1}$, wobei

$$y_i = \begin{cases} 0x_1, & i = 1, \\ 1x_i, & 2 \leq i < r, \\ 1x_r 0^d, & i = r, \\ 1\text{bin}_{t-1}(d), & i = r + 1, \end{cases} \quad (1)$$

und $\text{bin}_{t-1}(d)$ die durch führende Nullen auf die Länge $t - 1$ aufgefüllte Binärdarstellung von d ist

Satz

Die durch (1) definierte Preprocessing-Funktion y ist suffixfrei

Beweis.

- Seien $x \neq \tilde{x}$ zwei Texte mit $|x| \leq |\tilde{x}|$
- Wir müssen zeigen, dass $y(x) = y_1 y_2 \dots y_{r+1}$ kein Suffix von $y(\tilde{x}) = \tilde{y}_1 \tilde{y}_2 \dots \tilde{y}_{s+1}$ ist
- Im Fall $x = \varepsilon$ ist dies klar
- Für $x \neq \varepsilon$ machen wir folgende Fallunterscheidung
 1. Fall: $|x| \not\equiv_{t-1} |\tilde{x}|$. Dann folgt $d \neq \tilde{d}$ und somit $y_{r+1} \neq \tilde{y}_{s+1}$
 2. Fall: $|x| \equiv_{t-1} |\tilde{x}|$. In diesem Fall ist $r = s$. Wegen $x \neq \tilde{x}$ existiert ein Index $i \in \{1, \dots, r\}$ mit $x_i \neq \tilde{x}_i$. Dies impliziert $y_i \neq \tilde{y}_i$, also ist $y(x)$ kein Suffix von $y(\tilde{x})$
 3. Fall: $|x| \neq |\tilde{x}|$ und $|x| \equiv_{t-1} |\tilde{x}|$. In diesem Fall ist $r < s$. Da $y(x)$ mit einer Null beginnt, aber das $(s - r + 1)$ -te Bit von $y(\tilde{x})$ eine Eins ist, kann $y(x)$ kein Suffix von $y(\tilde{x})$ sein

Nun betrachten wir den Fall $t = 1$

- Sei y die durch $y(x) := 11f(x)$ definierte Funktion, wobei f wie folgt definiert ist:

$$f(x_1 \dots x_n) = f(x_1) \dots f(x_n) \text{ mit } f(0) = 0 \text{ und } f(1) = 01$$

- Dann ist leicht zu sehen, dass y suffixfrei ist



- Da die Kompressionsfunktion h bei der Berechnung von $\hat{h}(x)$ im Fall $t = 1$ für jedes Bit von $y(x)$ einmal aufgerufen wird, wird h genau $|y(x)| \leq 2(n+1)$ -mal aufgerufen
- Im Fall $t > 1$ werden dagegen nur $r + 1 = \lceil \frac{n}{t-1} \rceil + 1$ Aufrufe benötigt

Die MD4-Hashfunktion

- Die MD4-Hashfunktion (*Message Digest*) wurde 1990 von Rivest vorgeschlagen
- Die Bitlänge von MD4 beträgt $l = 128$ Bit
- Bei einer Wortlänge von 32 Bit entspricht dies 4 Wörtern
- MD4 und die im Folgenden vorgestellten Hashfunktionen benutzen u.a. folgende Operationen auf Wörtern $X, Y \in \{0, 1\}^{32}$

Wort-Operationen	
$X \wedge Y$	bitweises „Und“ von X und Y
$X \vee Y$	bitweises „Oder“ von X und Y
$X \oplus Y$	bitweises „exklusives Oder“ von X und Y
$\neg X$	bitweises Komplement von X
$X + Y$	Ganzzahl-Addition modulo 2^{32}
$X \rightarrow s$	Rechtssshift um s Stellen
$X \leftarrow s$	zirkulärer Linkssshift um s Stellen

- Die Ganzzahl-Addition wird bei MD4 und MD5 in **little endian** Architektur ausgeführt
- D.h. dass ein aus 4 Bytes, zusammengesetztes Wort $X = a_3a_2a_1a_0$, dessen Bytes $a_i \in 2^8$ die Zahlenwerte $(a_i)_2 \in [0, 255]$ haben, die Zahl $(a_0)_22^{24} + (a_1)_22^{16} + (a_2)_22^8 + (a_3)_2$ repräsentiert
- Dagegen verwendet SHA-1 eine **big endian** Architektur
- D.h. dass $X = a_3a_2a_1a_0$ die Zahl $(a_3)_22^{24} + (a_2)_22^{16} + (a_1)_22^8 + (a_0)_2$ repräsentiert
- Der MD4-Algorithmus benutzt die folgenden Funktionen f_j für $j = 0, \dots, 47$:

$$f_j(X, Y, Z) := \begin{cases} (X \wedge Y) \vee (\neg X \wedge Z), & j = 0, \dots, 15 \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), & j = 16, \dots, 31 \\ X \oplus Y \oplus Z, & j = 32, \dots, 47 \end{cases}$$

- Zudem benutzt er die folgenden Konstanten y_j, z_j, s_j für $j = 0, \dots, 47$:

	y_j (in Hexadezimaldarstellung)
$j = 0, \dots, 15$	0
$j = 16, \dots, 31$	5a827999
$j = 32, \dots, 47$	6ed9eba1

	z_j
$j = 0, \dots, 15$	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
$j = 16, \dots, 31$	0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15
$j = 32, \dots, 47$	0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15

	s_j
$j = 0, \dots, 15$	3, 7, 11, 19, 3, 7, 11, 19, 3, 7, 11, 19, 3, 7, 11, 19
$j = 16, \dots, 31$	3, 5, 9, 13, 3, 5, 9, 13, 3, 5, 9, 13, 3, 5, 9, 13
$j = 32, \dots, 47$	3, 9, 11, 15, 3, 9, 11, 15, 3, 9, 11, 15, 3, 9, 11, 15

Die MD4-Hashfunktion

MD4(x)

```

1 input  $x \in \{0, 1\}^*, |x| = n$ 
2  $y := x10^k \text{bin}_{64}(n)$ ,  $k \in \{0, 1, \dots, 511\}$  mit
    $n + 1 + k + 64 \equiv 0 \pmod{512}$ 
3  $(H_1, H_2, H_3, H_4) := (67452301, \text{efcdab89}, 98\text{badcfe}, 10325476)$ 
4 sei  $y = M_1 \cdots M_r$ ,  $r = (n + 1 + k + 64)/512$ 
5 for  $i := 1$  to  $r$  do
6   sei  $M_i = X[0] \cdots X[15]$ 
7    $(A, B, C, D) := (H_1, H_2, H_3, H_4)$ 
8   for  $j := 0$  to 47 do
9      $(A, B, C, D) := (D, (A + f_j(B, C, D) + X[z_j] + y_j) \leftarrow s_j, B, C)$ 
10     $(H_1, H_2, H_3, H_4) := (H_1 + A, H_2 + B, H_3 + C, H_4 + D)$ 
11 output  $H_1 H_2 H_3 H_4$ 

```

- In Zeile 9 wird die **Kompressionsfunktion** von MD4 berechnet:
 $(A, B, C, D, X[z_j]) \mapsto (D, (A + f_j(B, C, D) + X[z_j] + y_j) \leftarrow s_j, B, C)$

- Für MD4 konnten nach ca. 2^{20} Hashwertberechnungen Kollisionen aufgespürt werden
- Deshalb gilt MD4 heutzutage nicht mehr als kollisionsresistent

