

Vorlesungsskript

Kryptologie

Wintersemester 2022/23

Prof. Dr. Johannes Köbler
Humboldt-Universität zu Berlin
Lehrstuhl Komplexität und Kryptografie

5. Januar 2023

Inhaltsverzeichnis

1	Kryptografische Hashverfahren	1
1.1	Einführung	1
1.2	Schlüssellose Hashfunktionen (MDCs)	3
1.2.1	Vergleich von Sicherheitsanforderungen	4
1.2.2	Das Zufallsorakelmodell (ZOM)	5
1.2.3	Iterierte Hashfunktionen	8
1.2.4	Die Merkle-Damgaard-Konstruktion	9
1.2.5	Die MD4-Hashfunktion	10
1.2.6	Die MD5-Hashfunktion	11
1.2.7	Die SHA1-Hashfunktion	12
1.2.8	Die SHA2-Familie	13
1.2.9	Die Sponge-Konstruktion	16
1.2.10	Die SHA3-Familie	19
1.3	Nachrichten-Authentikationscodes (MACs)	21
1.3.1	Angriffe gegen symmetrische Hashfunktionen	21
1.3.2	Informationstheoretische Sicherheit von MACs	22
1.3.3	CBC-MACs	31
1.3.4	Kombination einer Hashfunktion mit einem MAC (HMAC)	32
2	Elliptische Kurven	35
2.1	Elliptische Kurven über den reellen Zahlen	35
2.2	Elliptische Kurven über endlichen Körpern	37
3	Digitale Signaturverfahren	40
3.1	Das RSA-Signaturverfahren	41
3.2	Das ElGamal-Signaturverfahren	42
3.3	Das Schnorr-Signaturverfahren	44
3.4	Der Digital Signature Algorithm (DSA)	45
3.5	ECDSA (Elliptic Curve DSA)	46
3.6	One-time Signatur (Lamport 1979)	47
3.7	Full Domain Hash (FDH) Signaturen	50

1 Kryptografische Hashverfahren

1.1 Einführung

Durch kryptographische Verfahren lassen sich unter anderem die folgenden **Schutzziele** realisieren.

- *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

Kryptografische Hashverfahren 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, sind die Nachrichten- und Teilnehmerauthentikation.

- „Nachrichtenauthentikation“ (message authentication)

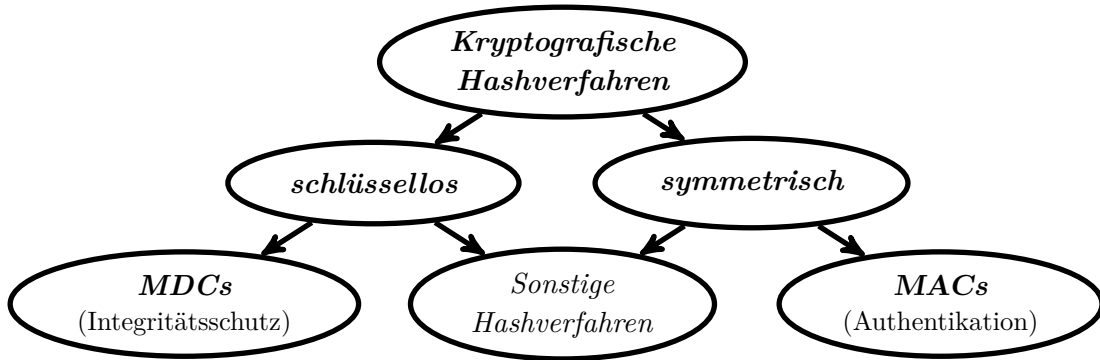


Abbildung 1.1: Eine grobe Einteilung von kryptografischen Hashverfahren.

- 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?

Klassifikation von Hashverfahren

Kryptografische Hashverfahren lassen sich grob danach klassifizieren, ob der Hashwert lediglich in Abhängigkeit vom Eingabetext berechnet wird oder zusätzlich von einem symmetrischen Schlüssel abhängt (siehe Abbildung 1.1).

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** bezeichnet (**M**anipulation **D**etection **C**ode [englisch] = Code zur Erkennung von Manipulationen). 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). Abbildung 1.2 zeigt eine typische Anwendung von MDCs.

Um die Integrität eines Datensatzes x sicherzustellen, der über einen ungesi-

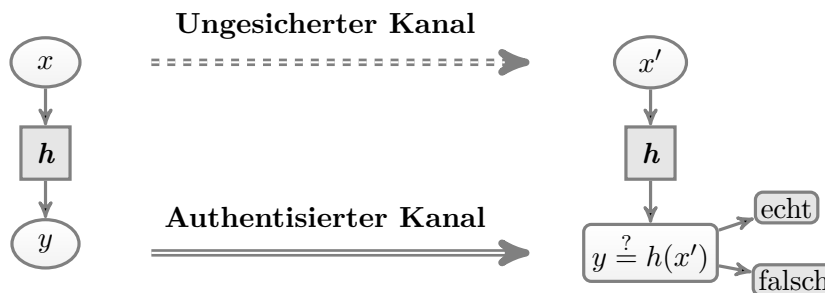


Abbildung 1.2: Einsatz eines MDC h zur Überprüfung der Integrität eines Datensatzes x .

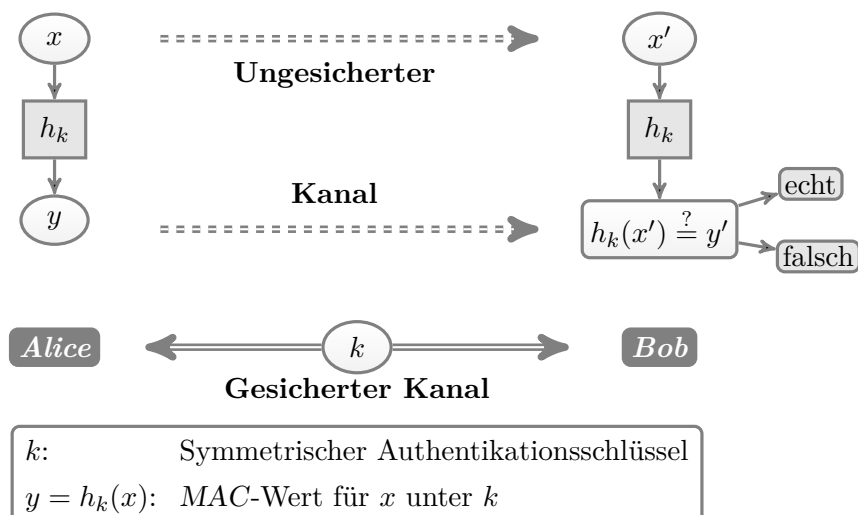


Abbildung 1.3: Verwendung eines MAC zur Nachrichtenauthentikation.

cherten Kanal gesendet (bzw. auf einem vor Manipulationen nicht sicheren Webserver abgelegt) wird, kann man wie folgt verfahren. Man sendet den MDC-Hashwert von x über einen authentisierten Kanal und prüft, ob der Datensatz nach der Übertragung noch denselben Hashwert liefert.

Kryptografische Hashverfahren mit symmetrischen Schlüsseln finden hauptsächlich bei der Authentifizierung von Nachrichten Verwendung. Diese werden daher auch als **MAC** (**m**essage **a**uthentication **c**ode [englisch] = Code zur Nachrichtenauthentifizierung) oder als **Authentifikationscode** 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. Abbildung 1.3 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.

Indem Alice ihre Nachricht x um den Hashwert $y = h_k(x)$ ergänzt, gibt sie Bob nicht nur die Möglichkeit, anhand von y die empfangene Nachricht auf Manipulationen zu überprüfen. Die Benutzung des geheimen Schlüssels k erlaubt zudem eine Überprüfung der Herkunft der Nachricht.

1.2 Schlüssellose Hashfunktionen (MDCs)

In diesem Abschnitt betrachten wir verschiedene Sicherheitsanforderungen an einzelne Hashfunktionen h . Dabei nehmen wir an, dass h öffentlich bekannt ist, d.h. h ist eine schlüssellose Hashfunktion (MDC).

Sei $h: X \rightarrow Y$ eine Hashfunktion. Ein Paar $(x, y) \in X \times Y$ heißt **gültig** für h , falls $h(x) = y$ ist. Ein Paar (x, x') mit $h(x) = h(x')$ 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*).

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 nicht möglich ist, falls der Hashwert y vorgegeben wird.

Problem P1: Bestimmung eines Urbilds

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

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

Falls es einen immensen Aufwand erfordert, für einen *vorgegebenen* 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.

Problem P2: Bestimmung eines zweiten Urbilds

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

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

Falls sich für einen *vorgegebenen* Text x nur mit großem Aufwand ein weiterer Text $x' \neq x$ mit dem gleichen Hashwert $h(x') = h(x)$ finden lässt, heißt h **schwach kollisionsresistent** (*weakly collision resistant* bzw. *second preimage resistant*). Diese Eigenschaft wird in der durch Abbildung 1.2 skizzierten Anwendung benötigt. Beim Versuch, eine digitale Signatur zu fälschen (siehe unten), sieht sich der Gegner dagegen mit folgender Problemstellung konfrontiert.

Problem P3: Bestimmung einer Kollision

Gegeben: Eine Hashfkt. $h: X \rightarrow Y$.

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

Falls sich dieses Problem nur mit einem immensen Aufwand lösen lässt, heißt h (**stark**) **kollisionsresistent** (*collision resistant*).

Obwohl die schwache Kollisionsresistenz eine gewisse Ähnlichkeit mit der Einweg-Eigenschaft aufweist, sind die beiden Eigenschaften im allgemeinen unvergleichbar. So muss eine schwach kollisionsresistente Funktion 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.

1.2.1 Vergleich von Sicherheitsanforderungen

In diesem Abschnitt zeigen wir, dass stark kollisionsresistente Hashfunktionen sowohl schwach kollisionsresistent als auch Einweghashfunktionen sind.

Satz 1. *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,*

```

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

```

Abbildung 1.4: Reduktion des Kollisionsproblems auf das Problem, 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 der in Abbildung 1.4 dargestellte Las-Vegas Algorithmus mit Wahrscheinlichkeit ε ein Kollisionspaar findet. \square

Als nächstes zeigen wir, wie sich das Kollisionsproblem auf das Urbildproblem reduzieren lässt.

Satz 2. 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 den in Abbildung 1.5 dargestellten Las-Vegas Algorithmus B .

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

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

\square

1.2.2 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 $h(x)$ auf 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 besteht für den Gegner die

```

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(?)

```

Abbildung 1.5: Reduktion des Kollisionsproblems auf das Urbildproblem

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(?)

```

Abbildung 1.6: Bestimmung eines Urbilds für einen Hashwert

einzigste Möglichkeit, Informationen über h zu erhalten, darin, sich für eine Reihe von Texten die zugehörigen Hashwerte zu besorgen (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 Gegner bereits die Hashwerte einer beliebigen Zahl von anderen Texten kennt.

Proposition 3. Sei $X_0 = \{x_1, \dots, x_k\}$ eine beliebige Menge von k verschiedenen Texten aus 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 im ZOM zu erhalten, betrachten wir den in Abbildung 1.6 dargestellten Algorithmus. Hier (und bei den beiden folgenden Algorithmen) gibt der Parameter q die Anzahl der Hashwertberechnungen (also die Anzahl der gestellten Orakelfragen an das Zufallsorakel h) an. Der Zeitaufwand der Algorithmen ist dabei proportional zu q .

Satz 4. *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 Proposition 3 sind diese Ereignisse stochastisch unabhängig und ihre Wahrscheinlichkeit ist $\Pr[E_i] = 1/m$ ($i = 1, \dots, q$). Also folgt

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

□

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(?)

```

Abbildung 1.7: Bestimmung eines 2. Urbilds für einen Hashwert

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(?)

```

Abbildung 1.8: Bestimmung eines Kollisionspaares

Der in Abbildung 1.7 dargestellte Algorithmus liefert uns eine obere Schranke für die Komplexität des Problems, ein zweites Urbild für $h(x)$ zu bestimmen. Die Erfolgswahrscheinlichkeit lässt sich vollkommen analog zum vorherigen Satz bestimmen.

Satz 5. *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.

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 ist deutlich zeiteffizienter zu realisieren. Wie der Name schon andeutet, basiert dieser Angriff auf dem sogenannten Geburtstagsparadoxon, welches in seiner einfachsten Form folgendes besagt.

Geburtstagsparadoxon: Bereits in einer Schulklasse mit 23 Schulkindern haben mit einer Wahrscheinlichkeit größer $1/2$ mindestens zwei Kinder am gleichen Tag Geburtstag.* Tatsächlich zeigt der nächste Satz, 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 mehr als einmal gezogen wird. Für $m = 365$ und $q = 23$ ergibt dies einen Wert von ungefähr 0,507.

Zur Kollisionsbestimmung verwenden wir den in Abbildung 1.8 dargestellten Algorithmus. 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 (herkömmliche) Hashtabelle der Größe q ein, so wird der Zeitaufwand für die Bearbeitung jedes einzelnen Textes x im wesentlichen durch die Berechnung von $h(x)$ bestimmt.

Satz 6. *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 $X_0 = \{x_1, \dots, x_q\}$. Für $i = 1, \dots, q$ bezeichne E_i das Ereignis

$$"h(x_i) \notin \{h(x_1), \dots, h(x_{i-1})\}."$$

*Da die Häufigkeiten der Geburtstage in Wirklichkeit nicht ganz gleichmäßig über das Jahr verteilt sind, ist die Wahrscheinlichkeit sogar noch etwas höher.

Dann beschreibt $E_1 \cap \dots \cap E_q$ das Ereignis “Collision(h, q) gibt ? aus” und für $i = 1, \dots, q$ gilt

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

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 $1 - x \approx e^{-x}$ folgt

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

Somit erhalten wir die Abschätzung

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

mit $c_\varepsilon = \sqrt{2\varepsilon}$. Diese Abschätzung ist nur für ε -Werte nahe Null hinreichend genau. Eine bessere Abschätzung ergibt sich aus der Approximation $\varepsilon \approx 1 - e^{-\frac{q^2}{2m}}$:

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

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

1.2.3 Iterierte Hashfunktionen

In diesem Abschnitt 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.

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 Hashwert $\hat{h}(x) = g(z_r)$, wobei $g: \{0, 1\}^m \rightarrow \{0, 1\}^l$ eine öffentlich bekannte Funktion ist. (Meist wird für g die Identität verwendet.)

Um $\hat{h}(x)$ zu berechnen, muss also die Kompressionsfunktion h genau r -mal aufgerufen werden. Wir formulieren nun eine für Preprocessing-Funktionen wünschenswerte Eigenschaft.

Definition 7. 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 (d.h. 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.

Satz 8. 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 $x \neq \tilde{x}$). Sei

$$y(x) = y_1y_2 \dots y_{k-1}y_k \text{ und } y(\tilde{x}) = \tilde{y}_1\tilde{y}_2 \dots \tilde{y}_{l-1}\tilde{y}_l \text{ mit } k \leq l.$$

Da y suffixfrei ist, muss ein Index $i \in \{1, \dots, k\}$ mit $y_i \neq \tilde{y}_{l-k+i}$ existieren. Weiter seien z_i ($i = 0, \dots, k$) und \tilde{z}_j ($j = 0, \dots, l$) die in der Processing-Phase berechneten Hashwerte. Da g injektiv ist, muss mit $g(z_k) = \hat{h}(x) = \hat{h}(\tilde{x}) = g(\tilde{z}_l)$ auch $z_k = \tilde{z}_l$ gelten. Sei i_{max} der größte Index $i \in \{1, \dots, k\}$ mit $z_{i-1}y_i \neq \tilde{z}_{l-k+i-1}\tilde{y}_{l-k+i}$. Dann bilden $z_{i_{max}-1}y_{i_{max}}$ und $\tilde{z}_{l-k+i_{max}-1}\tilde{y}_{l-k+i_{max}}$ wegen

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

ein Kollisionspaar für h . □

1.2.4 Die Merkle-Damgaard-Konstruktion

Merkle und Damgaard schlugen 1989 folgende konkrete Realisierung ihrer Konstruktion vor. Als Initialisierungsvektors 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.)

Für $x = \varepsilon$ sei $y(x) = 0^t$ und für $x \in \{0, 1\}^n$ mit $n > 0$ sei $k = \lceil \frac{n}{t-1} \rceil$ und $x = x_1x_2 \dots x_{k-1}x_k$ mit $|x_1| = |x_2| = \dots = |x_{k-1}| = t-1$ sowie $|x_k| = t-1-d$, wobei

$0 \leq d < t - 1$. Im Fall $k = 1$ ist dann $y(x) = 0x0^d1bin_{t-1}(d)$ und für $k > 1$ ist $y(x) = y_1 \cdots y_{k+1}$, wobei

$$y_i = \begin{cases} 0x_1, & i = 1, \\ 1x_i, & 2 \leq i < k, \\ 1x_k0^d, & i = k, \\ 1bin_{t-1}(d), & i = k + 1, \end{cases} \quad (1.1)$$

und $bin_{t-1}(d)$ die durch führende Nullen auf die Länge $t - 1$ aufgefüllte Binärdarstellung von d ist.

Satz 9. Die durch (1.1) definierte Preprocessing-Funktion y ist suffixfrei.

Beweis. Seien $x \neq \tilde{x}$ zwei Texte mit $|x| \leq |\tilde{x}|$. Wir müssen zeigen, dass $y(x) = y_1y_2 \cdots y_{k+1}$ kein Suffix von $y(\tilde{x}) = \tilde{y}_1\tilde{y}_2 \cdots \tilde{y}_{l+1}$ 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 $y_{k+1} \neq \tilde{y}_{l+1}$.
- 2. Fall:** $|x| = |\tilde{x}|$. In diesem Fall ist $k = l$. Wegen $x \neq \tilde{x}$ existiert ein Index $i \in \{1, \dots, k\}$ 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 $k < l$. Da $y(x)$ mit einer Null beginnt, aber das $(l - k + 1)$ -te Bit von $y(\tilde{x})$ eine Eins ist, kann $y(x)$ kein Suffix von $y(\tilde{x})$ sein. \square

Nun kommen wir zum 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 $k+1 = \lceil \frac{n}{t-1} \rceil + 1$ Aufrufe benötigt.

1.2.5 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. Die im Folgenden vorgestellten Hashfunktionen benutzen u.a. folgende Operationen auf Wörtern.

Operatoren auf $\{0, 1\}^{32}$	
$X \wedge Y$	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}
$X \rightarrow s$	Rechtsshift um s Stellen
$X \leftarrow s$	zirkulärer Linksshift um s Stellen

Während die Ganzzahl-Addition bei MD4 und MD5 in *little endian* Architektur (d.h. ein aus 4 Bytes $a_3a_2a_1a_0$, $0 \leq a_i \leq 255$ zusammengesetztes Wort repräsentiert die Zahl $a_02^{24} + a_12^{16} + a_22^8 + a_3$) ausgeführt wird, verwendet **SHA-1** eine *big endian* Architektur (d.h. $a_3a_2a_1a_0$, $0 \leq a_i \leq 255$ repräsentiert die Zahl $a_32^{24} + a_22^{16} + a_12^8 + a_0$). Der MD4-Algorithmus benutzt die folgenden Konstanten $y_j, z_j, s_j, j = 0, \dots, 47$

	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

und folgende Funktionen $f_j, j = 0, \dots, 47$

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

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

MD4(x)

```

1 input  $x \in \{0, 1\}^*, |x| = n$ 
2  $y := x10^k \mathit{bin}_{64}(n)$ ,  $k \in \{0, 1, \dots, 511\}$  mit  $n + 1 + k + 64 \equiv 0 \pmod{512}$ 
3  $(H_1, H_2, H_3, H_4) := (67452301, \mathit{efcdab89}, \mathit{98badcfe}, 10325476)$ 
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  $H_1H_2H_3H_4$ 

```

1.2.6 Die MD5-Hashfunktion

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 -Konstanten sind definiert als $y_j :=$ die ersten 32 Bit der Binärdarstellung von $\text{abs}(\sin(j + 1))$, $0 \leq j \leq 63$, und für z_j und s_j werden folgende Konstanten benutzt.

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

Für MD5 konnten in 2004 ebenfalls Kollisionspaare gefunden werden (für die Kompressionsfunktion von MD5 gelang dies bereits 1996).

MD5(x)