- Der MD5 ist eine 1991 von Rivest präsentierte verbesserte Version von MD4
- Die Bitlänge von MD5 beträgt wie bei MD4 $l = 128$ Bit
- Bei einer Wortlänge von 32 Bit entspricht dies 4 Wörtern
- In MD5 werden teilweise andere Konstanten als in MD4 verwendet
- Zudem besitzt MD5 eine zusätzliche 4. Runde ($j = 48, \dots, 63$), in der die Funktion $f_j(X, Y, Z) = Y \oplus (X \vee \neg Z)$ verwendet wird
- Außerdem wurde die in Runde 2 von MD4 verwendete Funktion durch $f_j(X, Y, Z) := (X \wedge Z) \vee (Y \wedge \neg Z)$, $j = 16 \dots 31$, ersetzt
- Die y_j -Konstanten sind definiert als
$$y_j := \text{die ersten 32 Bit der Binärdarstellung von } \text{abs}(\sin(j + 1)), \\ 0 \leq j \leq 63,$$

- Zudem benutzt der MD5 die folgenden Konstanten z_j und s_j :

j	z_j
$0, \dots, 15$	$z_j = j$: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
$16, \dots, 31$	$z_j = (5j + 1) \bmod 16$: 1, 6, 11, 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12
$32, \dots, 47$	$z_j = (3j + 5) \bmod 16$: 5, 8, 11, 14, 1, 4, 7, 10, 13, 0, 3, 6, 9, 12, 15, 2
$48, \dots, 63$	$z_j = 7j \bmod 16$: 0, 7, 14, 5, 12, 3, 10, 1, 8, 15, 6, 13, 4, 11, 2, 9
j	s_j
$0, \dots, 15$	7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22
$16, \dots, 31$	5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20
$32, \dots, 47$	4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23
$48, \dots, 63$	6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21

MD5(x)

```
1 input  $x \in \{0, 1\}^*, |x| = n$   
2  $y := x10^k \text{bin}_{64}(n)$ ,  $k \in \{0, 1, \dots, 511\}$  mit  
    $n + 1 + k + 64 \equiv 0 \pmod{512}$   
3  $(H_1, H_2, H_3, H_4) := (67452301, \text{efcdab89}, 98\text{badcfe}, 10325476)$   
4 sei  $y = M_1 \cdots M_r$ ,  $r = (n + 1 + k + 64)/512$   
5 for  $i := 1$  to  $r$  do  
6   sei  $M_i = X[0] \cdots X[15]$   
7    $(A, B, C, D) := (H_1, H_2, H_3, H_4)$   
8   for  $j := 0$  to  $63$  do  
9      $(A, B, C, D) := (D, \textcolor{red}{B + (A + f_j(B, C, D) + X[z_j] + y_j)} \leftarrow s_j, B, C)$   
10     $(H_1, H_2, H_3, H_4) := (H_1 + A, H_2 + B, H_3 + C, H_4 + D)$   
11 output  $H_1 H_2 H_3 H_4$ 
```

- Für MD5 konnten in 2004 ebenfalls Kollisionspaare gefunden werden
- Für die **Kompressionsfunktion** von MD5 gelang dies bereits 1996

Die SHA-1-Hashfunktion

- Der **Secure Hash Algorithm** (SHA-1) ist eine Weiterentwicklung des MD4 bzw. MD5 Algorithmus
- Er gilt in den USA als Standard und ist Bestandteil des von der US-Behörde NIST (National Institute of Standards and Technology) im August 1991 veröffentlichten DSS (Digital Signature Standard)
- Die Bitlänge von SHA-1 beträgt $l = 160$ Bit
- Bei einer Wortlänge von 32 Bit entspricht dies 5 Wörtern
- SHA-1 unterscheidet sich nur geringfügig von der SHA-0 Hashfunktion, in der eine Schwachstelle dazu führt, dass nach Berechnung von ca. 2^{61} Hashwerten ein Kollisionspaar gefunden werden kann (obwohl bei einem Geburtstagsangriff auf Grund der Hashwertlänge von 160 Bit ca. 2^{80} Berechnungen erforderlich sein müssten)
- Diese potentielle Schwäche von SHA-0 wurde im SHA-1 dadurch entfernt, dass SHA-1 in Zeile 8 einen zirkulären Shift um eine Bitstelle ausführt

- Der SHA-1-Algorithmus benutzt die folgenden Konstanten K_j für $j = 0, \dots, 79$:

	K_j (in Hexadezimaldarstellung)
$j = 0, \dots, 19$	5a827999
$j = 20, \dots, 39$	6ed9eba1
$j = 40, \dots, 59$	8f1bbcdc
$j = 60, \dots, 79$	ca62c1d6

und folgende Funktionen f_j für $j = 0, \dots, 79$:

$$f_j(X, Y, Z) := \begin{cases} (X \wedge Y) \vee (\neg X \wedge Z), & j = 0, \dots, 19 \\ X \oplus Y \oplus Z, & j = 20, \dots, 39 \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), & j = 40, \dots, 59 \\ X \oplus Y \oplus Z, & j = 60, \dots, 79 \end{cases}$$

SHA-1(x)

```
1 input  $x \in \{0, 1\}^*, |x| = n$ 
2  $y := x10^k \text{bin}_{64}(n)$ ,  $k \in \{0, 1, \dots, 511\}$  mit
    $n + 1 + k + 64 \equiv 0 \pmod{512}$ 
3  $(H_0, \dots, H_4) := (67452301, \text{efcdab89}, 98\text{badcfe}, 10325476, \text{c3d2e1f0})$ 
4 sei  $y = M_1 \cdots M_r$ ,  $r = (n + 1 + k + 64)/512$ 
5 for  $i := 1$  to  $r$  do
6   sei  $M_i = X[0] \cdots X[15]$ 
7   for  $t := 16$  to  $79$  do
8      $X[t] := (X[t - 3] \oplus X[t - 8] \oplus X[t - 14] \oplus X[t - 16]) \leftarrow 1$ 
9      $(A, B, C, D, E) := (H_0, H_1, H_2, H_3, H_4)$ 
10    for  $j := 0$  to  $79$  do
11       $\text{temp} := (A \leftarrow 5) + f_j(B, C, D) + E + X[j] + K_j$ 
12       $(A, B, C, D, E) := (\text{temp}, A, B \leftarrow 30, C, D)$ 
13       $(H_0, \dots, H_4) := (H_0 + A, \dots, H_4 + E)$ 
14 output  $H_0 H_1 H_2 H_3 H_4$ 
```

- Im Jahr 2001 veröffentlichte die US-Behörde NIST drei weitere Hashfunktionen der SHA-Familie: SHA-256, SHA-384, and SHA-512
- Diese Funktionen werden auch als SHA-2 Hashfunktionen bezeichnet
- In 2004 kam noch SHA-224 als vierte Variante hinzu
- SHA-256 und SHA-512 haben denselben Aufbau, unterscheiden sich aber in erster Linie in der benutzten Wortlänge: 32 Bit bei SHA-256 und 64 Bit bei SHA-512
- Zudem werden unterschiedliche Shift- und Summationskonstanten verwendet und auch die Rundenzahlen differieren
- SHA-224 und SHA-384 sind reduzierte Varianten von SHA-256 und SHA-512

- Der SHA-256-Algorithmus benutzt die folgenden Konstanten K_j , $j = 0, \dots, 63$ (in Hexadezimaldarstellung):

428a2f98, 71374491, b5c0fbcf, e9b5dba5, 3956c25b, 59f111f1, 923f82a4, ab1c5ed5,
d807aa98, 12835b01, 243185be, 550c7dc3, 72be5d74, 80deb1fe, 9bdc06a7, c19bf174,
e49b69c1, efbe4786, 0fc19dc6, 240ca1cc, 2de92c6f, 4a7484aa, 5cb0a9dc, 76f988da,
983e5152, a831c66d, b00327c8, bf597fc7, c6e00bf3, d5a79147, 06ca6351, 14292967,
27b70a85, 2e1b2138, 4d2c6dfc, 53380d13, 650a7354, 766a0abb, 81c2c92e, 92722c85,
a2bfe8a1, a81a664b, c24b8b70, c76c51a3, d192e819, d6990624, f40e3585, 106aa070,
19a4c116, 1e376c08, 2748774c, 34b0bcb5, 391c0cb3, 4ed8aa4a, 5b9cca4f, 682e6ff3,
748f82ee, 78a5636f, 84c87814, 8cc70208, 90befffa, a4506ceb, bef9a3f7, c67178f2

- Dies sind jeweils die ersten 32 Bit der binären Nachkommastellen der dritten Wurzeln der ersten 64 Primzahlen $2, \dots, 311$

```

1 input  $x \in \{0, 1\}^*, |x| = n$ 
2  $y := x10^k \text{bin}_{64}(n)$ ,  $k \in \{0, \dots, 511\}$  mit  $n+1+k+64 \equiv 0 \pmod{512}$ 
3  $(H_0, \dots, H_7) := (6a09e667, \dots, 5be0cd19)$ 
4 sei  $y = M_1 \cdots M_r$ ,  $r = (n+1+k+64)/512$ 
5 for  $i := 1$  to  $r$  do
6   sei  $M_i = X[0] \cdots X[15]$ 
7   for  $t := 16$  to  $63$  do
8      $s0 := (X[t-15] \hookrightarrow 7) \oplus (X[t-15] \hookrightarrow 18) \oplus (X[t-15] \rightarrow 3)$ 
9      $s1 := (X[t-2] \hookrightarrow 17) \oplus (X[t-2] \hookrightarrow 19) \oplus (X[t-2] \rightarrow 10)$ 
10     $X[t] := X[t-16] + s0 + X[t-7] + s1$ 
11     $(A, B, C, D, E, F, G, H) := (H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7)$ 
12    for  $j := 0$  to  $63$  do  $\alpha$ 
13       $(H_0, H_1, \dots, H_7) := (H_0 + A, H_1 + B, \dots, H_7 + H)$ 
14 output  $H_0 H_1 H_2 H_3 H_4 H_5 H_6 H_7$ 

```

Die Werte von $H_0, \dots, H_7$ in Zeile 3 sind die ersten 32 Bit der binären Nachkommastellen der Wurzeln der Primzahlen 2, 3, 5, 7, 11, 13, 17, 19

Programmstück α

1	$s0 := (A \hookrightarrow 2) \oplus (A \hookrightarrow 13) \oplus (A \hookrightarrow 22)$
2	$maj := (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$
3	$t2 := s0 + maj$
4	$s1 := (E \hookrightarrow 6) \oplus (E \hookrightarrow 11) \oplus (E \hookrightarrow 25)$
5	$ch := (E \wedge F) \oplus (\neg E \wedge G)$
6	$t1 := H + s1 + ch + K_j + X[j]$
7	$(A, B, C, D, E, F, G, H) := (t1 + t2, A, B, C, D + t1, E, F, G)$

- Bereits 1991 wurden von den Boer und Bosselaers Schwächen im MD4 aufgedeckt
- Im August 2004 erschien ein Bericht [1] mit einer Anleitung, wie sich Kollisionen für MD4 mittels “hand calculation” finden lassen
- In 1993, fanden den Boer und Bosselaers einen Weg, so genannte “Pseudo-Kollisionen” für die MD5 Kompressionsfunktion zu generieren
- In 1996, fand Dobbertin ein Kollisionspaar für die MD5 Kompressionsfunktion
- Im August 2004 wurden schließlich Kollisionen für MD5 von Xiaoyun Wang, Dengguo Feng, Xuejia Lai und Hongbo Yu berechnet
- Der benötigte Aufwand wurde mit ca. 1 Stunde auf einem IBM p690 Cluster abgeschätzt

- Im März 2005 veröffentlichten Arjen Lenstra, Xiaoyun Wang und Benne de Weger zwei X.509 Zertifikate mit unterschiedlichen Public-keys, die auf denselben MD5-Hashwert führten
- Nur wenige Tage später beschrieb Vlastimil Klima eine Möglichkeit, Kollisionen für MD5 innerhalb weniger Stunden auf einem Notebook zu berechnen
- Mittels der so genannten Tunneling-Methode wurde die Rechenzeit vom gleichen Autor im März 2006 auf eine Minute verkürzt
- Auf der CRYPTO 98 stellten Chabaud und Joux einen Angriff auf SHA-0 vor, der ein Kollisionspaar mit nur 2^{61} Hashwertberechnungen (anstelle von 2^{80} bei einem Geburtstagsangriff) aufspürt

- In 2004 fanden Biham und Chen Beinahe-Kollisionen für den SHA-0, bei denen sich die Hashwerte nur an 18 von den 160 Bitpositionen unterschieden
- Zudem legten sie volle Kollisionen für den auf 62 Runden reduzierten SHA-0 Algorithmus vor
- Schließlich wurde im August 2004 die Berechnung einer Kollision für den vollen 80-Runden SHA-0 Algorithmus von Joux, Carribault, Lemuet und Jalby bekannt gegeben
- Hierzu wurden lediglich 2^{51} Hashwerte berechnet, die ca. 80 000 Stunden CPU-Rechenzeit auf einem 2-Prozessor 256-Itanium Supercomputer benötigten

- Ebenfalls im August 2004 wurde von Wang, Feng, Lai und Yu auf der CRYPTO 2004 eine Angriffsmethode für MD5, SHA-0 und andere Hashfunktionen vorgestellt, mit der sich die Anzahl der Hashwertberechnungen auf 2^{40} senken lässt
- Dies wurde im Februar 2005 von Xiaoyun Wang, Yiqun Lisa Yin und Hongbo Yu geringfügig auf 2^{39} Hashwertberechnungen verbessert
- Aufgrund der erfolgreichen Angriffe auf SHA-0 rieten mehrere Experten von einer weiteren Verwendung des SHA-1 ab. Daraufhin kündigte die amerikanische Behörde NIST an, SHA-1 in 2010 zugunsten der SHA-2 Varianten abzulösen

Kryptoanalyse von Hashfunktionen (SHA-1 und SHA-2) ⁶⁸

- Im Jahr 2005 veröffentlichten Rijmen und Oswald einen Angriff, der mit weniger als 2^{80} Hashwertberechnungen ein Kollisionspaar für den auf 53 Runden reduzierten SHA-1 Algorithmus findet
- Nur wenig später kündigten Xiaoyun Wang, Yiqun Lisa Yin und Hongbo Yu einen Angriff auf den vollen 80-Runden SHA-1 mit 2^{69} Hashwertberechnungen an
- Im August 2005 erfuhr der benötigte Aufwand von Xiaoyun Wang, Andrew Yao und Frances Yao auf der CRYPTO 2005 eine weitere Reduktion auf 2^{63} Berechnungen
- In 2008 wurde von Stephane Manuel ein Kollisionsangriff mit einem geschätzten Aufwand von 2^{51} bis 2^{57} Berechnungen veröffentlicht
- Im Februar 2017 fanden Stevens, Bursztein, Karpman, Albertini und Markov die erste Kollision für SHA-1
- Die besten bekannten Angriffe gegen SHA-2 brechen die von 64 auf 41 Runden reduzierte Variante von SHA-256 und die von 80 auf 46 Runden reduzierte Variante von SHA-512