```

1 input  $x \in \{0, 1\}^*$ ,  $|x| = n$ 
2  $y := x10^k \mathbf{bin}_{64}(n)$ ,  $k \in \{0, 1, \dots, 511\}$  mit  $n + 1 + k + 64 \equiv 0 \pmod{512}$ 
3  $(H_1, H_2, H_3, H_4) := (67452301, \mathit{efcdab89}, \mathit{98badcfe}, 10325476)$ 
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      $(A, B, C, D) := (D, B + (A + f_j(B, C, D) + X[z_j] + y_j) \leftarrow s_j, B, C)$ 
10     $(H_1, H_2, H_3, H_4) := (H_1 + A, H_2 + B, H_3 + C, H_4 + D)$ 
11 output  $H_1 H_2 H_3 H_4$ 

```

1.2.7 Die SHA1-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 , $j = 0, \dots, 79$

	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 , $j = 0, \dots, 79$

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

SHA-1(x)

```

1 input  $x \in \{0, 1\}^*$ ,  $|x| = n$ 
2  $y := x10^k \mathbf{bin}_{64}(n)$ ,  $k \in \{0, 1, \dots, 511\}$  mit  $n + 1 + k + 64 \equiv 0 \pmod{512}$ 
3  $(H_0, H_1, H_2, H_3, H_4) := (67452301, efc dab89, 98badcfe, 10325476, c3d2e1f0)$ 
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      $X[t] := (X[t - 3] \oplus X[t - 8] \oplus X[t - 14] \oplus X[t - 16]) \leftrightarrow 1$ 
9      $(A, B, C, D, E) := (H_0, H_1, H_2, H_3, H_4)$ 
10    for  $j := 0$  to  $79$  do
11       $temp := (A \leftrightarrow 5) + f_j(B, C, D) + E + X[j] + K_j$ 
12       $(A, B, C, D, E) := (temp, A, B \leftrightarrow 30, C, D)$ 
13       $(H_0, H_1, H_2, H_3, H_4) := (H_0 + A, H_1 + B, H_2 + C, H_3 + D, H_4 + E)$ 
14 output  $H_0 H_1 H_2 H_3 H_4$ 

```

1.2.8 Die SHA2-Familie

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, 34b0cb5, 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 $2, \dots, 311$. SHA-256 arbeitet wie folgt.

SHA-256(x)

```

1 input  $x \in \{0, 1\}^*$ ,  $|x| = n$ 
2  $y := x10^k \mathbf{bin}_{64}(n)$ ,  $k \in \{0, 1, \dots, 511\}$  mit  $n + 1 + k + 64 \equiv 0 \pmod{512}$ 
3  $(H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7) := (6a09e667, bb67ae85, 3c6ef372, a54ff53a,$ 
4    $510e527f, 9b05688c, 1f83d9ab, 5be0cd19)$ 
5 sei  $y = M_1 \cdots M_r$ ,  $r = (n + 1 + k + 64)/512$ 
6 for  $i := 1$  to  $r$  do
7   sei  $M_i = X[0] \cdots X[15]$ 
8   for  $t := 16$  to  $63$  do
9      $s0 := (X[t - 15] \leftrightarrow 7) \oplus (X[t - 15] \leftrightarrow 18) \oplus (X[t - 15] \rightarrow 3)$ 
10     $s1 := (X[t - 2] \leftrightarrow 17) \oplus (X[t - 2] \leftrightarrow 19) \oplus (X[t - 2] \rightarrow 10)$ 
11     $X[t] := X[t - 16] + s0 + X[t - 7] + s1$ 
12     $(A, B, C, D, E, F, G, H) := (H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7)$ 
13    for  $j := 0$  to  $63$  do
14       $s0 := (A \leftrightarrow 2) \oplus (A \leftrightarrow 13) \oplus (A \leftrightarrow 22)$ 
15       $maj := (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$ 
16       $t2 := s0 + maj$ 
17       $s1 := (E \leftrightarrow 6) \oplus (E \leftrightarrow 11) \oplus (E \leftrightarrow 25)$ 
18       $ch := (E \wedge F) \oplus (\neg E \wedge G)$ 
19       $t1 := H + s1 + ch + K_j + X[j]$ 
20       $(A, B, C, D, E, F, G, H) := (t1 + t2, A, B, C, D + t1, E, F, G)$ 
21       $(H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7)$ 
22       $:= (H_0 + A, H_1 + B, H_2 + C, H_3 + D, H_4 + E, H_5 + F, H_6 + G, H_7 + H)$ 
23 output  $H_0 H_1 H_2 H_3 H_4 H_5 H_6 H_7$ 

```

Die Initialwerte von H_0, \dots, H_7 in den Zeilen 3 und 4 sind jeweils die ersten 32 Bit der binären Nachkommastellen der Wurzeln der Primzahlen 2, 3, 5, 7, 11, 13, 17, 19.

Kryptoanalyse von Hashfunktionen

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.

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.

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 ausgeschrieben 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.

1.2.9 Die Sponge-Konstruktion

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. Mittels der Funktion f wird eine Folge von **Zuständen** $s_i \in \{0, 1\}^b$ generiert, deren Länge b als **Weite** des Sponges bezeichnet wird. Die Zustände $s_i = z_i u_i$ werden in zwei Teilblöcke z_i und u_i der Länge r bzw. c zerlegt, 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 $c = |z_i|$ der Bits des inneren Zustands wird als **Kapazität** des Sponges bezeichnet und ist sein wichtigster Sicherheitsparameter. Die Anzahl $r = |u_i|$ der Bits des äußeren Zustands heißt **Bitrate**. Da $r + c = b$ gelten muss, lässt sich über die Wahl der Parameter r und c ein Trade-off zwischen der Effizienz und der Sicherheit des Sponges realisieren.

Die Funktion f bildet den Kern der Sponge-Konstruktion. Bevor f iteriert angewendet wird, um die Zustandsfolge s_i zu generieren, findet ein Preprocessing statt. Wir definieren nun mögliche Anforderungen an die Preprocessing-Funktion y .

Definition 10. Sei y eine Funktion der Form $y : \{0, 1\}^* \rightarrow \{0, 1\}^*$. y heißt **Paddingfunktion**, falls für alle $x \in \{0, 1\}^*$ ein $z \in \{0, 1\}^*$ mit $y(x) = xz$ existiert. Eine solche Funktion heißt **Paddingfunktion für eine Bitrate** $r \geq 1$, falls $|y(x)| \equiv_r 0$ für alle $x \in \{0, 1\}^*$ gilt. Die Funktion y heißt **sponge-konform** für Bitrate $r \geq 1$, falls gilt:

- $\forall i = 0, \dots, r-1 \exists z \in \{0, 1\}^* \forall x \in \{0, 1\}^*$ mit $|x| \equiv_r i : y(x) = xz$
- $\forall k \geq 0 \forall x \neq x' : y(x) \neq y(x')0^{kr}$.

Es ist leicht zu sehen, dass die Funktion $\text{pad}10^*1_r : \{0, 1\}^* \rightarrow \{0, 1\}^*$ definiert durch

$$\text{pad}10^*1_r(x) = x10^d1 \text{ mit } d = \min\{i \geq 0 \mid |x| + 2 + i \equiv_r 0\}$$

eine sponge-konforme Paddingfunktion für die Bitrate r ist. Ohne die abschließende 1 wäre dies nicht der Fall.

Definition 11. 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 $i = 1, \dots, k$. Sei $s_0 = 0^b$ und für $i \geq 1$ seien $s_i = z_i u_i \in \{0, 1\}^b$ mit $z_i \in \{0, 1\}^r, u_i \in \{0, 1\}^c$ mit

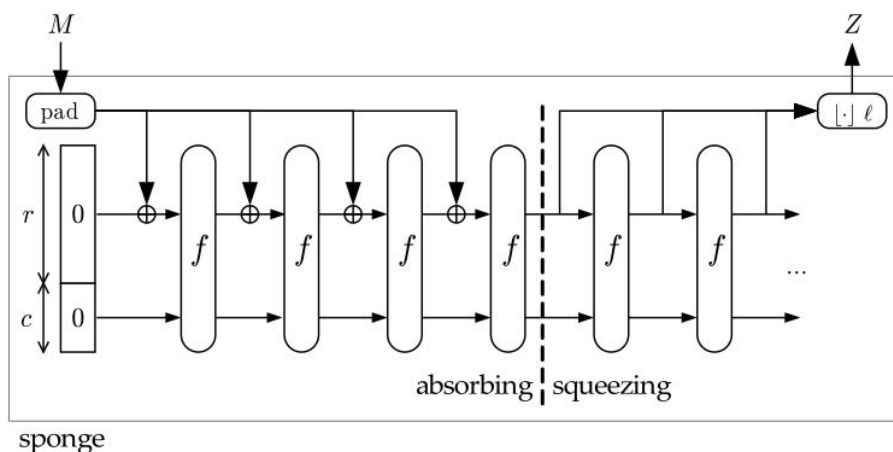
$$s_i = \begin{cases} f(s_{i-1} \oplus y_i 0^c) & 1 \leq i \leq k & \text{(Absorptionsphase)} \\ f(s_{i-1}) & i > k & \text{(Squeezing-Phase)} \end{cases}$$

Dann ist $\text{Sponge}_{f,y,r} : \mathbb{N} \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ die Funktion mit

$$\text{Sponge}_{f,y,r}(l, x) = z_k \dots z_{k+m-1} z'_{k+m},$$

wobei $m = \lfloor \frac{l}{r} \rfloor$ und z'_{k+m} das Präfix von z_{k+m} der Länge $l - mr$ ist. Für die Analyse definieren wir noch die beiden Funktionen

$$\text{Absorb}_{f,r}(y_1 \dots y_k) = s_k \text{ und } \text{Squeeze}_{f,r}(l, s_k) = z_k \dots z_{k+m-1} z'_{k+m}.$$



Den Aufwand, ein Paar $x \neq x'$ mit $Sponge_{f,y,r}(l, x) = Sponge_{f,y,r}(l, x')$ zu finden, können wir nach oben durch den Aufwand abschätzen, ein Paar $y \neq y'$ in $\cup_{k \geq 1} \{0, 1\}^{kr}$ mit $y = y_1 \dots y_k$, $y' = y'_1 \dots y'_{k'}$ und $Absorb_{f,r}(y) = s_k = s'_{k'} = Absorb_{f,r}(y')$ zu finden. Es genügt sogar, dass die inneren Zustände u_k von $s_k = z_k u_k$ und $u'_{k'}$ von $s'_{k'} = z'_{k'} u'_{k'}$ gleich sind, d.h. $y \neq y'$ ist ein **inneres Kollisionspaar** für die Funktion $Absorb_{f,r}$.

Setzen wir nämlich y_{k+1} und $y'_{k'+1}$ auf die äußeren Zustände z_k von s_k und $z'_{k'}$ von $s'_{k'}$, so folgt für die Texte $x = yy_{k+1}$ und $x' = y'y'_{k'+1}$:

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

Falls das Suffix z von $y(x) = xz$ nur von $|x| \bmod r$ abhängt, gilt wegen $|x| \equiv_r |x'| \equiv_r 0$ dann auch $y(x') = x'z$ und somit folgt

$$Absorb_{f,r}(y(x)) = Absorb_{f,r}(xz) = Absorb_{f,r}(x'z) = Absorb_{f,r}(y(x'))$$

was $Sponge_{f,y,r}(l, x) = Sponge_{f,y,r}(l, x')$ für alle $l \geq 1$ impliziert.

Um eine innere Kollision $y \neq y'$ zu finden, hilft es, sich die 2^c inneren Zustände $u \in \{0, 1\}^c$ als Knoten eines gerichteten Multigraphen G vorzustellen, der für jeden Zustand $s = wv$ eine Kante $v \xrightarrow{w,z} u$ von v nach u mit dem Label w, z enthält, wobei $f(wv) = zu$ ist. Dann lässt sich jedem String $y = y_1 \dots y_k \in \{0, 1\}^{kr}$ ein Pfad

$$P_y : u_0 \xrightarrow{y_1, z_1} u_1 \xrightarrow{z_1 \oplus y_2, z_2} u_2 \xrightarrow{z_2 \oplus y_3, z_3} u_3 \dots \xrightarrow{z_{k-1} \oplus y_k, z_k} u_k$$

der Länge k in G mit dem Startknoten $u_0 = 0^c$ zuordnen. Umgekehrt ex. zu jedem Pfad $P : u_0 \xrightarrow{w_1, z_1} u_1 \xrightarrow{w_2, z_2} u_2 \xrightarrow{w_3, z_3} \dots \xrightarrow{w_k, z_k} u_k$ der Länge k in G mit Startknoten $u_0 = 0^c$ genau ein y mit $P_y = P$, nämlich $y = y_1 \dots y_k$ mit $y_1 = w_1$ und $y_i = z_{i-1} \oplus w_i$ für $i = 2, \dots, k$. Daher ist $y \neq y'$ genau dann eine innere Kollision für $Absorb_{f,r}$, wenn P_y und $P_{y'}$ zwei verschiedene Pfade vom Startknoten 0^c zum gleichen Zielknoten $u_k = u'_{k'}$ sind.

Ist f eine Zufallsfunktion (ZOM), so lassen bereits bekannte Werte von f keine Rückschlüsse auf die Werte von f für andere Argumente zu. Anders als beim ZOM für eine Hashfunktion kann es sich dennoch für den Gegner lohnen, die Argumente von f adaptiv nach einer Strategie \mathcal{S} zu wählen, um möglichst schnell eine innere Kollision $y \neq y'$ mit bestimmten Eigenschaften zu finden. Der Algorithmus **InnerCollision** auf der nächsten Folie fasst dieses Vorgehen zusammen. Die benutzte Strategie \mathcal{S} wird hierbei in Form einer Routine als Teil der Eingabe übergeben und der Parameter q beschränkt die Anzahl der Aufrufe der Transformation f .

Prozedur $\text{InnerCollision}(f, r, q, \mathcal{S})$ mit $f : \{0, 1\}^b \rightarrow \{0, 1\}^b$