- Im Oktober 2012 wurde der Hash-Algorithmus Keccak als Gewinner des vom NIST ausgeschriebenen Wettbewerbs für den SHA-3-Algorithmus ausgewählt
- Die Intention dabei war nicht, SHA-2 als Standard durch SHA-3 abzulösen, zumal bisher keine erfolgreichen Angriffe gegen SHA-2 bekannt sind
- Vielmehr ging es bei diesem Wettbewerb darum, angesichts der erfolgreichen Angriffe gegen MD5 und SHA-0, die einen ähnlichen Aufbau wie SHA-1 und SHA-2 haben, eine auf einem vollkommen anderen Entwurfsprinzip basierende Alternative zur Verfügung zu stellen

- Die Konstruktionsidee hinter dem SHA-3-Gewinner Keccak wird von den Autoren als *Sponge* (Schwamm) bezeichnet
- Auf der Basis dieser Entwurfsmethode lassen sich außer Hashfunktionen bspw. auch Pseudozufallsgeneratoren gewinnen
- Der Aufbau eines Sponges ähnelt oberflächlich betrachtet der bereits vorgestellten Konstruktion von iterierten Hashfunktionen, weist aber einige Unterschiede auf
- So basiert ein Sponge statt auf einer Kompressionsfunktion h auf einer Permutation (oder allgemeiner Transformation) $f : \{0, 1\}^b \rightarrow \{0, 1\}^b$, die wie h iteriert angewendet wird
- Dabei wird der aktuelle b -Bitblock in zwei Teilblöcke der Länge r und c unterteilt, die als äußerer bzw. innerer Zustand bezeichnet werden

- Wie der Name schon sagt, verbleiben die Bits des inneren Zustands im Sponge, d.h. sie dienen nur zur Berechnung des nächsten Zustands und werden im Gegensatz zu den Bits des äußeren Zustands nicht unmittelbar für die Gewinnung der Ausgabe genutzt
- Die Anzahl c der Bits des inneren Zustands wird als **Kapazität** des Sponges bezeichnet und ist sein wichtigster Sicherheitsparameter
- Die Anzahl r der Bits des äußeren Zustands heißt **Bitrate**, wobei $r + c = b$ gelten muss

- Bevor die Funktion f im Kern des Algorithmus iteriert angewendet wird, um eine Zustandsfolge zu generieren, wird ein Preprocessing ausgeführt
- Die Anforderungen an diese Funktion beschreiben wir vorab

Definition

- Eine Funktion $y: \{0, 1\}^* \rightarrow \bigcup_{k \geq 1} \{0, 1\}^{kr}$ der Form $y(x) = xz$ heißt **Paddingfunktion** für Bitrate $r \geq 1$
- Eine solche Funktion heißt **sponge-konform** für Bitrate $r \geq 1$, falls gilt:
 - $\forall n \geq 0 \exists z \forall x \in \{0, 1\}^n : y(x) = xz$
 - $\forall k \geq 0 \forall x \neq x' : y(x) \neq y(x')0^{kr}$

Beispiel

- Es ist leicht zu sehen, dass die Funktion

$$\text{pad10}^*1_r(x) = x10^d1 \text{ mit } d = \min\{i \geq 0 \mid i + 2 + |x| \equiv_r 0\}$$

eine sponge-konforme Paddingfunktion für die Bitrate r ist

- Tatsächlich ist pad10^*1_r sogar für jede Bitrate $r' \geq 1$ sponge-konform
- Ohne die 1 am Ende von $\text{pad10}^*1_r(x) = x10^d1$ wäre dies nicht der Fall

Die Sponge-Konstruktion

Definition

- Sei y eine Paddingfunktion für $r \geq 1$ und sei $f : \{0, 1\}^b \rightarrow \{0, 1\}^b$
- Für $x \in \{0, 1\}^*$ sei $y(x) = y_1 \dots y_k$ mit $|y_i| = r$ für $i = 1, \dots, k$
- Wir definieren die Zustände

$$s_i = \begin{cases} 0^b & i = 0 \\ f(s_{i-1} \oplus (y_i 0^c)) & 1 \leq i \leq k \quad (\text{Absorptionsphase}) \\ f(s_{i-1}) & i > k \quad (\text{Squeezing-Phase}) \end{cases}$$

- Weiter bezeichne z_i für $i \geq 1$ die ersten r Bit von s_{k+i-1}
- Zudem sei $m = \lfloor \frac{l}{r} \rfloor$ und z'_{m+1} sei das Präfix von z_{m+1} der Länge $l - mr$
- Dann ist die Funktion $\text{Sponge}_{f,y,r} : \mathbb{N} \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ wie folgt definiert: $\text{Sponge}_{f,y,r}(l, x) = z_1 \dots z_m z'_{m+1}$
- Für die Analyse definieren wir noch die Funktionen

$$\text{Absorb}_{f,r}(y_1 \dots y_k) = s_k \text{ und } \text{Squeeze}_{f,r}(l, s_k) = z_1 \dots z_m z'_{m+1}$$

Die Sponge-Konstruktion

- Den Aufwand, für festes l ein Kollisionspaar x, x' mit $x \neq x'$ und $\text{Sponge}_{f,y,r}(l, x) = \text{Sponge}_{f,y,r}(l, x')$ zu finden, können wir nach oben durch den Aufwand abschätzen, ein Paar $x, x' \in \bigcup_{k \geq 1} \{0, 1\}^{kr}$ mit $x \neq x'$ und $\text{Absorb}_{f,r}(y(x)) = \text{Absorb}_{f,r}(y(x'))$ zu finden
- Hierbei reicht es, ein **inneres Kollisionspaar**, d.h. zwei Strings $w = y_1 \dots, y_k \neq w' = y'_1 \dots, y'_{k'}$ zu finden, so dass die inneren Zustände von $s_k = \text{Absorb}_{f,r}(w)$ und $s'_{k'} = \text{Absorb}_{f,r}(w')$ gleich sind
- Setzen wir nämlich y_{k+1} und $y'_{k'+1}$ auf die äußeren Zustände von s_k und $s'_{k'}$, so folgt für die Eingaben $x = wy_{k+1}$ und $x' = w'y'_{k'+1}$:

$$\begin{aligned} \text{Absorb}_{f,r}(x) &= f(s_k \oplus (y_{k+1} 0^c)) = f(0^r s_k^i) = f(0^r s_{k'}^i) \\ &= f(s'_{k'} \oplus (y'_{k'+1} 0^c)) = \text{Absorb}_{f,r}(x') \end{aligned}$$

wobei s_j^i den inneren Zustand von s_j bezeichnet

- Falls das Suffix z von $y(x) = xz$ nur von $|x| \bmod r$ abhängt, gilt wegen $|x| \equiv_r |x'|$ dann auch die Gleichheit $\text{Absorb}_{f,r}(y(x)) = \text{Absorb}_{f,r}(y(x'))$ und somit $\text{Sponge}_{f,y,r}(l, x) = \text{Sponge}_{f,y,r}(l, x')$

Die Sponge-Konstruktion

- Um eine solche innere Kollision zu finden, hilft es, sich die 2^c inneren Zustände $u \in \{0, 1\}^c$ als Knoten eines gerichteten Multigraphen G vorzustellen, der für jedes Paar $(xu, x'u')$ mit $f(xu) = x'u'$ eine Kante $(u, u')_{x, x'}$ von u nach u' mit dem Label x, x' enthält
- Ziel ist es dann, zwei verschiedene Pfade von 0^c zu demselben Knoten v zu finden, wobei zwei Pfade auch dann verschieden sind, wenn sich die Kanten nur in den Labels unterscheiden
- Wird f durch eine Zufallsfunktion modelliert (ZOM), so lassen bereits berechnete Werte von f keine Rückschlüsse auf die Werte für andere Argumente zu
- Anders als beim ZOM für eine Hashfunktion kann es sich dennoch für den Angreifer lohnen, die Argumente von f adaptiv nach einer Strategie S zu wählen
- Der Algorithmus `InnerCollision` fasst dieses Vorgehen zusammen

Prozedur InnerCollision($f, r, q, \mathcal{S}$)

```
1  $c := b - r$ , wobei  $f : \{0, 1\}^b \rightarrow \{0, 1\}^b$ 
2 initialisiere den Multi-Digraphen  $G = (V, A) := (\{0, 1\}^c, \emptyset)$ 
3 for  $i := 1$  to  $q$  do
4   wähle  $u \in V$  und  $x \in \{0, 1\}^r$  nach Strategie  $\mathcal{S}$ 
5    $x'u' := f(xu)$ 
6    $A := A \cup \{(u, u')_{x, x'}\}$ 
7   if  $\exists$  zwei Pfade  $(0^c, u_1)_{x_0, x'_0}, (u_1, u_2)_{x_1, x'_1}, \dots, (u_{k-1}, u_k)_{x_{k-1}, x'_{k-1}}$  und
8      $(0^c, v_1)_{y_0, y'_0}, (v_1, v_2)_{y_1, y'_1}, \dots, (v_{l-1}, v_l)_{y_{l-1}, y'_{l-1}}$  in  $G$  mit  $u_k = v_l$  then
9     return  $(x_0(x'_0 \oplus x_1) \dots (x'_{k-2} \oplus x_{k-1}), y_0(y'_0 \oplus y_1) \dots (y'_{l-2} \oplus y_{l-1}))$ 
10  else
11    return(?)
```

Satz

- Für jede Strategie $\mathcal{S}$ gibt $\text{INNERCOLLISION}(f, r, q, \mathcal{S})$ im ZOM mit Erfolgswahrscheinlichkeit höchstens

$$\varepsilon = 1 - \prod_{i=1}^q \left(1 - \frac{i}{2^c}\right)$$

ein inneres Kollisionspaar (x, x') aus

- Wählt $\mathcal{S}$ nur von 0^c aus erreichbare Knoten u und kein Argument xu mehrmals, so ist die Erfolgswahrscheinlichkeit exakt ε

Die Sponge-Konstruktion

Beweis.

- Sei E_i das Ereignis “ G enthält nach dem i -ten Durchlauf noch keine zwei verschiedenen Pfade von 0^c zu einem Knoten v ”
- Da nur durch eine Kante zwischen zwei von 0^c aus erreichbaren Knoten ein zweiter Pfad von 0^c aus geschlossen werden kann und nach $i - 1$ Durchläufen höchstens i von 2^c Knoten erreichbar sind, gilt

$$\Pr[E_i | E_1 \cap \dots \cap E_{i-1}] \geq 1 - \frac{i}{2^c}$$

- Wählt $\mathcal{S}$ nur erreichbare Knoten u und kein Argument xu mehrfach, so sind unter Annahme von $E_1 \cap \dots \cap E_{i-1}$ auch i Knoten erreichbar (sonst gäbe es bereits zwei Pfade von 0^c zu einem Knoten in G) und es gilt sogar Gleichheit
- Dies führt auf eine Erfolgswahrscheinlichkeit von

$$\begin{aligned} 1 - \Pr[E_1 \cap \dots \cap E_q] &= 1 - \Pr[E_1] \Pr[E_2 | E_1] \cdots \Pr[E_q | E_1 \cap \dots \cap E_{q-1}] \\ &\leq 1 - \left(1 - \frac{1}{2^c}\right) \left(1 - \frac{2}{2^c}\right) \cdots \left(1 - \frac{q}{2^c}\right) = \varepsilon \quad \square \end{aligned}$$

- Mit der Approximation $1 - x \approx e^{-x}$ erhalten wir die Abschätzung

$$\begin{aligned}\varepsilon &= 1 - \prod_{i=1}^q \left(1 - \frac{i}{2^c}\right) \approx 1 - \prod_{i=1}^q e^{-\frac{i}{2^c}} = 1 - e^{-\frac{1}{2^c} \sum_{i=1}^q i} \\ &= 1 - e^{-\frac{q(q+1)}{2 \cdot 2^c}} \approx 1 - e^{-\frac{q^2}{2 \cdot 2^c}} \approx q^2 / 2 \cdot 2^c\end{aligned}$$

- Für q ergibt sich daraus die Abschätzung

$$q \approx c_\varepsilon \sqrt{2^c}$$

mit einer von ε abhängigen Konstanten $c_\varepsilon = \sqrt{2\varepsilon}$

- Der Standard SHA-3 definiert die oben beschriebene Sponge-Konstruktion, 7 verschiedene bijektive Funktionen f_w , $w = 2^i$, $i \in \{0, \dots, 6\}$ als Kern von $\text{Sponge}_{f_w, \text{pad}10^*1_r, r}$, sowie verschiedene Kombinationen von Bitraten r und Ausgabelängen l (c ist durch $25w - r$ bestimmt)
- Jede Funktion $f_w : \{0, 1\}^{5 \times 5 \times w} \rightarrow \{0, 1\}^{5 \times 5 \times w}$ bildet ein zweidimensionales Feld A aus w -Bit-Wörtern auf ein ebensolches Feld $f_w(A)$ ab
- Dabei wird $(12 + \log_2 w)$ -mal eine Rundenfunktion $f'_w : \{0, 1\}^{5 \times 5 \times w} \times \{0, 1\}^w \rightarrow \{0, 1\}^{5 \times 5 \times w}$ aufgerufen, die A und eine Rundenkonstante RC_i auf A' abbildet
- Es gilt

$$f'_w(A, RC) = \iota_{RC}(\chi(\pi(\rho(\theta(A))))),$$

wobei θ , ρ , π , χ und ι_{RC} Bijektionen von $\{0, 1\}^{5 \times 5 \times w}$ nach $\{0, 1\}^{5 \times 5 \times w}$ sind

- Die Funktion θ besteht aus $\oplus$ -Operationen und ist so gewählt, dass sich $\theta^{-1}(A)$ an möglichst vielen Bits ändert, falls eines in A geflippt wird
- Danach permutieren die Funktionen ρ und π die Bits von A innerhalb und zwischen den Wörtern
- Ähnlich einer S-Box im SPN ist χ eine nichtlineare Funktion (die einzige solche in der Definition von f'_w), die nur auf 5-Bit-Blöcken arbeitet (jedes Bit hängt sogar nur von 2 anderen ab)
- Schlussendlich setzt ι_{RC} das Wort $A_{0,0}$ auf $A_{0,0} \oplus RC$
- Für die Werte $l \in \{224, 256, 384, 512\}$ definiert der Standard FIPS 202:

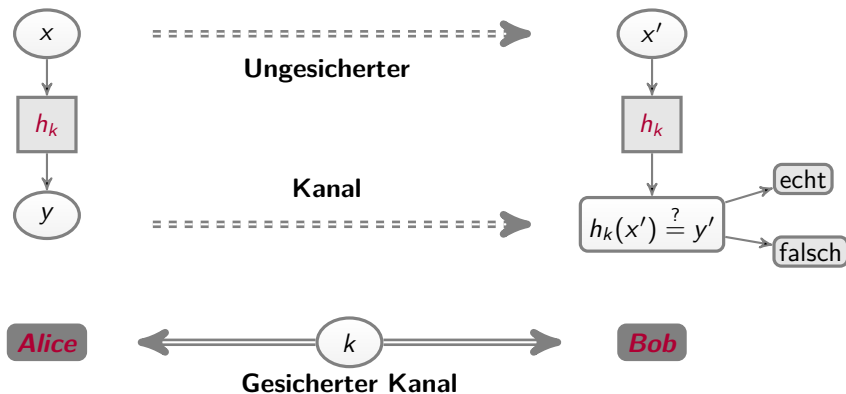
$$\text{SHA3-}l(x) = \text{Sponge}_{f_{64}, \text{pad}10^*1_r, r}(l, x01), \quad \text{wobei } r = 1600 - 2l$$

- Das zusätzliche Padding 01 soll dabei SHA-3 von anderen Anwendungen von Keccak mit denselben Werten w, l, r unterscheiden

Definition

Eine **Hashfamilie** $\mathcal{H} = (X, Y, K, H)$ wird durch folgende Komponenten beschrieben:

- X , eine endliche oder unendliche Menge von Texten
 - Y , endliche Menge aller möglichen **Hashwerte**, $\|Y\| \leq \|X\|$
 - K , endlicher **Schlüsselraum** (**key space**), wobei jeder Schlüssel $k \in K$ eine Hashfunktion $h_k: X \rightarrow Y$ in H spezifiziert, d.h. $H = \{h_k \mid k \in K\}$
-
- Im folgenden werden wir die Größe $\|X\|$ des Textraumes mit n , die des Hashwertbereiches Y mit m und die des Schlüsselraumes K mit l bezeichnen
 - Wir nennen dann $\mathcal{H}$ auch eine **(n, m, l)-Hashfamilie** oder einen **(n, m, l)-MAC**



- Hierbei ist k der symmetrische Authentifikationsschlüssel und $y = h_k(x)$ der **MAC**-Wert für x unter k
- Indem Alice ihre Nachricht x um den Hashwert $y = h_k(x)$ ergänzt, hat Bob nicht nur die Möglichkeit, anhand von y die empfangene Nachricht x' auf Manipulationen, sondern auch ihre Herkunft zu überprüfen

Damit ein geheimer Schlüssel k für die Authentifizierung mehrerer Nachrichten benutzt werden kann, ohne dass dies einem potentiellen Angreifer zur nichtautorisierten Berechnung von gültigen MAC-Werten verhilft, sollte folgende Bedingung erfüllt sein

Berechnungsresistenz: Auch wenn eine Reihe von unter einem Schlüssel k generierten Text-Hashwert-Paaren $(x_1, h_k(x_1)), \dots, (x_n, h_k(x_n))$ bekannt ist, erfordert es einen immensen Aufwand, ohne Kenntnis von k ein weiteres Paar (x, y) mit $y = h_k(x)$ zu finden

- Bei Verwendung eines berechnungsresistenten MACs ist es einem Angreifer nicht möglich, an Bob eine Nachricht x zu schicken, die Bob als von Alice stammend anerkennt
- Zu beachten ist allerdings, dass die Berechnungsresistenz nichts für den Fall aussagt, dass der Schlüssel k bekannt ist
- So kann nicht davon ausgegangen werden, dass die Funktion $h_k(x)$ bei bekanntem k die Einweg-Eigenschaft besitzt oder schwach (beziehungsweise stark) kollisionsresistent ist
- Es ist jedoch leicht zu sehen, dass es die Berechnungsresistenz erfordert, dass $h_k(x)$ bei geheimgehaltenem k zumindest schwach kollisionsresistent ist
- Dies ist etwa der Fall, wenn k im Speicher eines ausforschungssicheren Chips abgelegt wird

- Mithilfe eines berechnungsresistenten MACs kann der Integritätsschutz für mehrere Datensätze auf die Geheimhaltung eines Schlüssels k zurückgeführt werden
- Um die Datensätze $x_1, \dots, x_n$ gegen unbefugt vorgenommene Veränderungen zu schützen, legt man sie zusammen mit ihren MAC-Werten $y_1 = h_k(x_1), \dots, y_n = h_k(x_n)$ auf einem unsicheren Speichermedium ab und bewahrt den geheimen Schlüssel k an einem sicheren Ort auf
- Bei einem späteren Zugriff auf einen Datensatz x_i lässt sich dessen Unversehrtheit durch einen Vergleich von y_i mit dem Ergebnis $h_k(x_i)$ einer erneuten MAC-Berechnung überprüfen
- Da auf diese Weise ein wirksamer Schutz der Datensätze gegen Viren und andere Manipulationen erreicht wird, spricht man von einer **Versiegelung** der gespeicherten Datensätze

- Ein Angriff gegen einen MAC hat die unbefugte Berechnung von MAC-Werten zum Ziel
- Das heißt, der Angreifer versucht, MAC-Werte $h_k(x)$ ohne Kenntnis des geheimen Schlüssels k zu berechnen
- Entsprechend der Art des zur Verfügung stehenden Textmaterials lassen sich die Angriffe gegen einen MAC wie folgt klassifizieren

Impersonation:

Der Angreifer kennt nur den benutzten MAC und versucht ein Paar (x, y) mit $h_k(x) = y$ zu generieren, wobei k der (dem Angreifer unbekannte) Schlüssel ist

Substitution:

Der Angreifer versucht in Kenntnis eines Paares $(x, h_k(x))$ ein Paar (x', y') mit $x' \neq x$ und $h_k(x') = y'$ zu generieren

Angriff bei bekanntem Text (known-text attack):

Der Angreifer kennt für eine Reihe von Texten $x_1, \dots, x_r$ (die er nicht selbst wählen konnte) die zugehörigen MAC-Werte $h_k(x_1), \dots, h_k(x_r)$ und versucht, ein Paar (x', y') mit $h_k(x') = y'$ und $x' \notin \{x_1, \dots, x_r\}$ zu generieren

Angriff bei frei wählbarem Text (chosen-text attack):

Der Angreifer kann die Texte x_i selbst wählen

Angriff bei adaptiv wählbarem Text (adaptive chosen-text attack):

Der Angreifer kann die Wahl des Textes x_i von den zuvor erhaltenen MAC-Werten $h_k(x_j)$, $j < i$, abhängig machen

Wechseln die Anwender nach jeder MAC-Wertberechnung den Schlüssel, so genügt es, dass $\mathcal{H}$ einem **Impersonationsangriff** widersteht

Modell: Schlüssel k und Nachrichten x werden unabhängig gemäß einer Wahrscheinlichkeitsverteilung $p(k, x) = p(k)p(x)$ generiert, welche dem Angreifer bekannt ist

- Dabei nehmen wir an, dass $p(x) > 0$ und $p(k) > 0$ für alle $x \in X$ und $k \in K$ gilt
- Sei α die Wahrscheinlichkeit, mit der sich ein Angreifer bei optimaler Strategie als Alice ausgeben kann, ohne dass Bob dies bemerkt