```

1   $c := b - r$ 
2  initialisiere den Multi-Digraphen  $G = (V, E) := (\{0, 1\}^c, \emptyset)$ 
3  for  $i := 1$  to  $q$  do
4    wähle  $v \in V$  und  $w \in \{0, 1\}^r$  nach Strategie  $\mathcal{S}$ 
5     $E := E \cup \{v \xrightarrow{w, z} u\}$ , wobei  $f(wv) = zu$  ist
6  if  $\exists$  2 Pfade  $0^c \xrightarrow{w_1, z_1} u_1 \xrightarrow{w_2, z_2} u_2 \xrightarrow{w_3, z_3} \dots \xrightarrow{w_k, z_k} u_k$  und
7     $0^c \xrightarrow{w'_1, z'_1} u'_1 \xrightarrow{w'_2, z'_2} u'_2 \xrightarrow{w'_3, z'_3} \dots \xrightarrow{w'_{k'}, z'_{k'}} u'_{k'}$  mit  $u_k = u'_{k'}$  then
8     $y := y_1 \dots y_k$  mit  $y_1 = w_1$  und  $y_i = z_{i-1} \oplus w_i$  für  $i = 2, \dots, k$ 
9     $y' := y'_1 \dots y'_{k'}$  mit  $y'_1 = w'_1$  und  $y'_i = z'_{i-1} \oplus w'_i$  für  $i = 2, \dots, k'$ 
10   return( $y, y'$ )
11 else
12   return(?)
```

Abbildung 1.9: Bestimmung eines inneren Kollisionspaares

Satz 12. 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') für die Funktion $\text{Absorb}_{f,r}$ aus. Wählt \mathcal{S} nur von 0^c aus erreichbare Knoten v und kein Argument wv mehrmals, so ist die Erfolgswahrscheinlichkeit exakt ε .

Beweis. Sei E_i das Ereignis “ G enthält nach dem i -ten Durchlauf der for-Schleife zwei verschiedene Pfade von 0^c zu einem Knoten u .” 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

$$\Pr[E_i | \bar{E}_1 \cap \dots \cap \bar{E}_{i-1}] \leq \frac{i}{2^c}$$

Wählt \mathcal{S} nur erreichbare Knoten v und kein Argument wv mehrfach, so impliziert das Ereignis $\bar{E}_1 \cap \dots \cap \bar{E}_{i-1}$, dass nach dem i -ten Durchlauf **genau** i Knoten erreichbar sind (sonst gäbe es bereits zwei Pfade von 0^c zu einem Knoten in G) und es gilt sogar Gleichheit. Daher ist die Erfolgswahrscheinlichkeit von $\text{InnerCollision}(f, r, q, \mathcal{S})$

$$\begin{aligned} 1 - \Pr[\bar{E}_1 \cap \dots \cap \bar{E}_q] &= 1 - \Pr[\bar{E}_1] \Pr[\bar{E}_2 | \bar{E}_1] \dots \Pr[\bar{E}_q | \bar{E}_1 \cap \dots \cap \bar{E}_{q-1}] \\ &\leq 1 - \left(1 - \frac{1}{2^c}\right) \left(1 - \frac{2}{2^c}\right) \dots \left(1 - \frac{q}{2^c}\right) = \varepsilon \end{aligned} \quad \square$$

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 $c_\varepsilon = \sqrt{2\varepsilon}$. Da im Fall $l < c$ ein Geburtstagsangriff gegen die Funktion $x \mapsto \text{Sponge}_{f,y,r}(l, x)$ effizienter ist, erhalten wir die obere Schranke $\min(2^{c/2}, 2^{l/2})$ für den Aufwand eines mit Wahrscheinlichkeit $1/2$ erfolgreichen Angriffs gegen die Funktion $\text{Sponge}_{f,y,r}$ im ZOM.

1.2.10 Die SHA3-Familie

Im Oktober 2012 wurde der Hash-Algorithmus Keccak als Gewinner des vom NIST ausgeschriebenen Wettbewerbs für den SHA3-Algorithmus ausgewählt und am 5. August 2015 als Alternative zu SHA-2 standardisiert. 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, SHA-0 und SHA-1, die einen ähnlichen Aufbau wie SHA-2 haben, eine auf einem vollkommen anderen Entwurfsprinzip basierende Alternative zur Verfügung zu stellen.

Um geeignete Kandidaten für SHA-3 zu finden, veranstaltete das NIST ähnlich wie beim AES von Nov. 2007 bis Okt. 2012 einen Wettbewerb. Die Minimalanforderungen an die Hashfunktionen waren, dass sie Nachrichten bis zu einer Obergrenze von mindestens $2^{64} - 1$ Bit hashen und mindestens die Hash-Längen 224, 256, 384 und 512 Bit unterstützen. Von den eingereichten 64 Hashfunktionen erfüllten 51 die Teilnahmebedingungen und wurden für Runde 1 akzeptiert. In Runde 1 wurde die Sicherheit und Performanz dieser 51 Kandidaten von Kryptologen aus aller Welt in einem offenen Bewertungsprozess analysiert und 14 der 51 Kandidaten erreichten Runde 2.

Nach jeder Runde konnten Veränderungen an den Algorithmen vorgenommen werden, um auf die Ergebnisse der Analysen zu reagieren. Im Fall von Keccak wurde die Rundenzahl der Permutationsfunktion von 18 auf 24 erhöht, um eine größere Sicherheitsreserve zu erreichen. Zudem wurde die Bitrate bei einigen Varianten vergrößert, um die Effizienz zu verbessern.

Im Dezember 2010 wurden die fünf Finalisten BLAKE, Grøstl, JH, Keccak und Skein bekanntgegeben. Der Gewinner Keccak wies eine deutlich bessere Hardware-Performanz als seine Konkurrenten auf. Dagegen hatten BLAKE und Skein die beste Software-Performanz und waren in den meisten Fällen schneller als SHA-2. Keccak erlaubt für die Sponge-Weite die Wahl folgender 7 Werte $b_w \in \{25 \cdot 2^w \mid w = 0, \dots, 6\} = \{25, 50, 100, 200, 400, 800, 1600\}$. Entsprechend stehen 7 Permutationen $f_w : \{0, 1\}^{b_w} \mapsto \{0, 1\}^{b_w}$, $w = 0, \dots, 6$ zur Verfügung. Im SHA3-Standard ist $w = 6$, also $b = b_6 = 1600$ und SHA-3 stellt für $l = 224, 256, 384, 512$ jeweils eine Hashfunktion $\text{SHA3-}l$ mit der festen Hash-Länge l und der Kapazität $c = 2l$ zur Verfügung. Zudem enthält SHA-3 die Hashfunktionen SHAKE128 und SHAKE256 mit flexibler Hash-Länge und den Kapazitäten $c = 256$ bzw. $c = 512$.

Die SHA3-Familie besteht aus folgenden Hashfunktionen, die sich in der Kombination von Bitrate r und Ausgabelänge l unterscheiden:

Name	Hashlänge l	Rate r	Kap. c	Padding
SHA3-224	224	1152	448	0110*1
SHA3-256	256	1088	512	- "-
SHA3-384	384	832	768	- "-
SHA3-512	512	576	1024	- "-
SHAKE128	variabel	1344	256	111110*1
SHAKE256	- "-	1088	512	- "-

Für die Hash-Längen $l \in \{224, 256, 384, 512\}$ gilt also

$$\text{SHA3-}l(x) = \text{Sponge}_{f_6, \text{pad}10^*1_r, r}(l, x01) \text{ mit } c = 2l \text{ und } r = 1600 - 2l$$

Das zusätzliche Padding 01 soll dabei SHA3- l von anderen Anwendungen von Keccak mit denselben Werten b, l, r unterscheiden.

Wir betrachten nun die Funktion $f = f_w$ für $w = 6$ im Detail. Die Funktion $f : \{0, 1\}^{5 \times 5 \times 64} \rightarrow \{0, 1\}^{5 \times 5 \times 64}$ bildet einen $(5 \times 5 \times 64)$ -Block X auf einen ebensolchen Block $X' = f(X)$ ab. Hierzu wird $12 + 2w = 24$ -mal eine Rundenfunktion

$$f^r : \{0, 1\}^{5 \times 5 \times 64} \rightarrow \{0, 1\}^{5 \times 5 \times 64}, \quad r = 0, \dots, 23$$

aufgerufen, die unter Verwendung einer Rundenkonstanten RC_r den Block $X_{r+1} = f^r(X_r)$ für $r = 0, \dots, 23$ berechnet, wobei $X_0 = X$ und $X' = X_{24}$ ist.

Es gilt

$$f^r(X_r) = \iota_r(\chi(\pi(\rho(\theta(X_r))))),$$

wobei θ, ρ, π, χ und ι_r Permutationen auf der Menge $\{0, 1\}^{5 \times 5 \times 64}$ sind.

Zur Beschreibung der Permutationen θ, ρ, π, χ und ι_r bezeichnen wir die einzelnen Bits eines Blocks $Z \in \{0, 1\}^{5 \times 5 \times 64}$ mit $Z[x, y, z]$ für $x, y = 0, \dots, 4$ und $z = 0, \dots, 63$. Die Funktion θ ist so gewählt, dass sich $\theta^{-1}(Z)$ an möglichst vielen Bits ändert, falls eines in Z geflippt wird. Hierzu führt θ auf den **Wörtern** $Z[x, y] = Z[x, y, 0] \dots Z[x, y, 63]$ des Zustandsvektors Z folgende Operationen aus:

- $S(x) := Z[x, 0] \oplus \dots \oplus Z[x, 4]$
- $R(x) := S(x) \leftrightarrow 1$
- $Z[x, y] := Z[x, y] \oplus S(x-1) \oplus R(x+1),$

wobei die Indexoperationen modulo 5 erfolgen. Die Transposition ρ rotiert die Wörter von Z wie folgt:

- $Z[x, y] := Z[x, y] \leftrightarrow w_{x,y} \bmod 64,$

wobei $w_{0,0} = 0$ und $w_{x,y} = (t+1)(t+2)/2$ für $(x, y) = (0, 1) \begin{pmatrix} 3 & 1 \\ 2 & 0 \end{pmatrix}^t$ und $t = 0, \dots, 23$ gilt.

Die Transposition π permutiert die Wörter von Z wie folgt:

- $Z[3x+2y, x] := Z[x, y]$

Die Funktion χ ist die einzige nichtlineare Funktion und operiert innerhalb der 5-Bit-Spalten $Z[x, z]$ des Zustandsvektors, wobei jedes Bit nur von 2 anderen in dieser Spalte abhängt:

- $Z[x, y] := Z[x, y] \oplus (\neg Z[x, y+1] \wedge Z[x, y+2])$

Schlussendlich setzt ι_r das Wort $Z[0, 0]$ auf das Wort

- $Z[0, 0] := Z[0, 0] \oplus RC_r$,

wobei das Bit an Position $2^m - 1$ in RC_r für $m = 0, \dots, 6$ das Bit an Position $7r + m$ eines LFSR mit dem erzeugenden Polynom $x^8 + x^6 + x^5 + x^4 + 1$ ist und alle übrigen Bits in RC_r Null sind.

1.3 Nachrichten-Authentikationscodes (MACs)

Definition 13. 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**.

Damit ein geheimer Schlüssel k für die Authentifizierung mehrerer Nachrichten benutzt werden kann, ohne dass dies einem potentiellen Gegner 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 einer berechnungsresistenten Hashfunktion ist es einem Gegner nicht möglich, an Bob eine Nachricht x zu schicken, die Bob als von Alice stammend anerkennt.

Verwendung eines MAC zur Versiegelung von Software

Mithilfe einer berechnungsresistenten Hashfunktion 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.

1.3.1 Angriffe gegen symmetrische Hashfunktionen

Ein Angriff gegen einen MAC hat die unbefugte Berechnung von MAC-Werten zum Ziel. Das heißt, der Gegner 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 Gegner kennt nur den benutzten MAC und versucht ein Paar (x, y) mit $h_k(x) = y$ zu generieren, wobei k der (dem Gegner unbekannte) Schlüssel ist.

Substitution

Der Gegner 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 Gegner 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 Gegner kann die Texte x_i selbst wählen.

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

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

Auch wenn die Anwender den Schlüssel nach jeder MAC-Berechnung wechseln, sollte \mathcal{H} zumindest einem **Substitutionsangriff** widerstehen, da der Gegner sonst ein Paar (x, y) abfangen und durch ein anderes ersetzen könnte.

1.3.2 Informationstheoretische Sicherheit von MACs

Modell: Schlüssel k und Nachrichten x werden unabhängig gemäß einer Wahrscheinlichkeitsverteilung $p(k, x) = p(k)p(x)$ generiert, welche dem Gegner bekannt ist. Wir nehmen o.B.d.A. an, dass $p(x) > 0$ und $p(k) > 0$ für alle $x \in X$ und $k \in K$ gilt.

Erfolgswahrscheinlichkeit für Impersonation

Sei α die Wahrscheinlichkeit, mit der sich ein Gegner bei optimaler Strategie als Alice ausgeben kann, ohne dass Bob dies bemerkt.

Für ein Paar $(x, y) \in X \times Y$ sei $\alpha(x, y)$ die Wahrscheinlichkeit, dass dem Gegner mit der Wahl des Paares (x, y) eine Impersonation gelingt. Dann ist $\alpha(x, y) = p(y|x) = p(x \mapsto y)$ die Wahrscheinlichkeit, dass der zufällig gewählte Schlüssel den Text x auf den MAC-Wert y abbildet:

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

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

$$\alpha(x) = \max\{\alpha(x, y) \mid y \in Y\}$$

die maximale Wahrscheinlichkeit, mit der eine Impersonation gelingt, falls der Text x gewählt wird. Also ist $\alpha = \max\{\alpha(x) \mid x \in X\}$.

Beispiel 14. 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
		a	b	c	d
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. Dann hat der Gegner 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 15. Für alle $x \in X$ ist $\alpha(x) \geq \frac{1}{m}$ und daher gilt $\alpha \geq \frac{1}{m}$.

Beweis. Sei $x \in X$ beliebig. Dann 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 16. Wie der Beweis zeigt, gilt $\alpha = \frac{1}{m}$ genau dann, wenn für alle Paare $(x, y) \in X \times Y$ gilt,

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

D.h. bei Gleichverteilung der Schlüssel muss in jeder Spalte der Authentikationsmatrix 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.

Das folgende Lemma benötigen wir für den Beweis des nächsten Satzes.

Lemma 17. 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 18. Für jeden MAC (X, Y, K, H) gilt:

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

Hierbei sind $\mathcal{X}, \mathcal{Y}, \mathcal{K}$ 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.

Beweis. Da $\alpha = \max_{x,y} \alpha(x, y)$ ist, folgt $E(\alpha(\mathcal{X}, \mathcal{Y})) = \sum_{x,y} p(x, y) \alpha(x, y) \leq \alpha$, wobei $E(\alpha(\mathcal{X}, \mathcal{Y}))$ die Erfolgswahrscheinlichkeit eines (probabilistischen) Gegners ist, der das Paar (x, y) gemäß der Verteilung $(\mathcal{X}, \mathcal{Y})$ wählt. Somit folgt unter Anwendung von Lemma 17,

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

Wegen

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

und

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

gilt zudem $H(\mathcal{Y}|\mathcal{X}) = H(\mathcal{K}) - H(\mathcal{K}|\mathcal{X}, \mathcal{Y})$ und somit $\log \alpha \geq H(\mathcal{K}|\mathcal{X}, \mathcal{Y}) - H(\mathcal{K})$. \square

Beispiel 19 (Fortsetzung von Beispiel 14). 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 gemeinsame Verteilung von \mathcal{X}, \mathcal{Y} sowie die bedingten Verteilungen $\mathcal{K}_{x,y}$ für alle Paare $(x, y) \in X \times Y$.

(x, y)	$(a, 0)$	$(a, 1)$	$(b, 0)$	$(b, 1)$	$(c, 0)$	$(c, 1)$	$(d, 0)$	$(d, 1)$
$p(x, y)$	0,07	0,03	0,05	0,15	0,165	0,135	0,18	0,22
$p(1 x, y)$	$\frac{5}{14}$	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	$\approx 0,99$

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

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

\triangleleft

Erfolgswahrscheinlichkeit für Substitution

Bezeichne β die Wahrscheinlichkeit, mit der ein Gegner bei optimaler Strategie eine von Alice gesendete Nachricht x durch eine andere Nachricht x' ersetzen kann, ohne dass Bob dies bemerkt. Dabei gehen wir davon aus, dass der Gegner keinen Einfluss auf die Wahl der von Alice gesendeten Nachricht x hat.

Falls der Gegner ein von Alice gesendetes Paar (x, y) durch das Paar (x', y') ersetzt, ist seine Erfolgswahrscheinlichkeit gleich der bedingten Wahrscheinlichkeit

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

dass ein zufällig gewählter Schlüssel k den Text x' auf y' abbildet, wenn bereits bekannt ist, dass $h_k(x) = y$ ist. Falls Alice also das Paar (x, y) sendet, so ist die maximale Erfolgswahrscheinlichkeit des Gegners

$$\beta(x, y) := \max_{x' \neq x, y'} p(x' \mapsto y' | x \mapsto y).$$