Erfolgswahrscheinlichkeit für Impersonation

- Für ein Paar (x, y) sei $p(x \mapsto y)$ die Wahrscheinlichkeit, dass ein zufällig gewählter Schlüssel den Text x auf den MAC-Wert y abbildet:

$$p(x \mapsto y) = p(y|x) = \sum_{k \in K(x,y)} p(k)$$

wobei $K(x, y) = \{k \in K \mid h_k(x) = y\}$ alle Schlüssel enthält, die x auf y abbilden

- Bei einem Impersonationsangriff ist $p(x \mapsto y)$ also die Wahrscheinlichkeit, dass der Angreifer bei Wahl des Paares (x, y) Erfolg hat
- Deshalb bezeichnen wir diese Wahrscheinlichkeit auch mit $\alpha(x, y)$
- Schließlich ist $\alpha(x) = \max\{\alpha(x, y) \mid y \in Y\}$ die Wahrscheinlichkeit, mit der einem Angreifer bei optimaler Strategie eine Impersonation mit dem Text x gelingt
- Daher ist $\alpha = \max\{\alpha(x) \mid x \in X\}$

Beispiel

- Sei $K = \{1, 2, 3\}$, $X = \{a, b, c, d\}$ und $Y = \{0, 1\}$
- Wir beschreiben H durch die zugehörige **Authentikationsmatrix**
- Die Zeilen und Spalten dieser Matrix werden mit den Schlüsseln $k \in K$ und den Texten $x \in X$ indiziert und ihr Eintrag in Zeile k und Spalte x ist der Wert $h_k(x)$:

		0,1	0,2	0,3	0,4
		<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
0,25	1	0	0	0	1
0,30	2	1	1	0	1
0,45	3	0	1	1	0

- Die umrahmten Zahlen geben die Wahrscheinlichkeiten $p(x)$ bzw. $p(k)$ an

Beispiel (Fortsetzung)

- Dann hat der Angreifer folgende Erfolgsaussichten $\alpha(x)$, falls er an *Bob* den Text x senden möchte

x	a	b	c	d
$p(x \mapsto 0)$	0,7	0,25	0,55	0,45
$p(x \mapsto 1)$	0,3	0,75	0,45	0,55
$\alpha(x)$	0,7	0,75	0,55	0,55

- Folglich ist $\alpha = 0,75$



Satz

Für alle $x \in X$ ist $\alpha(x) \geq \frac{1}{m}$ und daher gilt $\alpha \geq \frac{1}{m}$

Beweis.

- Für beliebiges $x \in X$ gilt

$$\sum_{y \in Y} p(x \mapsto y) = \sum_{y \in Y} \sum_{k \in K(x,y)} p(k) = \sum_{k \in K} p(k) = 1$$

- Somit existiert für jedes $x \in X$ ein $y \in Y$ mit $p(x \mapsto y) \geq \frac{1}{m}$ und dies impliziert

$$\alpha(x) = \max_{y \in Y} p(x \mapsto y) \geq \frac{1}{m}$$



Bemerkung

- Wie der Beweis zeigt, gilt $\alpha = \frac{1}{m}$ genau dann, wenn für alle Paare $(x, y) \in X \times Y$ folgende Gleichheit gilt:

$$\sum_{k \in K(x,y)} p(k) = \frac{1}{m}$$

- D.h. bei Gleichverteilung der Schlüssel muss in jeder Spalte der Authentifikationsmatrix jeder MAC-Wert gleich oft vorkommen
- Dies lässt sich am einfachsten dadurch erreichen, dass man $K = Y$ setzt und für h_k die konstante Funktion $h_k(x) = k$ wählt

Ein Maß für den Informationsgehalt

- In der Informationstheorie wird die Unsicherheit über eine Nachrichtenquelle X nach ihrer Entropie bemessen
- Dabei entspricht die Unsicherheit über X genau dem Informationsgewinn, der sich aus der Beobachtung der Quelle X ergibt
- Intuitiv ist die in einer einzelnen Nachricht x steckende Information umso größer, desto unwahrscheinlicher sie ist
- Tritt eine Nachricht x mit der Wahrscheinlichkeit $p(x) = \Pr[X = x] > 0$ auf, dann ist ihr **Informationsgehalt** definiert als

$$\text{Inf}_X(x) = \log_2(1/p(x)) = -\log_2 p(x)$$

- Im Fall $p(x) = 0$ sei $\text{Inf}_X(x) = 0$

Ein Maß für den Informationsgehalt

- Diese Definition des Informationsgehalts ergibt sich zwangsläufig aus den beiden folgenden Axiomen:
 - Der (gemeinsame) Informationsgehalt $\text{Inf}_{X,Y}(x,y)$ von zwei Nachrichten x und y , die aus unabhängigen Quellen X und Y stammen, ist $\text{Inf}_X(x) + \text{Inf}_Y(y)$
 - Eine Nachricht x , die mit Wahrscheinlichkeit $\Pr[X = x] = 1/2$ auftritt, hat den Informationsgehalt $\text{Inf}_X(x) = 1$
- Die Einheit des Informationsgehalts ist bit (basic indissoluble information unit)
- Die Entropie von X ist nun der erwartete Informationsgehalt einer von X generierten Nachricht

Definition

- Sei X eine Zufallsvariable mit Wertebereich $W(X) = \{x_1, \dots, x_n\}$ und sei $p_i = \Pr[X = x_i]$
- Dann ist die **Entropie** von X definiert als

$$\mathcal{H}(X) = \sum_{i=1}^n p_i \ln_{\mathcal{H}}(x_i) = \sum_{i=1}^n p_i \log_2(1/p_i) = - \sum_{i=1}^n p_i \log_2(p_i)$$

Beispiel

- Sei X eine Zufallsvariable mit der Verteilung

x_i	sonnig	leicht bewölkt	bewölkt	stark bewölkt	Regen	Schnee	Nebel
p_i	1/4	1/4	1/8	1/8	1/8	1/16	1/16

- Dann ergibt sich die Entropie von X zu

$$\mathcal{H}(X) = 1/4 \cdot (2 + 2) + 1/8 \cdot (3 + 3 + 3) + 1/16 \cdot (4 + 4) = 2,625 \quad \triangleleft$$

Der Entropiebegriff

- Die Entropie nimmt im Fall der Gleichverteilung $p_1 = \dots = p_n = 1/n$ den Wert $\log_2(n)$ an, während sie für jede andere Verteilung auf einer n -elementigen Menge einen Wert $\mathcal{H}(X) < \log_2(n)$ hat (siehe unten)
- Die Unsicherheit über eine Zufallsvariable X ist um so größer, je größer der Wertebereich und je gleichmäßiger die Verteilung von X ist
- Bringt X zum Beispiel nur einen einzigen Wert mit positiver Wahrscheinlichkeit hervor, dann (und nur dann) hat $\mathcal{H}(X)$ den Wert 0
- Für den Nachweis von oberen Schranken für die Entropie benutzen wir folgende Hilfsmittel aus der Analysis

Definition

- Sei $I \subseteq \mathbb{R}$ ein Intervall. Eine Funktion $f : I \rightarrow \mathbb{R}$ heißt **konkav** auf I , falls für alle $x \neq y \in I$ und $0 \leq t \leq 1$ gilt:

$$f(tx + (1-t)y) \geq tf(x) + (1-t)f(y)$$

- Gilt sogar „ $>$ “ anstelle von „ $\geq$ “, so heißt f **streng konkav** auf I

Beispiel

Die Funktion $f(x) = \log_2(x)$ ist streng konkav auf $(0, \infty)$



Für den Beweis des nächsten Satzes benötigen wir die **Jensensche Ungleichung**, die wir ohne Beweis angeben

Satz

- Sei f eine streng konkave Funktion auf I und seien $0 < a_1, \dots, a_n < 1$ reelle Zahlen mit $\sum_{i=1}^n a_i = 1$
- Dann gilt für alle $x_1, \dots, x_n \in I$,

$$f\left(\sum_{i=1}^n a_i x_i\right) \geq \sum_{i=1}^n a_i f(x_i).$$

- Hierbei tritt Gleichheit genau dann ein, wenn alle x_i den gleichen Wert haben

Satz

- Sei X eine Zufallsvariable mit Wertebereich $W(X) = \{x_1, \dots, x_n\}$ und Verteilung $p_i = \Pr[X=x_i]$ für $i = 1, \dots, n$
- Dann gilt $H(X) \leq \log_2(n)$, wobei Gleichheit genau im Fall $p_i = 1/n$ für $i = 1, \dots, n$ eintritt

Beweis.

- Aufgrund der Jensenschen Ungleichung gilt

$$\mathcal{H}(X) = \sum_{i=1}^n p_i \log_2(1/p_i) \leq \log_2 \sum_{i=1}^n (p_i/p_i) = \log_2 n,$$

wobei Gleichheit genau im Fall $1/p_1 = \dots = 1/p_n$ eintritt

- Letzteres ist mit der Bedingung $p_i = 1/n$ für $i = 1, \dots, n$ gleichbedeutend



Definition. Seien X, Y Zufallsvariablen

Die **bedingte Entropie** von X unter Y ist definiert als

$$\mathcal{H}(X|Y) = \sum_{y \in W(Y)} p(y) \mathcal{H}(X|y) = \sum_y p(y) \sum_x p(x|y) \log_2(1/p(x|y)),$$

wobei $X|y$ die Zufallsvariable mit der Verteilung $p_y(x) = p(x|y)$ ist

Das folgende Lemma benötigen wir für den Beweis des nächsten Satzes

Lemma

- Sei $\mathcal{X}$ eine Zufallsvariable mit endlichem Wertebereich $W(\mathcal{X}) \subseteq \mathbb{R}^+$
- Dann gilt $\log E(\mathcal{X}) \geq E(\log \mathcal{X})$

Beweis.

- Sei $W(\mathcal{X}) = \{x_1, \dots, x_n\}$ und für $i = 1, \dots, n$ sei $p_i = \Pr[\mathcal{X} = x_i]$
- Da die Funktion $x \mapsto \log_2 x$ konkav ist, folgt mit der Jensenschen Ungleichung

$$\log E(\mathcal{X}) = \log_2\left(\sum p_i x_i\right) \geq \sum p_i \log_2 x_i = E(\log \mathcal{X})$$



Satz

- Für jeden MAC (X, Y, K, H) gilt:

$$\alpha \geq \frac{1}{2^{H(K)} - H(K|X,Y)} \geq 1/l$$

- Hierbei sind X, Y, K Zufallsvariablen, die die Verteilungen der Nachrichten, der MAC-Werte und der Schlüssel beschreiben

Der Wert von α kann also um so kleiner werden, je gleichmäßiger die Schlüsselverteilung ist und je mehr Information die Beobachtung eines gültigen Paares (x, y) über den Schlüssel liefert

Satz

Für jeden MAC (X, Y, K, H) gilt $\alpha \geq 1/2^{H(K)-H(K|X,Y)} \geq 1/l$

Beweis.

- Wegen $\alpha = \max_{x,y} \alpha(x,y)$ ist $E(\alpha(X,Y)) = \sum_{x,y} p(x,y)\alpha(x,y) \leq \alpha$
- Dabei ist $E(\alpha(X,Y))$ die Erfolgswahrscheinlichkeit eines (probabilistischen) Angreifers, der das Paar (x,y) gemäß der Verteilung (X,Y) wählt
- Somit folgt unter Anwendung von obigem Lemma

$$\begin{aligned} \log \alpha &\geq \log E(\alpha(X,Y)) \geq E(\log \alpha(X,Y)) \\ &= \sum_{x,y} \underbrace{p(x,y)}_{p(x)p(y|x)} \underbrace{\log \alpha(x,y)}_{\log p(y|x) = -\log \frac{1}{p(y|x)}} = -H(Y|X) \end{aligned}$$

Beweis.

- Somit folgt unter Anwendung von obigem Lemma

$$\log \alpha \geq -H(\mathcal{Y} | \mathcal{X})$$

- Zudem gilt

$$H(\mathcal{K}, \mathcal{Y}, \mathcal{X}) = H(\mathcal{X}) + H(\mathcal{Y} | \mathcal{X}) + H(\mathcal{K} | \mathcal{X}, \mathcal{Y})$$

und

$$H(\mathcal{K}, \mathcal{Y}, \mathcal{X}) = \underbrace{H(\mathcal{K}, \mathcal{X})}_{=H(\mathcal{K})+H(\mathcal{X})} + \underbrace{H(\mathcal{Y} | \mathcal{K}, \mathcal{X})}_{=0}$$

- Daher folgt $H(\mathcal{Y} | \mathcal{X}) = H(\mathcal{K}) - H(\mathcal{K} | \mathcal{X}, \mathcal{Y})$ und somit $\log \alpha \geq H(\mathcal{K} | \mathcal{X}, \mathcal{Y}) - H(\mathcal{K})$
- Dies ist äquivalent zu $\alpha \geq 1/2^{H(\mathcal{K})-H(\mathcal{K}|\mathcal{X},\mathcal{Y})}$



Beispiel (Fortsetzung)

- Es gilt

$$H(\mathcal{K}) = \sum_k p(k) \log \frac{1}{p(k)} = 0,45 \cdot 1,152 + 0,3 \cdot 1,737 + 0,25 \cdot 2,0 = 1,54$$

- Um $H(\mathcal{K}|\mathcal{X}, \mathcal{Y})$ zu bestimmen, benötigen wir die bedingten Verteilungen $\mathcal{K}_{x,y}$ für alle Paare $(x, y) \in X \times Y$:

(x, y)	$(a, 0)$	$(a, 1)$	$(b, 0)$	$(b, 1)$	$(c, 0)$	$(c, 1)$	$(d, 0)$	$(d, 1)$
$p(1 x, y)$	$\frac{5}{14}$	0	1	0	$\frac{5}{11}$	0	0	$\frac{5}{11}$
$p(2 x, y)$	0	1	0	$\frac{2}{5}$	$\frac{6}{11}$	0	0	$\frac{6}{11}$
$p(3 x, y)$	$\frac{9}{14}$	0	0	$\frac{3}{5}$	0	1	1	0
$H(\mathcal{K} x, y)$	$\approx 0,94$	0	0	$\approx 0,97$	$\approx 0,99$	0	0	$\approx 0,99$
$p(x, y)$	0,07	0,03	0,05	0,15	0,165	0,135	0,18	0,22

Beispiel (Fortsetzung)

- Hierbei gilt $p(x, y) = p(x)p(y|x) = p(x)p(x \mapsto y)$
- Somit ist

$$H(\mathcal{K}|\mathcal{X}, \mathcal{Y}) = \sum_{x,y} p(x, y) H(\mathcal{K}|x, y) \approx 0,52$$

und wir erhalten die untere Schranke

$$\alpha \geq \frac{1}{2^{H(\mathcal{K}) - H(\mathcal{K}|\mathcal{X}, \mathcal{Y})}} \approx \frac{1}{2^{1,54 - 0,52}} = \frac{1}{2^{1,02}} \approx 0,493$$