Man beachte, dass $\beta(x, y)$ nur im Fall $p(x, y) > 0$ definiert ist. Da der Gegner keinen Einfluss auf die Wahl von (x, y) hat, ist β gleich dem Erwartungswert von $\beta(x, y)$ unter der Verteilung $p(x, y)$, mit der Alice diese Paare generiert. Somit ergibt sich β zu

$$\beta = E(\beta(\mathcal{X}, \mathcal{Y})) = \sum_{x \in X, y \in Y} p(x, y) \beta(x, y).$$

Wegen $p(x, y) = p(x)p(x \mapsto y)$ können wir β unter Verwendung der Funktion

$$\beta'(x, y) = \beta(x, y)p(x \mapsto y) = \max_{x' \neq x, y'} p(x' \mapsto y', x \mapsto y)$$

auch einfacher mittels der Formel $\beta = \sum_{x \in X} p(x) \sum_{y \in Y} \beta'(x, y)$ berechnen.

Beispiel 20 (Fortsetzung von Beispiel 14).

(x, y)	$p(x' \mapsto y', x \mapsto y)$								$\beta'(x, y)$	$p(x \mapsto y)$	$\beta(x, y)$
	$(a, 0)$	$(a, 1)$	$(b, 0)$	$(b, 1)$	$(c, 0)$	$(c, 1)$	$(d, 0)$	$(d, 1)$			
$(a, 0)$			0,25	0,45	0,25	0,45	0,45	0,25	0,45	0,7	0,643
$(a, 1)$			0	0,3	0,3	0	0	0,3	0,3	0,3	1
$(b, 0)$	0,25	0			0,25	0	0	0,25	0,25	0,25	1
$(b, 1)$	0,45	0,3			0,3	0,45	0,45	0,3	0,45	0,75	0,6
$(c, 0)$	0,25	0,3	0,25	0,3			0	0,55	0,55	0,55	1
$(c, 1)$	0,45	0	0	0,45			0,45	0	0,45	0,45	1
$(d, 0)$	0,45	0	0	0,45	0	0,45			0,45	0,45	1
$(d, 1)$	0,25	0,3	0,25	0,3	0,55	0			0,55	0,55	1

Die optimalen Wahlmöglichkeiten des Gegners, ein Paar (x, y) durch ein anderes Paar (x', y') zu ersetzen, sind in der Tabelle fett gedruckt. Für β erhalten wir somit den Wert

$$\begin{aligned} \beta &= \sum_{x \in X} p(x) \sum_{y \in Y} \beta'(x, y) \\ &= 0,1(0,45 + 0,3) + 0,2(0,25 + 0,45) + 0,3(0,55 + 0,45) + 0,4(0,45 + 0,55) \\ &= 0,915. \end{aligned}$$

Als nächstes zeigen wir für β die gleiche untere Schranke wie für α .

Satz 21. Für alle $(x, y) \in X \times Y$ mit $p(x, y) > 0$ ist $\beta(x, y) \geq \frac{1}{m}$ und daher gilt $\beta \geq \frac{1}{m}$.

Beweis. Sei $(x, y) \in X \times Y$ ein Paar mit $p(x, y) > 0$. Dann gilt für beliebige $x' \in X - \{x\}$,

$$\sum_{y' \in Y} p(x' \mapsto y' | x \mapsto y) = \frac{\sum_{y' \in Y} \sum_{k \in K(x', y'; x, y)} p(k)}{\sum_{k \in K(x, y)} p(k)} = 1.$$

Somit existiert ein $y' \in Y$ mit $p(x' \mapsto y' | x \mapsto y) \geq \frac{1}{m}$ und dies impliziert

$$\beta(x, y) = \max_{x' \neq x, y'} p(x' \mapsto y' | x \mapsto y) \geq \frac{1}{m}.$$

Folglich ist

$$\beta = \sum_{x \in X, y \in Y} p(x, y) \beta(x, y) \geq \frac{1}{m} \sum_{x \in X, y \in Y} p(x, y) = \frac{1}{m}.$$

□

Beispiel 22. Sei $X = Y = \{0, 1, 2\} = \mathbb{Z}_3$ und sei $K = \mathbb{Z}_3 \times \mathbb{Z}_3$. Für $k = (a, b) \in K$ und $x \in X$ sei

$$h_k(x) = ax + b \pmod{3}.$$

Die zugehörige **Authentikationsmatrix** ist

	0	1	2
(0, 0)	0	0	0
(0, 1)	1	1	1
(0, 2)	2	2	2
(1, 0)	0	1	2
(1, 1)	1	2	0
(1, 2)	2	0	1
(2, 0)	0	2	1
(2, 1)	1	0	2
(2, 2)	2	1	0

Wir nehmen an, dass der Schlüssel unter Gleichverteilung gewählt wird. Ersetzt der Gegner ein Paar (x, y) durch ein Paar (x', y') mit $x' \neq x$, so wird dieses Paar von genau einem der 3 infrage kommenden Schlüssel akzeptiert. Dies liegt daran, dass in je 2 Spalten der Authentikationsmatrix jedes MAC-Wertepaar genau einmal vorkommt. Folglich ist $p(x' \mapsto y' | x \mapsto y) = 1/3$ und somit hat β den optimalen Wert $\beta = 1/3$. ◁

Lemma 23. Sei (X, Y, K, H) ein MAC mit $\beta = \frac{1}{m}$. Dann gilt

$$p(x' \mapsto y' | x \mapsto y) = 1/m$$

für alle Doppelpaare (x, y, x', y') mit $x \neq x'$.

Beweis. Wir zeigen zuerst, dass die Behauptung unter der Voraussetzung $p(x \mapsto y) > 0$ gilt. Wäre nämlich

$$p(x' \mapsto y' | x \mapsto y) > 1/m,$$

dann wäre auch

$$\beta(x, y) = \max_{x' \neq x, y'} p(x' \mapsto y' | x \mapsto y) > 1/m.$$

Da für alle Paare (u, v) mit $p(u \mapsto v) > 0$ nach Satz 21 die Ungleichung $\beta(u, v) \geq 1/m$ gilt und zudem $p(x, y) = p(x)p(x \mapsto y) > 0$ ist, folgt hieraus

$$\beta = \sum_{x \in X, y \in Y} p(x, y)\beta(x, y) > 1/m,$$

was im Widerspruch zur Voraussetzung des Satzes steht. Ist andererseits

$$p(x' \mapsto y' | x \mapsto y) < 1/m,$$

muss wegen

$$\sum_{y'' \in Y} p(x' \mapsto y'' | x \mapsto y) = 1$$

auch ein MAC-Wert y'' mit $p(x' \mapsto y'' | x \mapsto y) > 1/m$ existieren, woraus sich wie bereits gezeigt ein Widerspruch ergibt.

Es bleibt zu zeigen, dass $p(x \mapsto y) > 0$ für alle Paare (x, y) gilt. Wäre $p(x \mapsto y) = 0$, so würde für ein beliebiges Paar (u, v) mit $p(u \mapsto v) > 0$ auch $p(x \mapsto y | u \mapsto v) = 0$ sein. Wie bereits gezeigt, steht dies jedoch im Widerspruch zur Voraussetzung $\beta = 1/m$. \square

Satz 24. Ein MAC (X, Y, K, H) erfüllt $\beta = \frac{1}{m}$ genau dann, wenn

$$p(x \mapsto y, x' \mapsto y') = 1/m^2$$

für alle Doppelpaare (x, y, x', y') mit $x \neq x'$ gilt.

Beweis. Sei (X, Y, K, H) ein MAC mit $\beta = \frac{1}{m}$. Nach obigem Lemma impliziert dies, dass

$$p(x' \mapsto y' | x \mapsto y) = 1/m$$

für alle Doppelpaare (x, y, x', y') mit $x \neq x'$ gilt. Dies impliziert nun

$$p(x' \mapsto y') = \sum_y p(x \mapsto y)p(x' \mapsto y' | x \mapsto y) = 1/m$$

und daher

$$p(x \mapsto y, x' \mapsto y') = p(x' \mapsto y')p(x \mapsto y | x' \mapsto y') = 1/m^2.$$

Umgekehrt rechnet man leicht nach, dass die Bedingung $\beta = \frac{1}{m}$ erfüllt ist, wenn für alle Doppelpaare (x, y, x', y') mit $x \neq x'$ die Gleichheit $p(x \mapsto y, x' \mapsto y') = 1/m^2$ gilt. \square

Bemerkung 25. Nach obigem Satz gilt $\beta = \frac{1}{m}$ genau dann, wenn für alle Doppelpaare (x, y, x', y') mit $x \neq x'$ gilt,

$$p(x \mapsto y, x' \mapsto y') = \sum_{k \in K(x, y, x', y')} p(k) = \frac{1}{m^2}.$$

D.h. bei Gleichverteilung der Schlüssel gilt $\beta = \frac{1}{m}$ genau dann, wenn in je zwei Spalten der Authentifikationsmatrix jedes MAC-Wertepaar gleich oft vorkommt.

Ab jetzt setzen wir voraus, dass der Schlüssel unter Gleichverteilung gewählt wird, d.h. es gilt $p(k) = \frac{1}{\|K\|}$ für alle $k \in K$.

Definition 26. Ein MAC (X, Y, K, H) heißt **2-universal**, falls für alle $x, x' \in X$ mit $x \neq x'$ und alle $y, y' \in Y$ gilt:

$$\|K(x, y, x', y')\| = \frac{\|K\|}{m^2}.$$

Ein MAC (X, Y, K, H) ist also genau dann 2-universal, wenn für alle Textpaare $x, x' \in X$ mit $x \neq x'$ jedes MAC-Wertpaar $y, y' \in Y$ mit Wk $1/m^2$ auftritt.

Bemerkung 27. Bei der Konstruktion von 2-universalen MACs spielt der Parameter $\lambda = \frac{\|K\|}{m^2}$ eine wichtige Rolle. Da λ notwendigerweise positiv und ganzzahlig ist, muss insbesondere $\|K\| \geq m^2$ gelten.

Im Folgenden nennen wir einen 2-universalen (n, m, l) -MAC mit $\lambda = l/m^2$ kurz einen (n, m, l, λ) -MAC.

Auf Grund von Bemerkung 25 ist klar, dass ein MAC bei gleichverteilten Schlüsseln genau dann die Bedingung $\beta = \frac{1}{m}$ erfüllt, wenn er 2-universal ist. Auf Grund von Bemerkung 16 nimmt in diesem Fall auch α den optimalen Wert $\frac{1}{m}$ an.

Der nächste Satz zeigt eine einfache Konstruktionsmöglichkeit von 2-universalen MACs mit dem Parameterwert $\lambda = 1$.

Satz 28. Sei p prim und für $a, b, x \in \mathbb{Z}_p$ sei

$$h_{a,b}(x) = ax + b \pmod{p}.$$

Dann ist (X, Y, K, H) mit $X = Y = \mathbb{Z}_p$ und $K = \mathbb{Z}_p \times \mathbb{Z}_p$ ein $(p, p, p^2, 1)$ -MAC.

Beweis. Wir müssen zeigen, dass die Größe von $K(x, y, x', y')$ für alle Doppelpaare (x, y, x', y') mit $x \neq x'$ konstant ist. Ein Schlüssel (a, b) gehört genau dann zu dieser Menge, wenn er die beiden Kongruenzen

$$\begin{aligned} ax + b &\equiv_p y, \\ ax' + b &\equiv_p y' \end{aligned}$$

erfüllt. Da dies jedoch nur auf den Schlüssel (a, b) mit

$$\begin{aligned} a &= (y' - y)(x' - x)^{-1} \pmod{p}, \\ b &= y - x(y' - y)(x' - x)^{-1} \pmod{p} \end{aligned}$$

zutrifft, folgt $\|K(x', y', x, y)\| = 1$. □

Die Hashfunktionen des vorigen Satzes erfüllen wegen $n = m = p$ nicht die Kompressions-eigenschaft. Zwar lässt sich n noch geringfügig von p auf $p + 1$ (und somit der Quotient n/m von 1 auf $\frac{p+1}{p}$) vergrößern, ohne K und Y zu verändern (siehe Übungen). Wie der nächste Satz zeigt, lässt sich eine stärkere Kompression mit dem Parameterwert $\lambda = 1$ jedoch nicht realisieren.

Satz 29. Für einen $(n, m, l, 1)$ -MAC gilt

$$n \leq m + 1$$

und somit $l = m^2 \geq (n - 1)^2$ sowie $n/m \leq \frac{m+1}{m} (\approx 1)$.

Beweis. O.B.d.A. sei $K = \{1, \dots, l\}$ und $Y = \{1, \dots, m\}$. Es ist leicht zu sehen, dass eine (bijektive) Umbenennung $\pi: Y \rightarrow Y$ der MAC-Werte in einer einzelnen Spalte der Authentikationsmatrix A wieder auf einen 2-universalen MAC führt. Also können wir zudem annehmen, dass die erste Zeile der Authentikationsmatrix A nur Einsen enthält. Da A 2-universal ist, gilt:

- In jeder Zeile $i = 2, \dots, m^2$ kommt höchstens eine Eins vor.
- Jede Spalte j enthält eine Eins in Zeile 1 und $m - 1$ Einsen in den übrigen Zeilen.

Da in den Zeilen $i = 2, \dots, m^2$ insgesamt genau $n(m - 1)$ Einsen vorkommen, folgt

$$\underbrace{\text{Anzahl der Zeilen}}_{m^2} \geq \underbrace{\text{Anzahl der Zeilen mit einer Eins}}_{1+n(m-1)},$$

was $m^2 - 1 \geq n(m - 1)$ bzw. $n \leq m + 1$ impliziert. \square

Der nächste Satz liefert 2-universale MACs mit beliebig großem Kompressionsfaktor. Für den Beweis benötigen wir das folgende Lemma.

Lemma 30. *Sei A eine $(k \times \ell)$ -Matrix über einem endlichen Körper \mathbb{F} , deren k Zeilen linear unabhängig sind. Dann besitzt das lineare Gleichungssystem*

$$Ax = y$$

für jedes $y \in \mathbb{F}^k$ genau $|\mathbb{F}|^{\ell-k}$ Lösungen $x \in \mathbb{F}^\ell$.

Beweis. Siehe Übungen. \square

Satz 31. *Sei p prim und für $x = (x_1, \dots, x_d) \in \{0, 1\}^d$ und $k = (k_1, \dots, k_d) \in \mathbb{Z}_p^d$ sei*

$$h_k(x) = kx = \sum_{i=1}^d k_i x_i \pmod{p}.$$

Dann ist (X, Y, K, H) mit $X = \{0, 1\}^d - \{0^d\}$, $Y = \mathbb{Z}_p$ und $K = \mathbb{Z}_p^d$ ein $(2^d - 1, p, p^d, p^{d-2})$ -MAC.

Beweis. Wir müssen zeigen, dass die Größe von $K(x, y, x', y')$ für alle Doppelpaare (x, y, x', y') mit $x \neq x'$ konstant ist. Es gilt

$$\begin{aligned} k \in K(x, y, x', y') &\Leftrightarrow h_k(x) = y \wedge h_k(x') = y' \\ &\Leftrightarrow k \cdot x = y \wedge k \cdot x' = y'. \end{aligned}$$

Fassen wir $x = x_1 \cdots x_d$ und $x' = x'_1 \cdots x'_d$ zu einer Matrix A zusammen, so ist dies äquivalent zu

$$\begin{pmatrix} x_1 & \cdots & x_d \\ x'_1 & \cdots & x'_d \end{pmatrix} \cdot \begin{pmatrix} k_1 \\ \vdots \\ k_d \end{pmatrix} = \begin{pmatrix} y \\ y' \end{pmatrix}.$$

Da die beiden Zeilen von A verschieden und damit linear unabhängig sind, folgt mit obigem Lemma, dass genau $|K(x, y, x', y')| = p^{d-2}$ Schlüssel $k = (k_1, \dots, k_d)$ mit dieser Eigenschaft existieren. \square

Bemerkung 32. Obige Konstruktion liefert einen λ -Wert von $\frac{\|K\|}{m^2} = p^{d-2}$. Durch Erweiterung von X auf eine geeignete Teilmenge $X' \subseteq \mathbb{Z}_p^d$ lässt sich der Textraum von $2^d - 1$ auf $\frac{p^d-1}{p-1}$ vergrößern (siehe Übungen). Dies führt auf einen beliebig groß wählbaren Kompressionsfaktor $\frac{n}{m} = \frac{p^d-1}{p(p-1)} \approx p^{d-2}$ bei einem λ -Wert von $\lambda = p^{d-2}$. Wie der nächste Satz zeigt, lässt sich dies nicht mit einem kleineren λ -Wert (bzw. nicht mit einer kleineren Schlüssellänge) erreichen.

Im Beweis des nächsten Satzes benötigen wir folgendes Lemma.

Lemma 33. Für beliebige reelle Zahlen $b_1, \dots, b_m \in \mathbb{R}$ gilt $\left(\sum_{i=1}^m b_i\right)^2 \leq m \sum_{i=1}^m b_i^2$.

Beweis. Da die Funktion $x \mapsto x^2$ konvex ist, folgt mit der Jensenschen Ungleichung $(\sum b_i/m)^2 \leq \sum b_i^2/m$ und somit

$$\left(\sum b_i\right)^2 = m^2 \underbrace{\left(\sum b_i/m\right)^2}_{\leq \sum b_i^2/m} \leq m \sum b_i^2.$$

□

Satz 34. Für jeden (n, m, l, λ) -MAC gilt

$$\underbrace{\lambda m^2}_{=l} \geq n(m-1) + 1$$

und somit $n/m \leq (\lambda - 1/m^2) \frac{m}{m-1} (\approx \lambda)$.

Beweis. O.B.d.A. können wir wieder $K = \{k_1, \dots, k_l\}$ und $Y = \{1, \dots, m\}$ annehmen, und dass die erste Zeile der Authentifikationsmatrix nur aus Einsen besteht. Für jede Zeile $i = 1, \dots, l$ bezeichne e_i die Anzahl der Einsen in dieser Zeile (also $e_1 = n$). Da in jeder Spalte jeder MAC-Wert genau λm -mal vorkommt, gilt

$$\sum_{i=1}^l e_i = \lambda n m \quad \text{und} \quad \sum_{i=2}^l e_i = \lambda n m - n = n(\lambda m - 1).$$

Sei $z = \sum_{i=2}^l z_i$, wobei z_i die Anzahl von Spaltenpaaren (j, j') mit $j \neq j'$ und $h_{k_i}(x_j) = h_{k_i}(x_{j'}) = 1$ ist. Dann folgt

$$z = \sum_{i=2}^l z_i = \sum_{i=2}^l e_i(e_i - 1) = \sum_{i=2}^l e_i^2 - \sum_{i=2}^l e_i = \sum_{i=2}^l e_i^2 - n(\lambda m - 1).$$

Mit obigem Lemma ergibt sich

$$\sum_{i=2}^l e_i^2 \geq \frac{\left(\sum_{i=2}^l e_i\right)^2}{l-1} = \frac{(n(\lambda m - 1))^2}{l-1}.$$

Da andererseits in jedem Spaltenpaar das MAC-Wertepaar $(1, 1)$ in genau λ Zeilen vorkommt (genauer: einmal in Zeile 1 und $(\lambda - 1)$ -mal in den Zeilen $i = 2, \dots, l$), und da $n(n-1)$ solche Spaltenpaare existieren, ergibt sich andererseits die Gleichung

$$z = (\lambda - 1)n(n - 1).$$

Somit erhalten wir

$$\begin{aligned}
(\lambda - 1)n(n - 1) = z &= \sum_{i=2}^l e_i^2 - n(\lambda m - 1) \geq \frac{(n(\lambda m - 1))^2}{l - 1} - n(\lambda m - 1) \\
&\Rightarrow ((\lambda - 1)n(n - 1) + n(\lambda m - 1))(\lambda m^2 - 1) \geq (n(\lambda m - 1))^2 \\
&\Rightarrow (\lambda n - n - \lambda + \lambda m)(\lambda m^2 - 1) \geq n(\lambda m - 1)^2 \\
&\Rightarrow -\lambda^2 m^2 + \lambda^2 m^3 \geq \lambda n m^2 + \lambda n - \lambda + \lambda m - 2\lambda n m \\
&\Rightarrow \lambda^2(m^3 - m^2) \geq \lambda(n(m - 1)^2 + m - 1) \\
&\Rightarrow \lambda m^2 \geq n(m - 1) + 1.
\end{aligned}$$

□

1.3.3 CBC-MACs

Als Basis für die Konstruktion eines MAC kann auch ein symmetrisches Kryptosystem dienen.

Sei (M, C, K, E, D) ein symmetrisches Kryptosystem mit $M = C = \{0, 1\}^t$. Zudem sei $IV := 0^t$ und sei $k \in K$ ein geheimer Schlüssel. Sei y eine Funktion für den Preprocessing-Schritt, die für jeden Text $x \in \{0, 1\}^*$ einen nichtleeren Bitstring $y(x) \in \bigcup_{n \geq 1} \{0, 1\}^{tn}$ liefert, dessen Länge durch t teilbar ist.

Berechnung von $h_k(x)$:

```

1  $y := y(x) = y_1 \dots y_n, \quad n \geq 1, \quad y_i \in \{0, 1\}^t$ 
2  $z_0 := IV$ 
3 for  $i = 1$  to  $n$  do
4    $z_i := E(k, z_{i-1} \oplus y_i)$ 
5 output  $h_k(x) = z_n$ 

```

Die MAC-Wertlänge beträgt also t Bit. Wird auf den Preprocessing-Schritt verzichtet, so lässt sich leicht ein Angriff mit 2 adaptiven Fragen ausführen. Kennt der Gegner die MAC-Werte $z = h_k(x)$ und $z' = h_k(x')$ für die Texte $x = x_1 \dots x_n$ und $x' = (x_{n+1} \oplus IV \oplus z)x_{n+2} \dots x_{n+m}$, wobei $|x_i| = t$ für $i = 1, \dots, n + m$ ist, so muss auch der Text $x'' = x_1 \dots x_{n+m}$ den MAC-Wert $h_k(x'') = z'$ haben.

Diesen Angriff kann man zwar ausschließen, indem man eine feste Länge nt für die Texte vorschreibt, wodurch die Anwendbarkeit des CBC-MACs allerdings eingeschränkt wird. Der folgende Geburtstagsangriff ist auch bei fester Textlänge möglich.

Geburtstagsangriff auf einen CBC-MAC

Dieser Angriff ermöglicht es, mit $q + 1$ MAC-Fragen (wobei $q \approx 1,17 \cdot 2^{\frac{t}{2}}$) den MAC-Wert $h_k(x)$ für einen zuvor nicht erfragten Text x zu finden, wobei $x = x_1 \dots x_n \in \{0, 1\}^{tn}$ abgesehen vom ersten t -Bitblock $x_1 \in \{0, 1\}^t$ beliebig wählbar ist. Hierzu wählt der Gegner zunächst $n - 2$ beliebige Blöcke $x_3, \dots, x_n \in \{0, 1\}^t$ und $q \approx 1,17 \cdot 2^{\frac{t}{2}}$ paarweise verschiedene Blöcke $x_1^1, \dots, x_1^q \in \{0, 1\}^t$. Anschließend wählt er zufällig q weitere Blöcke $x_2^1, \dots, x_2^q \in \{0, 1\}^t$ und erfragt die MAC-Werte $z_i = h_k(x^i)$ für die Texte $x^i = x_1^i x_2^i x_3 \dots x_n, i = 1, \dots, q$.

Wegen $x_1^i \neq x_1^j$ für $i \neq j$ sind auch die Texte x^1, \dots, x^q paarweise verschieden. Seien z_1^1, \dots, z_1^q die nach der ersten Iteration des CBC-MACs berechneten Kryptotexte $z_1^i =$

$E_k(IV \oplus x_1^i)$. Da die Blöcke x_2^i zufällig gewählt werden, sind auch die Eingangsblöcke $z_1^i \oplus x_2^i$ für die 2. Iteration zufällig, d.h. es gilt

$$\Pr[\exists i \neq j : z_1^i \oplus x_2^i = z_1^j \oplus x_2^j] = \Pr[\exists i \neq j : x_2^i = x_2^j] \approx \frac{1}{2}.$$

Da die Gleichheit der Eingangsblöcke $z_1^i \oplus x_2^i$ und $z_1^j \oplus x_2^j$ für die 2. Iteration mit der Gleichheit der Ausgangsblöcke z_n^i und z_n^j der n -ten Iteration und damit mit der Gleichheit der zugehörigen MAC-Werte z^i und z^j äquivalent ist, kann der Gegner das Indexpaar (i, j) mit $z_1^i \oplus x_2^i = z_1^j \oplus x_2^j$ auch leicht finden, sofern es existiert (was wir im Folgenden annehmen).

Da $x_1^i \neq x_1^j$ gilt, sind auch die Blöcke $z_1^i = E_k(IV \oplus x_1^i)$ und $z_1^j = E_k(IV \oplus x_1^j)$ verschieden. Wegen $z_1^i \oplus x_2^i = z_1^j \oplus x_2^j$ sind dann auch die beiden Blöcke x_2^i und x_2^j verschieden. O.B.d.A. gelte $x_2^i \neq x_2$ (sonst vertauschen wir die Indizes i und j). Nun erfragt der Gegner den MAC-Wert $\tilde{z}_j = h_k(x^{q+1})$ für den Text

$$x^{q+1} = \tilde{x}^j = x_1^j \tilde{x}_2^j x_3 \cdots x_n,$$

wobei $\tilde{x}_2^j = x_2^j \oplus x_2^i \oplus x_2 \neq x_2^j$ ist. Wegen

$$z_1^j \oplus \tilde{x}_2^j = z_1^j \oplus x_2^j \oplus x_2^i \oplus x_2 = z_1^i \oplus x_2^i \oplus x_2^i \oplus x_2 = z_1^i \oplus x_2$$

ist dieser zugleich MAC-Wert des Textes $\tilde{x}^i = x_1^i x_2 x_3 \cdots x_n$, den er zuvor nicht erfragt hat.

Definition 35. Sei $0 \leq \varepsilon \leq 1$ und sei $q \in \mathbb{N}$. Ein (ε, q) -Fälscher für einen MAC \mathcal{H} ist ein probabilistischer Algorithmus \mathcal{A} , der q Fragen x_1, \dots, x_q stellt und aus den Antworten $z_i = h_k(x_i)$ mit Wahrscheinlichkeit mindestens ε (bei zufällig gewähltem Schlüssel k) ein Paar (x, z) berechnet mit $x \notin \{x_1, \dots, x_q\}$ und $h_k(x) = z$.

Wir unterscheiden zwischen adaptiven Fragen (d.h. der Text x_i darf von den MAC-Werten der Texte x_1, \dots, x_{i-1} abhängen) und nicht-adaptiven Fragen. Zudem unterscheiden wir zwischen selektiven Fälschungen (d.h. der Gegner kann den MAC-Wert für einen Text seiner Wahl generieren) und existentiellen Fälschungen (d.h. der Gegner kann den MAC-Wert für irgendeinen Text $x \notin \{x_1, \dots, x_q\}$ generieren, auf dessen Wahl er keinen Einfluss hat).

Beispiel 36. Der oben beschriebene Geburtstagsangriff auf einen CBC-MAC führt auf einen $(\frac{1}{2}, q+1)$ -Fälscher für $q \approx 1,17 \cdot 2^{\frac{t}{2}}$. Dabei ist nur die letzte MAC-Frage adaptiv und der Text x kann abgesehen vom ersten t -Bitblock beliebig vorgegeben werden. \triangleleft

Eine Variante dieses Angriffs ist auch bei Verwendung einer Preprocessing-Funktion möglich. Meist wird hierzu die Funktion $y : x \mapsto y(x) = y_0 \dots y_n$ mit $y_0 = \text{bin}_t(|x|)$ und $y_1 \dots y_n = x^{0^{nt-|x|}}$ verwendet, wobei $n = \lceil |x|/t \rceil$ ist. Der erste Block $y_0 = \text{bin}_t(|x|)$ kodiert also die Länge von x als Binärzahl, die mit führenden Nullen auf die Länge t erweitert wird, und der letzte Block wird ebenfalls mit Nullen auf die Länge t aufgefüllt.

1.3.4 Kombination einer Hashfunktion mit einem MAC (HMAC)

Falls der Textraum eines MAC den Werteraum eines anderen MAC enthält, lassen sich diese leicht komponieren (Nested-MAC oder NMAC).

Definition 37. Seien $\mathcal{H}_1 = (X, Y, K_1, F)$ mit $F = \{f_k \mid k \in K_1\}$ und $\mathcal{H}_2 = (Y, Z, K_2, G)$ mit $G = \{g_k \mid k \in K_2\}$ MACs. Dann ist $\mathcal{H}_1 \circ \mathcal{H}_2 = (X, Z, K, H)$ die Komposition von \mathcal{H}_1 und \mathcal{H}_2 , wobei $K = K_1 \times K_2$ und $H = \{g_{k_2} \circ f_{k_1} \mid (k_1, k_2) \in K\}$ ist.

Beispiel 38. Wählt man für \mathcal{H}_2 einen MAC mit fester Textlänge und für \mathcal{H}_1 eine (schlüssellose) Hashfunktion (etwa **SHA-1**), so erhält man einen so genannten HMAC (Hash-MAC). \triangleleft

Eine Variante hiervon ist der auf **SHA-1** basierende HMAC, bei dem zwei Varianten von **SHA-1** mit symmetrischen Schlüsseln komponiert werden, wobei jedoch beidesmal derselbe Schlüssel benutzt wird. Seien

$$ipad = \underbrace{36 \dots 36}_{64mal} \quad \text{und} \quad opad = \underbrace{5C \dots 5C}_{64mal}$$

512 Bit Konstanten. Dann berechnet sich HMAC wie folgt:

$$\text{HMAC}_k(x) = \text{SHA-1}((k \oplus opad)\text{SHA-1}((k \oplus ipad)x)).$$

Hierbei fungiert die Funktion $f_k(x) = \text{SHA-1}((k \oplus ipad)x)$ als Hashfunktion mit Schlüssel, die beliebig lange Texte hasht, und der MAC $g_k(y) = \text{SHA-1}((k \oplus opad)y)$ wird nur auf Bitstrings der Länge 512 angewendet. Wie der folgende Satz zeigt, genügt es, wenn f_k kollisionsresistent und g_k berechnungsresistent ist, um einen berechnungsresistenten HMAC zu erhalten.

Definition 39. Ein (ε, q) -Kollisionsangreifer für einen MAC $\mathcal{H} = (X, Y, K, H)$ ist ein probabilistischer Algorithmus \mathcal{A} , der q Fragen x_1, \dots, x_n stellt und aus den Antworten $y_i = h_k(x_i)$ mit Wahrscheinlichkeit mindestens ε ein Paar (x, x') berechnet mit $h_k(x) = h_k(x')$, wobei k der dem Gegner unbekannt (und zufällig gewählte) Schlüssel ist.

Da der Gegner den Schlüssel k nicht kennt, ist ein Kollisionsangriff gegen einen MAC \mathcal{H} meist schwieriger zu realisieren als ein Kollisionsangriff gegen eine schlüssellose Hashfunktion. Andererseits ist die Kenntnis des Schlüssels bei einem Geburtstagsangriff nicht von Vorteil.

Satz 40. Seien $\mathcal{H}_1 = (X, Y, K_1, F)$, $\mathcal{H}_2 = (Y, Z, K_2, G)$ MACs. Falls für \mathcal{H}_1 kein adaptiver $(\varepsilon_1, q+1)$ -Kollisionsangriff und für \mathcal{H}_2 kein adaptiver (ε_2, q) -Fälscher existieren, dann existiert auch für $\mathcal{H} = \mathcal{H}_1 \circ \mathcal{H}_2$ kein adaptiver $(\varepsilon_1 + \varepsilon_2, q)$ -Fälscher.

Beweis. Sei A ein adaptiver (ε, q) -Fälscher für \mathcal{H} . Seien x_1, \dots, x_q die Fragen, die A an sein Orakel $g_{k_2} \circ f_{k_1}$ stellt, und seien $z_i = g_{k_2}(f_{k_1}(x_i))$ die erhaltenen Antworten. Zudem sei (x, z) die Ausgabe von A . Wir müssen zeigen, dass die Erfolgswk von A

$$\Pr[x \notin \underbrace{\{x_1, \dots, x_q\}}_B \wedge \underbrace{g_{k_2}(f_{k_1}(x)) = z}_C] < \varepsilon_1 + \varepsilon_2$$

ist, wobei (k_1, k_2) zufällig aus $K = K_1 \times K_2$ gewählt wird.

Behauptung 41. $\Pr[\underbrace{x \notin \{x_1, \dots, x_q\}}_B \wedge \underbrace{f_{k_1}(x) \in \{f_{k_1}(x_1), \dots, f_{k_1}(x_q)\}}_D] < \varepsilon_1.$

Hierzu betrachten wir den adaptiven Kollisionsangreifer A' gegen \mathcal{H}_1 , der zufällig einen Schlüssel $k_2 \in K_2$ wählt und A wie folgt simuliert.

Für jede Frage x_i von A erfragt A' den MAC-Wert $y_i = f_{k_1}(x_i)$ und gibt an A die Antwort $z_i = g_{k_2}(y_i)$ zurück. Sobald A ein Paar (x, z) ausgibt, erfragt A' den MAC-Wert $y = f_{k_1}(x)$ und gibt im Fall $x \notin \{x_1, \dots, x_q\} \wedge y \in \{y_1, \dots, y_q\}$ das Paar (x, x_i) für einen beliebigen Index i mit $y = y_i$ aus.

Da A' genau dann Erfolg hat, wenn das Ereignis $B \cap D$ eintritt, folgt Behauptung 41.

Behauptung 42. $\Pr[\underbrace{f_{k_1}(x) \notin \{f_{k_1}(x_1), \dots, f_{k_1}(x_q)\}}_{\bar{D}} \wedge \underbrace{g_{k_2}(f_{k_1}(x)) = z}_{C}] < \varepsilon_2.$

Hierzu betrachten wir den adaptiven Fälscher A'' gegen \mathcal{H}_2 , der zufällig einen Schlüssel $k_1 \in K_1$ wählt und A wie folgt simuliert.

A'' gibt bei jeder Anfrage x_i von A die Antwort des Orakels g_{k_2} auf die Frage $y_i = f_{k_1}(x_i)$ zurück und sobald A ein Paar (x, z) ausgibt, gibt A'' das Paar $(f_{k_1}(x), z)$ aus.

Da A'' genau dann Erfolg hat, wenn das Ereignis $\bar{D} \cap C$ eintritt, folgt Behauptung 42. Damit folgt

$$\Pr(B \cap C) = \underbrace{\Pr(B \cap \bar{D} \cap C)}_{< \varepsilon_2} + \underbrace{\Pr(B \cap D \cap C)}_{< \varepsilon_1} < \varepsilon_1 + \varepsilon_2.$$

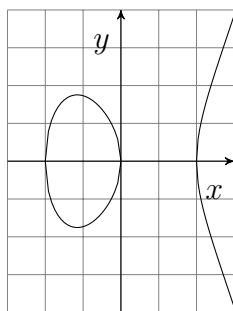
□

2 Elliptische Kurven

2.1 Elliptische Kurven über den reellen Zahlen

Definition 43. Seien $a, b \in \mathbb{R}$. Eine elliptische Kurve E über \mathbb{R} enthält alle Lösungen $(x, y) \in \mathbb{R}^2$ der Gleichung $y^2 = x^3 + ax + b$ und zusätzlich den Punkt \mathcal{O} (Punkt im Unendlichen; siehe Übungen). Im Fall $4a^3 + 27b^2 = 0$ heißt E singulär, sonst nicht-singulär.

Beispiel 44. Betrachte die durch $y^2 = x^3 - 4x$ definierte elliptische Kurve E .



Punkte: $(-2, 0)$, $(0, 0)$, $(2, 0)$, $(-1, \sqrt{3})$, $(-1, -\sqrt{3})$, $(3, \sqrt{15})$, $(3, -\sqrt{15})$.

Auf den nicht-singulären Punkten von E lässt sich eine additive Gruppenoperation $+$ definieren. Die Idee dabei ist, dass die Addition von 3 beliebigen Punkten von E , die auf einer Geraden liegen, das neutrale Element \mathcal{O} ergeben soll. Hierbei werden Tangentialpunkte doppelt und Wendepunkte dreifach gezählt und den parallel zur y -Achse verlaufenden Geraden wird zusätzlich noch der Punkt \mathcal{O} hinzugerechnet (d.h. alle Geraden, die parallel zur y -Achse verlaufen, schneiden sich im Punkt \mathcal{O}) und es werden nur solche Geraden g betrachtet, auf denen bei dieser Zählweise 3 Punkte von E liegen. Um nun die Summe $R = P + Q$ von zwei gegebenen Punkten $P = \{x_1, y_1\}$ und $Q = \{x_2, y_2\}$ zu berechnen, bestimmen wir zuerst die Gerade g , auf der P und Q liegen, wobei g im Fall $P = Q$ die Tangente an E im Punkt P ist.

Falls g parallel zur y -Achse verläuft, ist $x_1 = x_2$ und $y_1 = -y_2$ (also $Q = (x_1, -y_1)$). Da in diesem Fall zudem der Punkt \mathcal{O} auf g liegt, erhalten wir die Gleichung $P + Q(+\mathcal{O}) = \mathcal{O}$ bzw. $-P = Q = (x_1, -y_1)$.

Falls g nicht parallel zur y -Achse verläuft, können wir $P + Q$ wie folgt berechnen.

$P \neq Q$: In diesem Fall gilt $x_1 \neq x_2$. Zudem ist $g = \{(x, y) \in \mathbb{R}^2 | y = \lambda x + \mu\}$ mit $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$ und $\mu = y_1 - \lambda x_1 = y_2 - \lambda x_2$. Wir zeigen zuerst, dass es einen Punkt $R = (x_3, y_3) \in \mathbb{R}^2$ gibt mit

$$E \cap g = \{P, Q, R\}.$$

Für alle $(x, y) \in E \cap g$ gilt

$$(\lambda x + \mu)^2 = x^3 + ax + b$$

$$\leadsto \underbrace{x^3 - \lambda^2 x^2 + (a - 2\mu\lambda)x + b - \mu^2}_{p(x)} = 0.$$

Das Polynom $p(x)$ lässt sich in \mathbb{C} vollständig in Linearfaktoren zerlegen,

$$p(x) = (x - x_1)(x - x_2)(x - x_3).$$

Da sich der Koeffizient $-\lambda^2$ von x^2 aus der linearen Zerlegung von $p(x)$ zu

$$-\lambda^2 = -x_1 - x_2 - x_3$$

berechnet, muss $x_3 = \lambda^2 - x_1 - x_2$ sein. Da R auch auf g liegt, ist zudem $y_3 = \lambda(x_3 - x_1) + y_1$.

Folglich ist $P + Q = -R = (x_3, -y_3) = (\lambda^2 - x_1 - x_2, \lambda(x_1 - x_3) - y_1)$.

P = Q: In diesem Fall gilt $x_1 = x_2$ und $y_1 = y_2 \neq 0$. Sei g die Tangente durch P an E . Wir zeigen, dass es einen Punkt $R = (x_3, y_3) \in \mathbb{R}^2$ gibt mit

$$g \cap E = \{P, R\}.$$

Die Steigung λ von g erhalten wir durch implizites Differenzieren:

$$\lambda = \frac{dy}{dx} = \frac{-\frac{\partial F}{\partial x}(x_1, y_1)}{\frac{\partial F}{\partial y}(x_1, y_1)} = \frac{3x_1^2 + a}{2y_1},$$

wobei $F(x, y) = y^2 - x^3 - ax - b$ ist. Zur Begründung sei

$$T(x, y) = c(x - x_1) + d(y - y_1)$$

die Tangentialebene an die Fläche $F(x, y)$ im Punkt $(x_1, y_1, F(x_1, y_1)) = (x_1, y_1, 0)$. Dann gilt

$$c = \frac{\partial F}{\partial x}(x_1, y_1) = -3x_1^2 - a$$

und

$$d = \frac{\partial F}{\partial y}(x_1, y_1) = 2y_1.$$

Da die Tangente g sowohl in der Tangentialebene T als auch in der x, y -Ebene verläuft, folgt

$$\begin{aligned} (x, y) \in g &\Leftrightarrow T(x, y) = 0 \\ &\Leftrightarrow y - y_1 = -\frac{c}{d}(x - x_1), \end{aligned}$$

woraus sich $\lambda = -\frac{c}{d}$ ergibt. Genau wie im 1. Fall erhalten wir nun $P + Q = P + P = 2P = -R = (x_3, -y_3) = (\lambda^2 - x_1 - x_2, \lambda(x_1 - x_3) - y_1)$ mit $\lambda = \frac{3x_1^2 + a}{2y_1}$.

Satz 45. E bildet mit \mathcal{O} als neutralem Element und $+$ als Addition eine abelsche Gruppe, d.h.

- $+$ ist abgeschlossen auf E .
- $+$ ist kommutativ
- Jeder Punkt hat ein Inverses $-P$. P ist selbstinvers, falls $P = -P$ ist. Dies gilt für $P = \mathcal{O}$ und alle Kurvenpunkte der Form $P = (x, 0)$.
- $+$ ist assoziativ (ohne Beweis!).

2.2 Elliptische Kurven über endlichen Körpern

Definition 46. Sei \mathbb{F}_q ein endlicher Körper mit $q = p^n$ für eine Primzahl $p > 3$. Für $a, b \in \mathbb{F}_q$ mit $4a^3 + 27b^2 \neq 0$ heißt

$$E = \{(x, y) \in \mathbb{F}_q \mid y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}$$

elliptische Kurve über \mathbb{F}_q . Die Gruppenoperation $+$ ist auf E wie folgt definiert.

- \mathcal{O} ist neutrales Element, d.h. $\forall P \in E - \{\mathcal{O}\} : P + \mathcal{O} = \mathcal{O} + P = P$.
- Das Inverse zu $P = (x, y) \in E \setminus \{\mathcal{O}\}$ ist $-P = \bar{P} = (x, -y)$.
- Für $P, Q \in E \setminus \{\mathcal{O}\}$ ist

$$P + Q = \begin{cases} \mathcal{O}, & P = \bar{Q} \\ R, & \text{sonst} \end{cases}$$

wobei sich $R = (x_3, y_3)$ wie folgt aus $P = (x_1, y_1)$ und $Q = (x_2, y_2)$ berechnet:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

$$\text{wobei } \lambda = \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1}, & P \neq Q \\ (3x_1^2 + a)(2y_1)^{-1}, & P = Q \end{cases}$$

Satz 47. $(E, \mathcal{O}, +)$ bildet eine abelsche Gruppe (ohne Beweis).

Beispiel 48. Sei E definiert durch $y^2 = x^3 + x + 6$ über \mathbb{Z}_p , $p = 11$. Zur Erinnerung: Im Fall $p \equiv_4 3$ lassen sich für $z \in \mathbb{Q}R_p$ die Wurzeln y durch $\pm z^{\frac{p+1}{4}} \pmod p$ bestimmen.

x	0	1	2	3	4	5	6	7	8	9	10
$z = x^3 + x + 6$	6	8	5	3	8	4	8	4	9	7	4
$y = \pm\sqrt{z} \pmod{11}$	-	-	4; 7	5; 6	-	2; 9	-	2; 9	3; 8	-	2; 9

Da die Gruppe $(E, +, \mathcal{O})$ die Größe $\|E\| = 13$ hat und 13 prim ist, ist die Ordnung jedes Elements der Kurve 1 oder 13. Da nur das neutrale Element \mathcal{O} die Ordnung 1 haben kann, haben alle anderen Elemente $P \in E - \{\mathcal{O}\}$ die Ordnung 13 und sind daher Erzeuger der Gruppe. Folglich ist $(E, +, \mathcal{O})$ zyklisch und somit isomorph zu \mathbb{Z}_{13} : $(E, +, \mathcal{O}) \cong (\mathbb{Z}_{13}, +, 0)$.

Berechnung von $2g = (2, 7) + (2, 7) = (5, 2)$:

$$\begin{aligned} \lambda &= (3 \cdot 2^2 + 1)(2 \cdot 7)^{-1} \pmod{11} = 2 \cdot 3^{-1} = 2 \cdot 4 \pmod{11} = 8 \\ x_3 &= 8^2 - 2 - 2 \pmod{11} = 5 \\ y_3 &= 8(2 - 5) - 7 \pmod{11} = 2 \end{aligned}$$

Berechnung von $3g = 2g + g = (5, 2) + (2, 7) = (8, 3)$:

$$\begin{aligned} \lambda &= (7 - 2)(2 - 5)^{-1} \pmod{11} = 5 \cdot (-3)^{-1} \pmod{11} = 2 \\ x_3 &= 2^2 - 5 - 2 \pmod{11} = 8 \\ y_3 &= 2 \cdot (5 - 8) - 2 \pmod{11} = 3 \end{aligned}$$

k	1	2	3	4	5	6	7	8	9	10	11	12	13
$k \cdot g$	(2, 7)	(5, 2)	(8, 3)	(10, 2)	(3, 6)	(7, 9)	(7, 2)	(3, 5)	(10, 9)	(8, 8)	(5, 9)	(2, 4)	\mathcal{O}

◁

Satz 49. (Hasse) Für die Anzahl $\|E\|$ von Punkten einer elliptischen Kurve über einem endlichen Körper \mathbb{F}_q gilt

$$q + 1 - 2\sqrt{q} \leq \|E\| \leq q + 1 + 2\sqrt{q} \quad (\text{ohne Beweis}).$$

Bemerkung 50. Es gibt einen effizienten Algorithmus (von Schoof) mit Zeitkomplexität $O(\log^8 q)$, der $\|E\|$ bei Eingabe von a, b und q berechnet.

Satz 51. Sei E eine elliptische Kurve über \mathbb{F}_q . Dann ist $(E, \mathcal{O}, +)$ isomorph zu $(\mathbb{Z}_{n_1}, 0, +) \times (\mathbb{Z}_{n_2}, 0, +)$, wobei $n_1, n_2 \in \mathbb{N}^+$ sind und n_1 Teiler von n_2 und von $q - 1$ ist (ohne Beweis).

Bemerkung 52. Falls n_1 ein Teiler von n_2 ist, ist die (additive) Gruppe $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$ genau dann zyklisch, wenn $n_1 = 1$ (und somit $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \cong \mathbb{Z}_{n_2}$) ist. Eine hinreichende Bedingung hierfür ist, dass $\|E\|$ quadratfrei (also das Produkt von paarweise verschiedenen Primzahlen) ist.

Im Fall $n_1 > 1$ ist E dagegen nicht zyklisch, hat aber eine nicht-triviale zyklische Untergruppe, die zu \mathbb{Z}_{n_2} isomorph ist und für kryptografische Anwendungen benutzt werden kann.

Kompakte Darstellung von Punkten auf E

Für den Fall, dass sich Quadratwurzeln effizient in \mathbb{F}_q berechnen lassen, gibt es eine einfache Möglichkeit, Punkte auf einer elliptischen Kurve über \mathbb{F}_q kompakter darzustellen. Ist zum Beispiel $q = p$ prim mit $p \equiv_4 3$, so lassen sich die Wurzeln $\pm\sqrt{z} = \pm z^{(p+1)/4} \pmod p$ von $z \in \{x^2 \pmod p \mid x \in \mathbb{Z}_p\}$ effizient berechnen.

Folgende Funktion liefert dann eine kompakte Darstellung.

PointCompress: $E - \{\mathcal{O}\} \rightarrow \mathbb{Z}_p \times \mathbb{Z}_2$ mit $(x, y) \mapsto (x, y \pmod 2)$.

Für die Rekonstruktion können wir folgende Prozedur benutzen. Sei E eine durch $y^2 = p(x)$ mit $p(x) = x^3 + ax + b$ definierte elliptische Kurve über \mathbb{F}_p und $p \equiv_4 3$ prim,

Prozedur PointDeCompress(x, b)

```

1  $z := p(x) \pmod p$ 
2  $y := z^{(p+1)/4} \pmod p$ 
3 if  $y^2 \equiv_p z$  then
4   if  $y \not\equiv_2 b$  then  $y := p - y$ 
5   output( $x, y$ )
6 else output( ' 'error' ' )
```

Effiziente Berechnung von Vielfachen von Punkten auf E

In \mathbb{Z}_m^* berechnen wir Potenzen $a^e \pmod m$ durch ‘wiederholtes Quadrieren und Multiplizieren’. Ähnlich können wir in einer elliptischen Kurve E die Vielfachen mP eines Punktes P durch ‘wiederholtes Verdoppeln und Addieren’ berechnen. Da in E additive Inverse sehr leicht zu berechnen sind, kann mP durch ‘wiederholtes Verdoppeln, Addieren und Subtrahieren’ noch effizienter berechnet werden. Hierzu repräsentieren wir m in NAF-Darstellung (Non Adjacent Form).

Definition 53. $(c_{l-1}, \dots, c_0) \in \{-1, 0, 1\}^l$ heißt **SBR-Darstellung** (Signed Binary Representation) einer Zahl $c \in \mathbb{Z}$, falls

$$\sum_{i=0}^{l-1} c_i 2^i = c$$

ist. Ist von je zwei benachbarten Ziffern c_i und c_{i+1} mindestens eine 0, so ist (c_{l-1}, \dots, c_0) eine **NAF-Darstellung** von c .

Beispiel 54. Sowohl $(0, 1, 0, 1, 1)$ als auch $(1, 0, -1, 0, -1)$ sind SBR-Darstellungen von $c = 1 + 2 + 8 = 11 = -1 - 4 + 16$. \triangleleft

Satz 55. Jede Zahl $c \in \mathbb{Z}$ hat eine eindeutige NAF-Darstellung (Beweis siehe Übungen).

Berechnung einer NAF-Darstellung aus der Binärdarstellung: Ersetze jeden Teilstring der Form $(0, 1, \dots, 1)$ von rechts beginnend durch den Teilstring $(1, 0, \dots, 0, -1)$.

Beispiel 56. Um die NAF-Darstellung von $c = 79$ zu berechnen, bestimmen wir zuerst die Binärdarstellung von c . Es gilt $79_{10} = 101111_2$. Mit obiger Transformationsregel ergibt sich

$$\begin{aligned} & (0, 1, \underbrace{0, 1, 1, 1, 1}) \\ & \rightsquigarrow (\underbrace{0, 1, 1}, 0, 0, 0, -1) \\ & \rightsquigarrow (1, 0, -1, 0, 0, 0, -1) \end{aligned} \quad \triangleleft$$

Zur effizienten Berechnung von $Q = cP$ benutzen wir das Horner-Schema

$$c = \sum_{j=0}^s c_j 2^j = (\dots (\dots (\underbrace{c_s 2 + c_{s-1}}_{d_i}) 2 + \dots + c_i) 2 + \dots + c_1) 2 + c_0,$$

welches auf das folgende iterative Schema zur Berechnung der Punkte $Q_i = d_i P = \sum_{j=i}^s c_j 2^{j-i} P$ führt:

$$Q_i = \begin{cases} \mathcal{O}, & i = s + 1 \\ 2Q_{i+1} + c_i P, & i = s, \dots, 0. \end{cases}$$

Damit erhalten wir folgenden Algorithmus zur Berechnung von $Q = Q_0 = cP$:

Prozedur `DoubleAddSub`(P, c_s, \dots, c_0)

```

1  $Q := \mathcal{O}$ 
2 for  $i := s$  downto 0 do
3    $Q := 2Q + c_i P$ 
4 output( $Q$ )

```

Da eine $(s+1)$ -Bitzahl im Durchschnitt $s/2$ Nullen in Binärdarstellung und $(2/3)s$ Nullen in NAF-Darstellung enthält (siehe Übungen), benötigt `DoubleAddSub` bei Verwendung von NAF ca. $(4/3)s$ Additionen/Subtraktionen im Vergleich zu ca. $(3/2)s$ Additionen im Binärfall. Dies entspricht einer Beschleunigung um ca. 11 Prozent.

3 Digitale Signaturverfahren

Handschriftliche Signaturen

- Die durch die Unterschrift gekennzeichnete Person hat überprüfbar die Unterschrift geleistet.
- Die Unterschrift ist nicht auf ein anderes Dokument übertragbar, ohne ihre Gültigkeit zu verlieren.
- Das signierte Dokument kann nachträglich nicht unbemerkt verändert werden.

Eine direkte Übertragung dieser Eigenschaften in die digitale Welt ist nicht möglich.

Lösung: Die digitale Signatur wird nicht physikalisch, sondern logisch (inhaltlich) an ein elektronisches Dokument bzw. Text gebunden und die Fähigkeit, einen individuellen Schriftzug auszuführen, wird durch geheimes Wissen ersetzt.

Definition 57. Ein digitales Signaturverfahren besteht aus:

- einer Menge X von **Texten**,
- einer endlichen Menge Y von **Signaturen**,
- einem **Schlüsselraum** K ,
- einer Menge $S \subseteq K \times K$ von Schlüsselpaaren (\hat{k}, k) , bestehend aus einem **Signierschlüssel** \hat{k} und einem **Verifikationsschlüssel** k ,
- einem **Signieralgorithmus** $\text{sig} : K \times X \rightarrow Y$ und
- einem **Verifikationsalgorithmus** $\text{ver} : K \times X \times Y \rightarrow \{0, 1\}$, so dass $\text{ver}(k, x, y) = 1$ für alle Paare $(\hat{k}, k) \in S$ und $(x, y) \in X \times Y$ mit $y = \text{sig}(\hat{k}, x)$ gilt.

Im Fall $\text{ver}(k, x, y) = 1$ heißt y **gültige** Signatur für den Text x (unter k), andernfalls **ungültig**.

Ein wichtiger Unterschied zu MACs besteht darin, dass digitale Signaturverfahren asymmetrisch sind. Aufgrund dieser Asymmetrie kann Bob nämlich auch einem Dritten gegenüber nachweisen, dass eine von Alice erzeugte Signatur y tatsächlich von Alice stammt. Bei Verwendung eines MACs zur Authentifikation einer Nachricht x könnte Bob die Nachricht manipuliert und den MAC-Wert auch selbst erzeugt haben, weshalb Alice ihre Urheberschaft von x erfolgreich abstreiten kann.

Ein weiterer Vorteil von digitalen Signaturen gegenüber MACs ist, dass eine von Alice geleistete Signatur von allen verifizierbar ist, sofern sie den öffentlichen Verifikationsschlüssel von Alice kennen. Um bspw. die Authentizität eines Software-Updates x zu gewährleisten, kann eine SW-Firma das Update x zusammen mit ihrer Signatur y für x verschicken. Bei Verwendung eines MACs müsste die SW-Firma dagegen mit jedem einzelnen Kunden K_i einen symmetrischen Schlüssel k_i vereinbaren und den zugehörigen MAC-Wert $y_i = h_{k_i}(x)$ versenden.

Klassifikation von Angriffen gegen Signaturverfahren

Angriff bei bekanntem Verifikationsschlüssel (key-only attack): Dem Gegner ist nur der öffentliche Verifikationsschlüssel k bekannt und er versucht, ein Paar (x, y) mit $ver(k, x, y) = 1$ zu finden. Jedes solche Paar, das nicht von Alice unter Verwendung des geheimen Signierschlüssels erzeugt wurde, wird als **Fälschung** bezeichnet.

Angriff bei bekannter Signatur (known signature attack): Der Gegner kennt neben k die Signaturen $y_i = sig(\hat{k}, x_i)$ für eine Reihe von Texten x_1, \dots, x_q , auf deren Auswahl er keinen Einfluss hat, und versucht, eine Fälschung (x, y) mit $x \notin \{x_1, \dots, x_q\}$ zu finden.

Angriff bei frei wählbaren Texten (chosen document attack): Der Gegner kann die Texte x_1, \dots, x_q selbst wählen, erhält die Signaturen aber erst, nachdem er alle Texte vorgelegt hat.

Angriff bei adaptiv wählbaren Texten: Der Gegner kann die Wahl des Textes x_{i+1} von den Signaturen y_1, \dots, y_i abhängig machen.

Erfolgskriterien für die Fälschung digitaler Signaturen

uneingeschränktes Fälschungsvermögen (total break): Der Gegner hat einen Weg gefunden, die Funktion $x \mapsto sig(\hat{k}, x)$ bei Kenntnis von k effizient zu berechnen.

selektives Fälschungsvermögen (selective forgery): Der Gegner kann für Texte seiner Wahl die zugehörigen Signaturen bestimmen (eventuell mit Hilfe des legalen Unterzeichners).

nichtselektives (existentielles) Fälschungsvermögen: Der Gegner kann für bestimmte Texte x , auf deren Wahl er keinen Einfluss hat, die zugehörige digitale Signatur bestimmen.

3.1 Das RSA-Signaturverfahren

Beim **RSA-Signaturverfahren** ist $K = \{(a, n) | n = pq \text{ für Primzahlen } p, q \text{ und } a \in \mathbb{Z}_{\varphi(n)}^*\}$ und S die Relation $S = \{(d, n, e, n) \in K \times K | de \equiv_{\varphi(n)} 1\}$. Signiert wird mittels $sig(d, n, x) := x^d \bmod n$, wobei $X = Y = \mathbb{Z}_n$, und die Verifikationsbedingung ist

$$ver(e, n, x, y) = \begin{cases} 1, & y^e \equiv_n x \\ 0, & \text{sonst.} \end{cases}$$

Satz 58. Für alle $(d, n, e, n) \in S$ und $x, y \in \mathbb{Z}_n$ gilt:

$$ver(e, n, x, y) = \begin{cases} 1, & sig(d, n, x) = y, \\ 0, & \text{sonst.} \end{cases}$$

Beweis. Folgt direkt aus der Korrektheit des RSA-Kryptosystems. □

Wir betrachten eine Reihe von Angriffen gegen das RSA-Signaturverfahren und überlegen anschließend, durch welche Maßnahmen sich diese abwehren lassen.

- Es ist nicht schwer, eine nichtselektive Fälschung bei bekanntem Verifikationsschlüssel durchzuführen. Hierzu wählt der Gegner zu einer beliebigen Signatur $y \in Y$ den Text $x = y^e \bmod n$.

- Zudem ist eine existentielle Fälschung bei bekannten Signaturen möglich, falls der Gegner zwei signierte Texte $(x_1, y_1), (x_2, y_2)$ mit $ver(k, x_i, y_i) = 1$ kennt. Wegen $y_i^e \equiv_n x_i$ für $i = 1, 2$ folgt nämlich $(y_1 y_2)^e \equiv_n y_1^e y_2^e \equiv_n x_1 x_2$ und somit $ver(k, x_1 x_2 \bmod n, y_1 y_2 \bmod n) = 1$.
- Weiterhin kann der Gegner bei frei wählbaren Texten sogar eine selektive Fälschung durchführen. Ist bereits die Signatur für einen beliebigen Text $x' \in \mathbb{Z}_n^*$ bekannt und kann sich der Gegner die Signatur für den Text $x'' = x \cdot x'^{-1} \bmod n$ beschaffen, so kann er daraus wie oben eine gültige Signatur für den Text x berechnen.

Diese Angriffe kann man vereiteln, indem man den Text x mit Redundanz versieht (indem man z.B. anstelle von x den Text xx signiert). Um auch längere Texte effizient signieren zu können, wird i.a. jedoch eine geeignete Hashfunktion h benutzt und nicht der gesamte Text x , sondern nur der Hashwert $h(x)$ signiert.

Bei der Signaturerstellung benötigte Eigenschaften einer Hashfunktion h

- Die verwendete Hashfunktion h sollte die Einwegeigenschaft haben, da sonst der Gegner zu einem $y \in Y$ einen passenden Text x mit $h(x) = y$ bestimmen kann (zumindest wenn das Signaturverfahren anfällig gegen eine existentielle Fälschung ist, wie etwa RSA).
- Angenommen der Gegner kennt bereits ein Paar (x, y) mit $ver(k, h(x), y) = 1$. Dann sollte h zumindest schwach kollisionsresistent sein, da sonst der Gegner ein x' mit $h(x') = h(x)$ berechnen und das Paar (x', y) bestimmen könnte.
- Falls sich der Gegner für bestimmte von ihm selbst gewählte Texte x die zugehörige Signatur y beschaffen kann, so sollte h sogar kollisionsresistent sein. Andernfalls könnte der Gegner ein Kollisionspaar (x, x') für h finden, sich den (unverdächtigen) Text x signieren lassen und die erhaltene Signatur y für den Text x' verwenden.

3.2 Das ElGamal-Signaturverfahren

Das Signaturverfahren von ElGamal (1985) ist wie das gleichnamige asymmetrische Kryptosystem probabilistisch und beruht wie dieses auf dem diskreten Logarithmus.

Sei p eine große Primzahl und α ein Erzeuger von \mathbb{Z}_p^* (p und α sind öffentlich). Jeder Teilnehmer B wählt eine geheime Zahl $a \in \mathbb{Z}_{p-1} = \{0, \dots, p-2\}$ und gibt $\beta = \alpha^a \bmod p$ als Teil seines öffentlichen Verifikationsschlüssels bekannt:

Signierschlüssel: $\hat{k} = (p, \alpha, a)$,

Verifikationsschlüssel: $k = (p, \alpha, \beta)$.

Der Textraum ist $X = \mathbb{Z}_{p-1}$ und der Signaturenraum ist $Y = \mathbb{Z}_p^* \times \mathbb{Z}_{p-1} \setminus \{0\}$.

Signaturerstellung: Um einen Text $x \in X$ zu signieren, wählt der Signierer zufällig eine Zahl $z \in \mathbb{Z}_{p-1}^*$ und berechnet die Signatur $sig(\hat{k}, x, z) = (\gamma, \delta) \in Y$ mit $\gamma = \alpha^z \bmod p$ und $\delta = (x - a\gamma)z^{-1} \bmod p-1$. Falls $\delta = 0$ ist, muss eine neue Zufallszahl z gewählt und der Vorgang wiederholt werden.

Verifikation: $ver(k, x, (\gamma, \delta)) = 1$, falls $\beta^\gamma \gamma^\delta \equiv_p \alpha^x$ ist.

Lemma 59. Eine Signatur (γ, δ) mit $\text{ord}(\gamma) = p-1$ erfüllt genau dann die Verifikationsbedingung $\beta^\gamma \gamma^\delta \equiv_p \alpha^x$, wenn es ein $z \in \mathbb{Z}_{p-1}^*$ mit $sig(\hat{k}, x, z) = (\gamma, \delta)$ gibt.

Beweis. Wegen $\gamma \equiv \alpha^z \pmod{p}$ ist z durch γ (und γ durch z) eindeutig bestimmt. Weiter ist $\beta^\gamma \gamma^\delta \equiv_p \alpha^{a\gamma} \alpha^{z\delta} \equiv_p \alpha^{a\gamma+z\delta}$. Da α ein Erzeuger von \mathbb{Z}_p^* ist, gilt die Kongruenz $\alpha^{a\gamma+z\delta} \equiv_p \alpha^x$ genau dann, wenn $a\gamma + z\delta \equiv_{p-1} x$ ist, was wiederum mit $\delta \equiv_{p-1} (x - a\gamma)z^{-1}$ äquivalent ist. \square

Beispiel 60. Sei $p = 467$, $\alpha = 2$, $a = 127$ und $\beta = \alpha^a \pmod{p} = 2^{127} \pmod{467} = 132$. Um den Text $x = 100 \in \mathbb{Z}_{p-1} = \mathbb{Z}_{466}$ mit dem Signierschlüssel $\hat{k} = (p, \alpha, a) = (467, 2, 127)$ zu signieren, wählt Alice die geheime Zufallszahl $z = 213 \in \mathbb{Z}_{p-1}^*$ ($\rightsquigarrow z^{-1} \pmod{466} = 431$) und erhält

$$\gamma = 2^{213} \pmod{467} = 29 \text{ und } \delta = (100 - 127 \cdot 29)431 \pmod{466} = 51,$$

d.h. $\text{sig}(\hat{k}, x, z) = (29, 51)$. Um die Gültigkeit dieser Signatur für den Text $x = 100$ mit dem Verifikationsschlüssel $k = (p, \alpha, \beta) = (467, 2, 132)$ zu prüfen, verifiziert Bob die Kongruenz

$$\beta^\gamma \gamma^\delta \equiv_p 132^{29} 29^{51} \equiv_p 189 \equiv_p 2^{100} \equiv_p \alpha^x$$

◁

Zur Sicherheit des ElGamal-Systems

1. Falls der Gegner in der Gruppe \mathbb{Z}_p^* den diskreten Logarithmus von β zur Basis α bestimmen kann, so kann er den geheimen Schlüssel $a = \log_\alpha \beta$ berechnen.
2. Als nächstes betrachten wir verschiedene Szenarien für einen selektiven Angriff bei bekanntem Verifikationsschlüssel.
 - a) Der Gegner wählt zu einem gegebenen Text x zuerst γ und versucht, ein passendes δ zu finden. Mit $\alpha^x \equiv \beta^\gamma \gamma^\delta \pmod{p}$ folgt $\delta = \log_\gamma \alpha^x \beta^{-\gamma}$. D.h. die Bestimmung von δ ist eine Instanz des diskreten Logarithmus Problems (kurz: DLP).
 - b) Der Gegner wählt zu einem gegebenen Text x zuerst δ und versucht dann ein γ mit $\alpha^x \equiv \beta^\gamma \gamma^\delta \pmod{p}$ zu finden. Hierfür ist kein effizientes Verfahren bekannt.
 - c) Der Gegner versucht, zu einem gegebenen Text x gleichzeitig passende Zahlen γ und δ zu finden. Auch hierfür ist kein effizientes Verfahren bekannt.
3. Versucht der Gegner bei einem nichtselektiven Angriff, zuerst γ und δ zu wählen und dazu einen passenden Text x zu finden, so muss er den diskreten Logarithmus $x = \log_\alpha \beta^\gamma \gamma^\delta$ bestimmen.
4. Eine existentielle Fälschung lässt sich jedoch wie folgt durchführen (falls keine Hashfunktion benutzt wird). Der Gegner wählt beliebige Zahlen $u \in \mathbb{Z}_{p-1}$, $v \in \mathbb{Z}_{p-1}^*$ und berechnet $\gamma = \alpha^u \beta^v \pmod{p}$. Dann ist (γ, δ) genau dann eine gültige Signatur für einen Text x , wenn $\alpha^x \equiv_p \beta^\gamma (\alpha^u \beta^v)^\delta$ ist. Dies ist wiederum äquivalent zur Kongruenz $\alpha^{x-u\delta} \equiv_p \beta^{\gamma+v\delta}$, die sich im Fall $\text{ggT}(v, p-1) = 1$ für den Text $x = u\delta \pmod{p-1}$ mittels $\delta = -\gamma v^{-1} \pmod{p-1}$ erfüllen lässt. Bei Wahl von $v = 1$ erhalten wir z.B. die gültige Signatur $(\gamma, \delta) = (\alpha^u \beta \pmod{p}, -\alpha^u \beta \pmod{p-1})$ für den Text $x = u\delta \pmod{p-1}$, wobei $u \in \mathbb{Z}_{p-1}$ beliebig gewählt werden kann.

Bemerkung 61. Bei der Benutzung des ElGamal-Signaturverfahrens sind folgende Punkte zu beachten.

1. Die Zufallszahl z muss geheim gehalten werden.

2. Zufallszahlen dürfen nicht mehrfach verwendet werden.

Kennt nämlich der Gegner zu einer Signatur $(x, (\gamma, \delta))$ die Zufallszahl z , so kann er wegen $\delta \equiv_{p-1} (x - a\gamma)z^{-1}$ im Fall $\text{ggT}(\gamma, p-1) = 1$ die geheime Zahl

$$a = (x - z\delta)\gamma^{-1} \pmod{p-1}$$

als eindeutige Lösung der Kongruenz $\gamma a \equiv_{p-1} x - z\delta$ (*) berechnen. Ist allgemeiner $\text{ggT}(\gamma, p-1) = g \geq 1$, so ist g ein Teiler von γ und von $p-1$ sowie wegen (*) auch von $x - z\delta$. Setzen wir $\mu := \gamma/g$ und $\lambda := (x - z\delta)/g$, so führt (*) auf die Kongruenz $\mu a \equiv_{(p-1)/g} \lambda$ (**), aus der sich wegen $\text{ggT}(\mu, (p-1)/g) = 1$ folgende g Kandidaten a_i für a gewinnen lassen:

$$a_0 := \mu^{-1}\lambda \pmod{(p-1)/g} \text{ und } a_i := a_0 + i(p-1)/g \text{ für } i = 1, \dots, g-1.$$

Unter a_0, \dots, a_{g-1} lässt sich a durch Prüfen der Bedingung $\alpha^{a_i} \equiv_p \beta$ eindeutig bestimmen. Sind andererseits $(x_1, (\gamma, \delta_1))$ und $(x_2, (\gamma, \delta_2))$ mit demselben z generierte Signaturen, dann folgt wegen $\beta^\gamma \gamma^{\delta_1} \equiv_p \alpha^{x_1}$ und $\beta^\gamma \gamma^{\delta_2} \equiv_p \alpha^{x_2}$,

$$\gamma^{\delta_1 - \delta_2} \equiv_p \alpha^{x_1 - x_2} \Rightarrow \alpha^{z(\delta_1 - \delta_2)} \equiv_p \alpha^{x_1 - x_2} \Rightarrow z(\delta_1 - \delta_2) \equiv_{p-1} x_1 - x_2.$$

Aus dieser Kongruenz lassen sich $d = \text{ggT}(\delta_1 - \delta_2, p-1)$ Kandidaten für z gewinnen und daraus wie oben a berechnen, falls d nicht zu groß ist.

3.3 Das Schnorr-Signaturverfahren

Da die Primzahl p beim ElGamal-Signaturverfahren mindestens eine 512-Bit-Zahl (besser 1024-Bit-Zahl) sein sollte, beträgt die Signaturlänge 1024 bzw 2048 Bit. Folgende Variante des ElGamal-Signaturverfahrens, die als eine Vorstufe zum DSA betrachtet werden kann, wurde von Schnorr vorgeschlagen.

Die zugrunde liegende Idee ist folgende: Indem wir für α ein Element der Ordnung q mit $q \approx 2^{160}$ wählen, reduziert sich die Signaturlänge auf $2 \cdot 160 = 320$ Bit. Die Berechnungen werden aber nach wie vor modulo p mit $p \approx 2^{1024}$ ausgeführt, so dass das Problem des diskreten Logarithmus zur Basis α in \mathbb{Z}_p^* hart bleibt.

Sei g ein Erzeuger von \mathbb{Z}_p^* , wobei p die Bauart $p-1 = mq$ für eine Primzahl $q = \frac{p-1}{m} \approx 2^{160}$ hat. Dann ist $\alpha = g^{(p-1)/q}$ ein Element in \mathbb{Z}_p^* der Ordnung $\text{ord}_p(\alpha) = q$ (da $\text{ord}(g^i) = \frac{\text{ord}(g)}{\text{ggT}(i, \text{ord}(g))} = \frac{p-1}{\text{ggT}((p-1)/q, p-1)} = q$ ist; siehe Übungen). Weiter sei $h : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ eine Hashfunktion, die jedem Text $x \in X = \{0, 1\}^*$ einen Hashwert in \mathbb{Z}_q zuordnet.

Signierschlüssel: $\hat{k} = (p, q, \alpha, a)$, $a \in \mathbb{Z}_q$,

Verifikationsschlüssel: $k = (p, \alpha, \beta)$, $\beta = \alpha^a \pmod{p}$.

Signaturerstellung: Um einen Text $x \in X$ zu signieren, wählt der Signierer zufällig eine geheime Zahl $z \in \mathbb{Z}_q^*$ und berechnet

$$\text{sig}(\hat{k}, x, z) = (\gamma, \delta),$$

wobei $\gamma = h(\text{xbin}(\alpha^z \pmod{p}))$ und $\delta = (z + a\gamma) \pmod{q}$ ist. Der Signaturraum ist also $Y := \mathbb{Z}_q \times \mathbb{Z}_q$.

Verifikation: $\text{ver}(k, \gamma, \delta) = 1$, falls $h(\text{xbin}(\alpha^\delta \beta^{-\gamma} \pmod{p})) = \gamma$ ist.

Beispiel 62. Sei $q = 101$, $p = 78q + 1 = 7879$, $g = 3$, $\alpha = g^{(p-1)/q} = 3^{78} \bmod p = 170$, $a = 75$ und $\beta = \alpha^a \bmod p = 170^{75} \bmod 7879 = 4567$. Um einen Text $x \in \{0, 1\}^*$ mit dem Signierschlüssel $\hat{k} = (p, \alpha, a) = (7879, 170, 75)$ zu signieren, wählt Alice die geheime Zufallszahl $z = 50 \in \mathbb{Z}_q^*$ und berechnet den Wert $\alpha^z \bmod p = 170^{50} \bmod 7879 = 2518$. Dies führt auf den Hashwert $\gamma = h(\text{xbin}(2518)) \in \mathbb{Z}_q$. Unter der Annahme, dass $h(\text{xbin}(2518)) = 96$ ist, erhält Alice wegen

$$\delta = 50 + 75 \cdot 96 \bmod 101 = 79$$

die Signatur $\text{sig}(\hat{k}, x, z) = (96, 79)$. Um die Gültigkeit dieser Signatur für den Text x mit dem Verifikationsschlüssel $k = (p, \alpha, \beta) = (7879, 170, 4567)$ zu prüfen, berechnet Bob die Zahl

$$\beta^\gamma \gamma^\delta \equiv_p 170^{79} 4567^{-96} \equiv_p 2518$$

und verifiziert die Gleichheit $h(\text{xbin}(2518)) = 96$. ◁

3.4 Der Digital Signature Algorithm (DSA)

Der DSA wurde im August 1991 vom National Institute of Standards and Technology (NIST) für die Verwendung im Digital Signature Standard (DSS) empfohlen. Der DSS enthält neben dem DSA (ursprünglich der einzige im DSS definierte Algorithmus) als weitere Algorithmen die RSA-Signatur und ECDSA (siehe unten). Ausgehend vom ElGamal-Verfahren lässt sich der DSA durch folgende Modifikationen erhalten:

1. δ als Lösung von $z\delta - a\gamma \equiv_{p-1} x$ (d.h. $\delta = (x + a\gamma)z^{-1} \sim$ Verifikationsbedingung: $\alpha^x \beta^\gamma \equiv_p \gamma^\delta$ ($\alpha^x \alpha^{a\gamma} \equiv_p \alpha^{z(x+a\gamma)z^{-1}}$)
2. Ist $x + a\gamma \in \mathbb{Z}_{p-1}^*$, dann existiert $\delta^{-1} = (x + a\gamma)^{-1}z \bmod p - 1 \sim$ Verifikation durch: $\alpha^{x\delta^{-1}} \beta^{\gamma\delta^{-1}} \equiv_p \gamma$
3. Sei nun wie bei Schnorr $p = mq + 1$ mit $q \approx 2^{160}$ prim und sei $\alpha \in \mathbb{Z}_p^*$ mit $\text{ord}_p(\alpha) = q$. Dann kann bei der Verifikation von $\alpha^{x\delta^{-1}} \beta^{\gamma\delta^{-1}} \equiv_p \gamma$ auf der Exponentenebene *modulo* q gerechnet werden. Da γ jedoch rechts nicht als Exponent, sondern als Basiszahl, vorkommt, muss auch die linke Seite *modulo* q reduziert werden.

Beim DSA hat der Signierschlüssel also die Form $\hat{k} = (p, q, \alpha, a)$, wobei $a \in \mathbb{Z}_q^*$ ist, und der zugehörige Verifikationsschlüssel ist $k = (p, q, \alpha, \beta)$ mit $\beta = \alpha^a \bmod p$. Zudem gilt $X = \mathbb{Z}_q$ und $Y = \mathbb{Z}_q \times \mathbb{Z}_q^*$.

Zu gegebenem $x \in X$ wird zufällig eine geheime Zahl $z \in \mathbb{Z}_p^*$ gewählt.

$$\text{sig}(\hat{k}, z, x) = (\gamma, \delta), \text{ wobei } \begin{cases} \gamma = (\alpha^z \bmod p) \bmod q \\ \delta = (x + a\gamma)z^{-1} \bmod q \in \mathbb{Z}_q^* \end{cases}$$

Im Fall $\gamma = 0$ oder $\delta = 0$ muss ein neues z gewählt werden. Die Verifikationsbedingung ist

$$\text{ver}(k, x, \gamma, \delta) = \begin{cases} 1, & (\alpha^e \beta^d \bmod p) \bmod q = \gamma, \\ 0, & \text{sonst,} \end{cases}$$

wobei $e = x\delta^{-1} \bmod q$ und $d = \gamma\delta^{-1} \bmod q$ ist.

Korrektheit: Im Fall $\text{sig}(\hat{k}, z, x) = (\gamma, \delta)$ ist

$$\alpha^e \beta^d \equiv_p \alpha^{x\delta^{-1}} \alpha^{a\gamma\delta^{-1}} \equiv_p \alpha^{\delta^{-1}(x+a\gamma)} \equiv_p \alpha^{(x+a\gamma)^{-1}z(x+a\gamma)} \equiv_p \alpha^z$$

woraus sich

$$(\alpha^e \beta^d \bmod p) \bmod q = (\alpha^z \bmod p) \bmod q = \gamma$$

ergibt.

Beispiel 63. $q = 101$, $p = 78q + 1 = 7879$, $g = 3$ ($\text{ord}_p(3) = p - 1$)

$$\rightsquigarrow \alpha = 3^{78} \bmod p = 170 \text{ hat Ordnung } q$$

Wir wählen $a = 75 \in \mathbb{Z}_q^*$, d.h. $\beta = \alpha^a \bmod p = 170^{75} \bmod p = 4547$. Um den Text $x = 22 \in \mathbb{Z}_p^*$ zu signieren, wählen wir die geheime Zufallszahl $z = 50 \in \mathbb{Z}_p^*$ ($\rightsquigarrow z^{-1} = 99$) und erhalten dann

$$\begin{aligned} \gamma &= (170^{50} \bmod 7879) \bmod 101 \\ &= 2518 \bmod 101 \\ &= 94 \\ \delta &= (22 + 75 \cdot 94) \cdot 99 \bmod 101 \\ &= 97 \quad (\rightsquigarrow \delta^{-1} = 25) \end{aligned}$$

d.h. $\text{sig}(p, q, \alpha, z, x) = (94, 97)$, wobei $\hat{k} = (p, q, \alpha, a)$

Um diese Signatur zu prüfen berechnen wir:

$$\begin{aligned} e &= x\delta^{-1} \bmod q \\ &= 22 \cdot 25 \bmod 101 \\ &= 45 \\ d &= \gamma\delta^{-1} \bmod q \\ &= 94 \cdot 25 \bmod 101 \\ &= 27 \end{aligned}$$

$$\rightsquigarrow (\alpha^e \beta^d \bmod p) \bmod q = (170^{45} 4547^{27} \bmod 7879) \bmod 101 = 94. \quad \triangleleft$$

3.5 ECDSA (Elliptic Curve DSA)

Im Jahr 2000 als FIPS 186-2 als Standard deklariert.

Sei E eine elliptische Kurve über einem endlichen Körper \mathbb{F}_{p^n} . Sei $A \in E$ ein Punkt der Ordnung q (q prim), so dass das Diskrete-Logarithmus-Problem zur Basis A in E schwierig ist. Zudem sei $h: \{0, 1\}^* \rightarrow \mathbb{Z}_q$ eine kryptografische Hashfunktion.

Textraum: $X = \{0, 1\}^*$,

Signaturraum: $Y = \mathbb{Z}_q^* \times \mathbb{Z}_q^*$,

Signierschlüssel: $\hat{k} = (E, q, A, m)$, $m \in \mathbb{Z}_q^*$,

Verifikationsschlüssel: $k = (E, q, A, B)$, wobei $B = m \cdot A$.

Signaturerstellung: Um einen Text $x \in X$ zu signieren, wählt der Signierer zufällig eine geheime Zahl $z \in \mathbb{Z}_q^*$ und berechnet

$$\text{sig}(\hat{k}, x, z) = (\gamma, \delta),$$

wobei

$$\begin{aligned}(u, v) &:= zA \\ \gamma &:= u \bmod q \\ \delta &:= (h(x) + m\gamma)z^{-1} \bmod q\end{aligned}$$

Hierbei wird u als eine Zahl in $\{0, \dots, p^n - 1\}$ interpretiert. Falls $\gamma = 0$ oder $\delta = 0$ ist, muss eine neue Zufallszahl z gewählt und der Vorgang wiederholt werden.

Verifikation: $ver(k, x, \gamma, \delta) = 1$, falls $u \bmod q = \gamma$ ist, wobei

$$\begin{aligned}e &:= h(x)\delta^{-1} \bmod q \\ d &:= \gamma\delta^{-1} \bmod q \\ (u, v) &:= eA + dB\end{aligned}$$

Korrektheit der Verifikation beim ECDSA:

$$\begin{aligned}(u, v) &= eA + dB \\ &= (h(x)\delta^{-1})A + (\gamma\delta^{-1})mA \\ &= (h(x) + m\gamma)\delta^{-1}A \\ &= zA \text{ (da } (h(x) + m\gamma)\delta^{-1} \equiv_q z\text{)}\end{aligned}$$

Beispiel 64. Sei E über \mathbb{Z}_{11} definiert durch $y^2 = x^3 + x + 6$. Wir wählen $A = (2, 7)$, $m = 7 \rightarrow p = 11, q = 13, B = 7A = (7, 2)$.

Um einen Text x mit dem Hashwert $h(x) = 4$ unter Verwendung des Signierschlüssels $\hat{k} = (E, q, A, m)$ und der Zufallszahl $z = 3$ signieren, berechnet Alice

$$\begin{aligned}(u, v) &:= zA = 3 \cdot (2, 7) = (8, 3) \\ \gamma &:= u \bmod q = 8 \\ \delta &:= (4 + 7 \cdot 8)3^{-1} \bmod 13 = 7\end{aligned}$$

und erhält die Signatur $sig(\hat{k}, z, x) = (8, 7)$. Um diese Signatur mit dem Verifikationsschlüssel $k = (E, q, A, B)$ zu überprüfen, berechnet Bob

$$\begin{aligned}e &:= h(x)\delta^{-1} \bmod q = 4 \cdot 7^{-1} \bmod 13 = 4 \cdot 2 \bmod 13 = 8 \\ d &:= \gamma\delta^{-1} \bmod q = 8 \cdot 2 \bmod 13 = 3 \\ (u, v) &:= eA + dB = 8 \cdot (2, 7) + 3 \cdot (7, 2) = (8, 3)\end{aligned}$$

und testet die Kongruenz $u \equiv_q \gamma$. ◁

3.6 One-time Signatur (Lamport 1979)

Leslie Lamport konnte zeigen, dass sich digitale Signaturen auf der Basis einer Einwegfunktion f konstruieren lassen. Damit die Signatur allerdings sicher ist, muss für jeden Text ein neues Schlüsselpaar (\hat{k}, k) generiert werden, d.h. der Signierschlüssel \hat{k} darf nur zum Signieren eines einzelnen Textes verwendet werden.

Seien U und V endliche Mengen und sei $f : U \rightarrow V$ eine Funktion. Zudem sei $\ell \geq 1$ die vorgegebene Textlänge, d.h. der Textraum ist $X = \{0, 1\}^\ell$. Der Signaturraum ist dann $Y = U^\ell$.

Um ein Schlüsselpaar (\hat{k}, k) zu generieren, wird zufällig eine Folge von 2ℓ Elementen $u_{i,b}$ für $i = 1, \dots, \ell$ und $b = 0, 1$ aus U gewählt und der Signierschlüssel $\hat{k} = \begin{pmatrix} u_{1,0} \dots u_{\ell,0} \\ u_{1,1} \dots u_{\ell,1} \end{pmatrix}$ gebildet.

Der zugehörige Verifikationsschlüssel ist dann $k = \begin{pmatrix} v_{1,0} \dots v_{\ell,0} \\ v_{1,1} \dots v_{\ell,1} \end{pmatrix}$ mit $v_{i,b} = f(u_{i,b})$ für alle $i = 1, \dots, \ell$ und $b = 0, 1$.

Signaturerstellung: Die Signatur für einen Text $x = x_1 \dots x_\ell \in X$ ist

$$\text{sig}(\hat{k}, x) = (u_{1,x_1}, \dots, u_{\ell,x_\ell}).$$

Verifikation: Für eine Signatur $y = (u_1, \dots, u_\ell)$ und einen Text $x = x_1 \dots x_\ell$ gilt

$$\text{ver}(k, x, y) := \begin{cases} 1, & f(u_i) = v_{i,x_i} \text{ für } i = 1, \dots, \ell, \\ 0, & \text{sonst.} \end{cases}$$

Beispiel 65. Wir wählen als Einwegfunktion eine Funktion der Form $f : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_p^*$ mit $f(u) = g^u \bmod p$, wobei g ein Erzeuger von \mathbb{Z}_p^* ist.

Z.B. sei $p = 7879$ und $g = 3$, also $f(u) = 3^u \bmod 7879$. Weiter sei $\ell = 3$.

Dann erhalten wir für den zufällig gewählten Signierschlüssel $\hat{k} = \begin{pmatrix} 5831 & 4285 & 2467 \\ 803 & 735 & 6449 \end{pmatrix}$ den zugehörigen Verifikationsschlüssel $k = \begin{pmatrix} 2009 & 268 & 4721 \\ 4672 & 3810 & 5731 \end{pmatrix}$. Die Signatur y für den Text $x = 110$ ist dann

$$y = \text{sig}(\hat{k}, x) = (u_{1,1}, u_{2,1}, u_{3,0}) = (803, 735, 2467).$$

Für diese Signatur $y = (u_1, u_2, u_3)$ ist $\text{ver}(k, x, y) = 1$, da $f(u_i) = v_{i,x_i}$ für $i = 1, 2, 3$ gilt:

$$i = 1 : f(u_1) = f(803) = 3^{803} \bmod 7879 = 4672 = v_{1,x_1}$$

$$i = 2 : f(u_2) = f(735) = 3^{735} \bmod 7879 = 3810 = v_{2,x_2}$$

$$i = 3 : f(u_3) = f(2467) = 3^{2467} \bmod 7879 = 4721 = v_{3,x_3} \quad \triangleleft$$

Ähnlich wie bei MACs können wir einen Angriff gegen ein digitales Signaturverfahren wie folgt modellieren. Hierbei nehmen wir an, dass der Gegner die Texte, deren Signaturen er kennt, adaptiv wählen kann (existentielle Fälschung bei adaptiv wählbaren Texten).

Definition 66. Sei $0 \leq \varepsilon \leq 1$ und sei $q \in \mathbb{N}$. Ein (ε, q) -**Fälscher** für ein digitales Signaturverfahren ist ein probabilistischer Algorithmus \mathcal{A} , der bei Eingabe eines Verifikationsschlüssels k (wobei das Schlüsselpaar (\hat{k}, k) zufällig gewählt wird) nach den Signaturen $y_i = \text{sig}(\hat{k}, x_i)$ von q Texten x_1, \dots, x_q fragt und mit Wahrscheinlichkeit mindestens ε eine Fälschung (x, y) mit $\text{ver}(k, x, y) = 1$ und $x \notin \{x_1, \dots, x_q\}$ ausgibt.

Satz 67. Sei $f : U \rightarrow V$ eine Funktion. Falls für die zugehörige one-time Signatur ein $(\varepsilon, 0)$ -Fälscher **Lamport-Fälschung**(k) existiert, dann lässt sich für ein zufällig gewähltes $u \in_R U$ mit Wahrscheinlichkeit mindestens $\varepsilon/2$ ein Urbild von $v = f(u)$ bestimmen.

Beweis. Betrachte folgenden probabilistischen Algorithmus **Lamport-Urbild**(v).

Prozedur Lamport-Urbild(v)

```

1 wähle zufällig ein Indexpaar  $(j, a)$  und setze  $v_{j,a} := v$ 
2 for all  $(i, b) \in [\ell] \times \{0, 1\} \setminus \{(j, a)\}$  do
3   wähle zufällig  $u_{i,b} \in_R U$  und setze  $v_{i,b} := f(u_{i,b})$ 
4  $k := \begin{pmatrix} v_{1,0} \dots v_{\ell,0} \\ v_{1,1} \dots v_{\ell,1} \end{pmatrix}$ 
5  $(x_1 \dots x_\ell, (u_1, \dots, u_\ell)) =: \text{Lamport-Fälschung}(k)$ 
6 if  $f(u_j) = v$  then output( $u_j$ ) else output(?)
```

Wie üblich bezeichnen wir die Zufallsvariablen, die die Wahl von v, j, a, k und $(x, y) = (x_1 \dots x_\ell, (u_1, \dots, u_\ell))$ beschreiben, mit entsprechenden Großbuchstaben. Dann müssen wir zeigen, dass $\mathcal{U}_{\mathcal{J}}$ mit Wahrscheinlichkeit mindestens $\varepsilon/2$ ein f -Urbild von \mathcal{V} ist, wobei \mathcal{V} die Wahl von $v = f(u)$ für ein zufällig gewähltes $u \in_R U$ beschreibt.

Da die Verteilung von \mathcal{K} identisch zur Schlüsselgenerierung der Lamport-Signatur und **Lamport-Fälschung** ein $(\varepsilon, 0)$ -Fälscher ist, folgt

$$\Pr[\text{ver}(\mathcal{K}, \mathcal{X}, \mathcal{Y}) = 1] \geq \varepsilon.$$

Da zudem \mathcal{K} (und damit auch $(\mathcal{X}, \mathcal{Y})$) unabhängig von $(\mathcal{J}, \mathcal{A})$ und auch \mathcal{J} und \mathcal{A} unabhängig voneinander sind, ist \mathcal{A} von $(\mathcal{J}, \mathcal{K}, \mathcal{X}, \mathcal{Y})$ und damit auch von $\mathcal{X}_{\mathcal{J}}$ unabhängig. Sei p die Erfolgswk von **Lamport-Urbild** bei Eingabe \mathcal{V} . Wegen

$$\text{ver}(k, x_1 \dots x_\ell, (u_1, \dots, u_\ell)) = 1 \wedge x_j = a \implies f(u_j) = v_{j,x_j} = v_{j,a} = v$$

folgt nun

$$\begin{aligned} p &\geq \Pr[\text{ver}(\mathcal{K}, \mathcal{X}, \mathcal{Y}) = 1 \wedge \mathcal{X}_{\mathcal{J}} = \mathcal{A}] \\ &= \Pr[\text{ver}(\mathcal{K}, \mathcal{X}, \mathcal{Y}) = 1] \underbrace{\Pr[\mathcal{X}_{\mathcal{J}} = \mathcal{A} \mid \text{ver}(\mathcal{K}, \mathcal{X}, \mathcal{Y}) = 1]}_{1/2} = \varepsilon/2. \end{aligned}$$

□

Als nächstes untersuchen wir die Sicherheit der Lamport-Signatur, falls der Gegner in der Lage ist, sich für einen beliebigen Text x' seiner Wahl eine gültige Signatur y' zu beschaffen.

Satz 68. *Sei $f : U \rightarrow V$ eine Funktion. Falls für die zugehörige one-time Signatur ein $(\varepsilon, 1)$ -Fälscher **Lamport-Fälschung'**(k) existiert, so lässt sich für ein zufällig gewähltes $u \in_R U$ mit Wahrscheinlichkeit $\geq \varepsilon/2\ell$ ein f -Urbild von $v = f(u)$ bestimmen.*

Für den Beweis betrachten wir folgenden probabilistischen Algorithmus

Prozedur Lamport-Urbild'(v)

```

1 wähle zufällig ein Indexpaar  $(j, a)$  und setze  $v_{j,a} := v$ 
2 for all  $(i, b) \neq (j, a)$  do
3   wähle zufällig  $u_{i,b} \in_R U$  und setze  $v_{i,b} := f(u_{i,b})$ 
4  $k := \begin{pmatrix} v_{1,0} \dots v_{\ell,0} \\ v_{1,1} \dots v_{\ell,1} \end{pmatrix}$ 
5 simuliere Lamport-Fälschung'( $k$ ) und beantworte die Frage  $x'$  mit
    $u_{1,x'_1}, \dots, u_{\ell,x'_\ell}$  (falls  $x'_j = a$  ist, brich ab und gib ? aus); sei
    $(x, y) = (x_1 \dots x_\ell, (u_1, \dots, u_\ell))$  die erzeugte Ausgabe
6 if  $f(u_j) = v$  then output( $u_j$ ) else output(?)
```

und zeigen, dass **Lamport-Urbild'** für ein zufällig gewähltes $u \in_R U$ bei Eingabe $v = f(u)$ mit Wahrscheinlichkeit $\geq \varepsilon/2\ell$ ein f -Urbild von v ausgibt.

Beweis. Sei p' die Erfolgswk von **Lamport-Urbild'** bei Eingabe \mathcal{V} . Es ist klar, dass u_j im Fall $\text{ver}(k, x_1 \dots x_\ell, (u_1, \dots, u_\ell)) = 1 \wedge x'_j \neq x_j = a$ ein Urbild von v ist. Allerdings kann **Lamport-Urbild'** nur dann die Frage nach der Signatur von x' beantworten, wenn $x'_j \neq a$ ist. Da also die Simulation von **Lamport-Fälschung'**(k) teilweise abgebrochen wird (und die Abbruchbedingung von (j, a) abhängt), können wir nicht mehr davon ausgehen, dass diese Simulation mit Wahrscheinlichkeit ε eine Fälschung (x, y) liefert und (x, y) unabhängig von (j, a) ist.

Durch eine einfache Modifikation von **Lamport-Urbild'**(v) erhalten wir jedoch eine Prozedur **Lamport-Urbild*** (ohne Eingabe), deren Ausgabeverhalten mit der von **Lamport-Urbild'**(V) identisch ist, und von der wir zeigen können, dass sie mit Wahrscheinlichkeit $p^* \geq \varepsilon/2\ell$ Erfolg hat (also nicht Fragezeichen ausgibt).

Prozedur **Lamport-Urbild***

```

1 wähle zufällig ein Indexpaar  $(j, a)$ 
2 for all  $(i, b)$  do wähle zufällig  $u_{i,b} \in_R U$  und setze  $v_{i,b} := f(u_{i,b})$ 
3  $k := \begin{pmatrix} v_{1,0} \dots v_{\ell,0} \\ v_{1,1} \dots v_{\ell,1} \end{pmatrix}$ 
4 simuliere Lamport-Fälschung'( $k$ ) und beantworte die Frage  $x'$  mit
    $u_{1,x'_1}, \dots, u_{\ell,x'_\ell}$ ; sei  $(x, y) = (x_1 \dots x_\ell, (u_1, \dots, u_\ell))$  die erzeugte Ausgabe
5 if  $f(u_j) = v_{j,a} \wedge x'_j \neq a$  then output( $u_j$ ) else output(?)
```

Im Unterschied zu **Lamport-Urbild'**(v) wählt sich **Lamport-Urbild*** also die Eingabe $v = v_{j,a}$ gemäß der Verteilung von \mathcal{V} selbst und kennt daher auch ein Urbild $u_{j,a}$ von $v_{j,a}$. Somit kann **Lamport-Urbild*** bei der Simulation von **Lamport-Fälschung'**(k) die Frage nach der Signatur von x' auch im Fall $x'_j = a$ beantworten. Die Bedingung für die Ausgabe von u_j ist jedoch bei beiden Prozeduren dieselbe, d.h. die Ausgabe von **Lamport-Urbild*** hat dieselbe Verteilung wie die von **Lamport-Urbild'**(\mathcal{V}) und somit gilt $p' = p^*$. Der einzige Unterschied ist, dass immer wenn **Lamport-Urbild'**(v) in Zeile 4 ein Fragezeichen ausgibt, **Lamport-Urbild*** dies erst in Zeile 5 tut. Da in der Prozedur **Lamport-Urbild*** die ZV $(\mathcal{J}, \mathcal{A})$ unabhängig von $(\mathcal{K}, \mathcal{X}', \mathcal{X}, \mathcal{Y})$ ist, folgt nun

$$\begin{aligned}
p^* &= \Pr[f(\mathcal{U}_{\mathcal{J}}) = \mathcal{V}_{\mathcal{J},\mathcal{A}} \wedge \mathcal{X}'_{\mathcal{J}} \neq \mathcal{A}] \\
&\geq \Pr[\text{ver}(\mathcal{K}, \mathcal{X}, \mathcal{Y}) = 1 \wedge \mathcal{X}_{\mathcal{J}} = \mathcal{A}] \wedge \mathcal{X}'_{\mathcal{J}} \neq \mathcal{A}] \\
&= \Pr[\text{ver}(\mathcal{K}, \mathcal{X}, \mathcal{Y}) = 1] \underbrace{\Pr[\mathcal{X}'_{\mathcal{J}} \neq \mathcal{X}_{\mathcal{J}} = \mathcal{A} \mid \text{ver}(\mathcal{K}, \mathcal{X}, \mathcal{Y}) = 1]}_{\geq 1/2\ell} \geq \varepsilon/2\ell. \quad \square
\end{aligned}$$

Die Lamport-Signatur hat aus praktischer Sicht einige Nachteile, die sich jedoch teilweise beheben lassen (siehe Übungen). So lässt sich sowohl die Länge des privaten Signierschlüssels (mittels Pseudozufallsgeneratoren) als auch des öffentlichen Verifikationsschlüssels (mittels Hash-Listen) verringern. Zudem können bei Verwendung von Hash-Bäumen mit demselben Schlüsselpaar auch mehrere Nachrichten signiert und verifiziert werden.

3.7 Full Domain Hash (FDH) Signaturen

Sei $\mathcal{F} = \{f_k \mid k \in K\}$ eine Familie von Falltür-Permutationen auf einer Menge U , d.h. es lassen sich (zufällig) Schlüsselpaare $(\hat{k}, k) \in K \times K$ generieren, so dass gilt:

- $f_{\hat{k}}(f_k(u)) = u$ für alle $u \in U$.
- f_k ist eine Einweg-Permutation auf U , d.h. für ein zufällig gewähltes Schlüsselpaar $(\hat{k}, k) \in K \times K$ und ein zufällig gewähltes $v \in U$ ist es schwer, ohne Kenntnis von \hat{k} ein Urbild u mit $f_k(u) = v$ zu finden (genauer: jedem effizienten Gegner gelingt dies nur mit vernachlässigbarer Wahrscheinlichkeit).

Weiter sei $h : \{0, 1\}^* \rightarrow U$ eine Funktion.

Die auf \mathcal{F} und h basierende FDH-Signatur funktioniert wie folgt. Zuerst wird ein Schlüsselpaar $(\hat{k}, k) \in K \times K$ generiert, wobei \hat{k} als Signierschlüssel und k als Verifikationsschlüssel fungiert. Der Textraum ist $X = \{0, 1\}^*$ und der Signaturenraum ist U .

Signaturerstellung: Die Signatur für einen Text $x \in X$ ist

$$\text{sig}(\hat{k}, x) = f_{\hat{k}}(h(x)).$$

Verifikation: Für eine Signatur $y \in U$ und einen Text $x \in \{0, 1\}^*$ gilt

$$\text{ver}(k, x, y) := \begin{cases} 1, & f_k(y) = h(x), \\ 0, & \text{sonst.} \end{cases}$$

Z.B. beruht das RSA-Signaturverfahren in Verbindung mit einer Hashfunktion auf diesem Prinzip. Ein Problem hierbei ist allerdings, dass die benutzten RSA-Falltür-Permutationen einen Definitionsbereich der Größe 2^{1024} haben, um eine ausreichend große Sicherheit zu erreichen, wogegen die benutzten Hashfunktionen nur eine Länge von 160 Bit haben. In der Praxis behilft man sich damit, dass man die 160-Bit-Hashwerte durch eine deterministische Paddingfunktion auf 1024-Bit aufbläht, was die Sicherheit allerdings beeinträchtigen kann.

Sicherheitsanalyse der FDH-Signatur im ZOM

Bei Verwendung einer Zufallsfunktion $G : \{0, 1\}^* \rightarrow U$ (vgl. Zufalls-Orakel-Modell, ZOM) anstelle von h lässt sich die Fälschungssicherheit der resultierenden FDH-Signatur aus der Falltüreigenschaft von \mathcal{F} herleiten. Das ZOM modelliert eine Hashfunktion mit optimalen kryptografischen Eigenschaften, d.h. die Zufallsvariablen $U_x = G(x)$ sind stochastisch unabhängig und gleichverteilt auf U . Zudem füllt der Wertebereich von G den gesamten Definitionsbereich der Funktionen f_k aus (full domain hash).

Wir betrachten zuerst den Fall einer existentiellen Fälschung bei bekanntem Verifikationsschlüssel, d.h. der Gegner muss eine Fälschung (x, y) mit $\text{ver}(k, x, y) = 1$ produzieren ohne auch nur eine Signatur y' für einen Text x' zu kennen.

Sei **FDH-Fälschung** ein probabilistischer Algorithmus, der für einen zufällig generierten Verifikationsschlüssel k mit Wahrscheinlichkeit ε eine existentielle Fälschung (x, y) mit $f_k(y) = G(x)$ ausgibt. Dabei nehmen wir an, dass **FDH-Fälschung** eine Folge von q verschiedenen Fragen x_1, \dots, x_q an G stellt. Es ist klar, dass ein solcher Angriff im Fall $x \notin \{x_1, \dots, x_q\}$ mit der Wahrscheinlichkeit $\varepsilon = 1/\|U\|$ gelingt. Da diese Erfolgswk durch Ausgabe eines beliebigen Paares (x, y) bereits mit $q = 0$ Fragen an G erreicht wird, können wir zudem annehmen, dass $x \in \{x_1, \dots, x_q\}$ enthalten ist (sofern $q \geq 1$ ist).

Betrachte folgenden Invertieralgorithmus für f_k .

Prozedur $\text{FDH-Invert}(k, v)$

- 1 wähle zufällig $j \in_R \{1, \dots, q\}$
 - 2 simuliere **FDH-Fälschung**(k) und beantworte dabei die Frage x_i im Fall $i = j$ durch $v_j = v$ und sonst durch ein zufällig gewähltes $v_i \in_R U$; sei (x, y) die erzeugte Ausgabe
 - 3 **if** $f_k(y) = v$ **then** **output**(y) **else** **output**(?)
-

Satz 69. Falls **FDH-Fälschung**(k) für einen zufällig gewählten Verifikationsschlüssel k mit Wahrscheinlichkeit ε eine Fälschung (x, y) mit $f_k(y) = G(x)$ ausgibt und dabei $q \geq 1$ Fragen an G stellt, so gibt **FDH-Invert**(k, v) für einen zufälligen Verifikationsschlüssel k und ein zufälliges $v \in_R U$ mit Wahrscheinlichkeit $\geq \varepsilon/q$ ein f_k -Urbild von v aus.

Beweis. Seien $\mathcal{J}, \mathcal{K}, \mathcal{U}, \mathcal{V}, \mathcal{X}, \mathcal{X}_1, \dots, \mathcal{X}_q$ Zufallsvariablen, die die Wahl von $j, k, u, v, x, x_1, \dots, x_q$ beschreiben. Da die Eingabe $v = v_j$ und die Strings $v_i, i \neq j$, unabhängig unter Gleichverteilung aus U gewählt werden, sind die Antworten auf die Fragen x_1, \dots, x_q unabhängig gleichverteilt, was dem ZOM entspricht. Daher liefert die Simulation von **FDH-Fälschung**(k) für einen zufällig generierten Verifikationsschlüssel k mit Wahrscheinlichkeit ε eine Fälschung (x, y) mit $f_k(y) = G(x)$, d.h. es gilt $\Pr[f_{\mathcal{K}}(\mathcal{Y}) = G(\mathcal{X})] = \varepsilon$.

Wir wollen zeigen, dass $\Pr[f_{\mathcal{K}}(\mathcal{Y}) = V] \geq \varepsilon/q$ ist. Wegen $x \in \{x_1, \dots, x_q\}$ existiert ein $i \in \{1, \dots, q\}$ mit $x = x_i$ und die Gleichheit $f_k(y) = G(x)$ impliziert $f_k(y) = G(x_i) = v_i$. Folglich gilt die Implikation

$$f_k(y) = G(x) \wedge j = i \Rightarrow f_k(y) = v_j = v$$

und es folgt

$$\Pr[f_{\mathcal{K}}(\mathcal{Y}) = \mathcal{V}] \geq \Pr[f_{\mathcal{K}}(\mathcal{Y}) = G(\mathcal{X}) \wedge \mathcal{J} = \mathcal{I}]$$

Zudem wird $j \in_R \{1, \dots, q\}$ zufällig gewählt und die Fragen x_1, \dots, x_q werden unabhängig voneinander durch zufällige $v_1, \dots, v_q \in_R U$ beantwortet (nach Voraussetzung trifft dies auch auf $v_j = v$ zu). Daher erhält **FDH-Fälschung** weder durch k noch durch die Antworten v_1, \dots, v_q irgendeine Information über j . Folglich sind neben der Eingabe \mathcal{K} auch die Ausgabe $(\mathcal{X}, \mathcal{Y})$ und somit auch die Zufallsvariable \mathcal{I} , die den Index $i \in \{1, \dots, q\}$ mit $x = x_i$ bestimmt, stochastisch unabhängig von \mathcal{J} . Daher folgt

$$\begin{aligned} \Pr[f_{\mathcal{K}}(\mathcal{Y}) = \mathcal{V}] &\geq \Pr[f_{\mathcal{K}}(\mathcal{Y}) = G(\mathcal{X}) \wedge \mathcal{J} = \mathcal{I}] \\ &= \underbrace{\Pr[f_{\mathcal{K}}(\mathcal{Y}) = G(\mathcal{X})]}_{\varepsilon} \underbrace{\Pr[\mathcal{J} = \mathcal{I} \mid f_{\mathcal{K}}(\mathcal{Y}) = G(\mathcal{X})]}_{1/q} \\ &= \Pr[f_{\mathcal{K}}(\mathcal{Y}) = G(\mathcal{X})]/q = \varepsilon/q \end{aligned}$$

□

Falls sich also f_k nur mit einer vernachlässigbaren Wahrscheinlichkeit $\leq \varepsilon'$ effizient invertieren lässt, so gelingt einem ähnlich effizienten Gegner, der nicht mehr als q Hashwertberechnungen durchführt, im ZOM höchstens mit einer (ebenfalls vernachlässigbaren) Wahrscheinlichkeit $\varepsilon \leq q\varepsilon'$ eine existentielle Fälschung für die FDH-Signatur.

Als nächstes beweisen wir die Fälschungssicherheit der FDH-Signatur im ZOM gegenüber einem existentiellen Angriff mit adaptiv gewählten Texten.

Sei **FDH-Fälschung'** ein probabilistischer Algorithmus, der für einen zufällig generierten Verifikationsschlüssel k mit Wahrscheinlichkeit ε eine existentielle Fälschung (x, y) mit $f_k(y) = G(x)$ ausgibt und insgesamt für q Texte x_1, \dots, x_q den Wert $G(x_i)$ oder die

Signatur $\text{sig}(\hat{k}, x_i)$ erfragt. Dabei können wir o.B.d.A. annehmen, dass **FDH-Fälschung'** zwar den G -Wert aber nicht die Signatur von x erfragt und vor jeder Signaturfrage den G -Wert des betreffenden Textes erfragt.

Satz 70. Falls **FDH-Fälschung'**(k) für einen zufällig gewählten Verifikationsschlüssel k mit Wahrscheinlichkeit ε eine Fälschung (x, y) mit $f_k(y) = G(x)$ berechnet und dabei für q Texte x_i den Wert $G(x_i)$ sowie im Fall $x_i \neq x$ evtl. auch die Signatur $\text{sig}(\hat{k}, x_i)$ erfragt, so lässt sich für einen zufälligen Verifikationsschlüssel k und ein zufälliges $v \in_R U$ mit Wahrscheinlichkeit $\geq \varepsilon/q$ ein f_k -Urbild von v bestimmen.

Für den Beweis (siehe Übungen) betrachten wir folgenden probabilistischen Algorithmus

Prozedur **FDH-Invert'**(k, v)

- 1 wähle zufällig $j \in_R \{1, \dots, q\}$
 - 2 simuliere **FDH-Fälschung'**(k) und beantworte dabei jede Frage x_i an G im Fall $i = j$ durch $v_j = v$ und sonst durch $v_i = f_k(u_i)$, wobei u_i zufällig aus U gewählt wird; falls später die Signatur von x_i erfragt wird, gib u_i als Antwort (falls $i = j$ ist, brich ab und gib ? aus); sei (x, y) die erzeugte Ausgabe
 - 3 **if** $f_k(y) = v$ **then** **output**(y) **else** **output**(?)
-

und zeigen für einen zufälligen Verifikationsschlüssel k und ein zufälliges $v \in_R U$,

$$\Pr[\text{FDH-Invert}'(k, v) \text{ findet ein } f_k\text{-Urbild von } v] \geq \varepsilon/q.$$

Beweis. Da die Frage nach der Signatur von x_i im Fall $i = j$ unbeantwortet bleibt, können wir nicht mehr schließen, dass die Simulation von **FDH-Fälschung'**(k) mit Wahrscheinlichkeit ε eine Fälschung (x, y) mit $f_k(y) = G(x)$ liefert. Wir können aber eine Prozedur **FDH-Invert***(k) angeben, die nur k als Eingabe erhält, so dass die Ausgaben von **FDH-Invert***(\mathcal{K}) und von **FDH-Invert'**(\mathcal{K}, \mathcal{V}) identisch verteilt sind, was

$$\Pr[\text{FDH-Invert}'(\mathcal{K}, \mathcal{V}) \neq ?] = \Pr[\text{FDH-Invert}^*(\mathcal{K}) \neq ?]$$

impliziert. Zudem lässt sich zeigen, dass

$$\Pr[\text{FDH-Invert}^*(\mathcal{K}) \neq ?] \geq \varepsilon/q$$

gilt (siehe Übungen). □