Vorlesungsskript

Kryptologie

Sommersemester 2016

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

20. Juni 2016

Inhaltsverzeichnis

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)

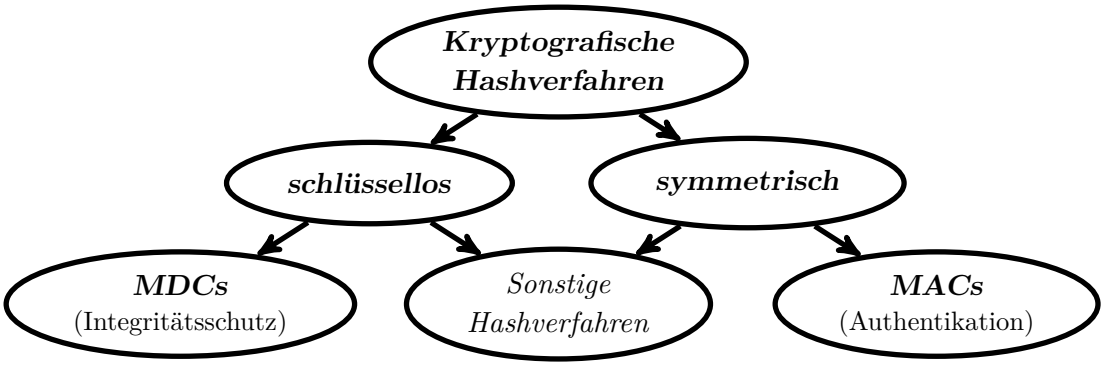

Abbildung 1.1: Eine grobe Einteilung von kryptografischen Hashverfahren.

  - 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-

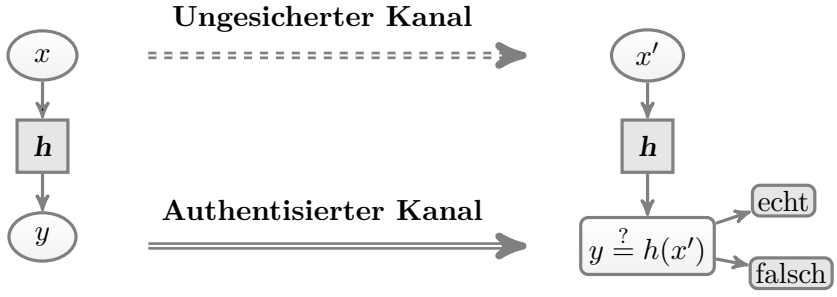

Abbildung 1.2: Einsatz eines MDC h zur Überprüfung der Integrität eines Datensatzes x.

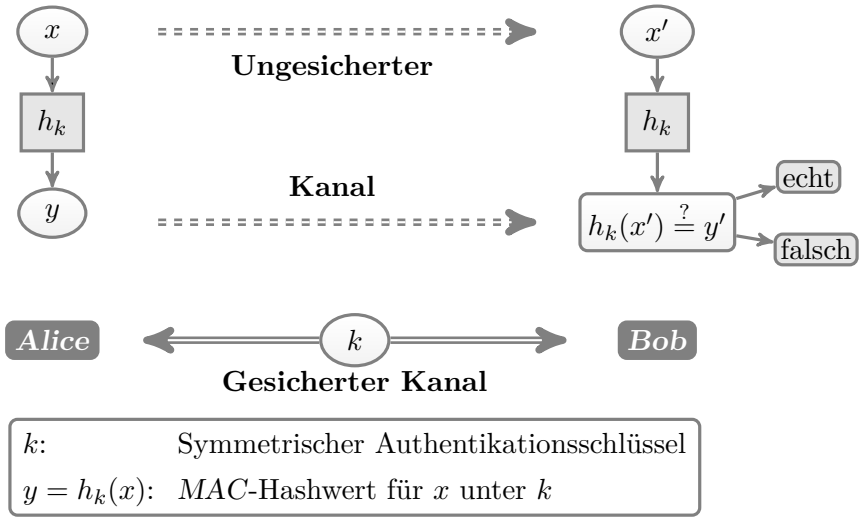

Abbildung 1.3: Verwendung eines MAC zur Nachrichtenauthentikation.

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 **Authentikationscode** bezeichnet. Daneben gibt es auch Hashverfahren mit asymmetrischen Schlüsseln. Diese werden jedoch der Rubrik der Signaturverfahren zugeordnet, da mit ihnen ausschließlich digitale Unterschriften 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 er den zugehörigen *MAC*-Hashwert $y = h_k(x)$ und fügt diesen der Nachricht x hinzu. Bob überprüft die Echtheit der empfangenen Nachricht (x', y'), indem sie ihrerseits den zu x' gehörigen Hashwert $h_k(x')$ berechnet und das Ergebnis mit y' vergleicht. Der geheime Authentikationsschlüssel k muss hierbei genau wie bei einem symmetrischen Kryptosystem über einen gesicherten Kanal vereinbart werden.

Indem Alice seine Nachricht x um den Hashwert $y = h_k(x)$ ergänzt, gibt er 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 $h\colon X \to Y$ 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. $h\colon X \to Y$ 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. $h\colon X \to Y$ 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. $h\colon X \to Y$.

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 $h\colon X \to Y$ 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 ∈ X
2 x' := A(x)
3 if x' ≠ ? then return(x, x') else return(?)
```

Abbildung 1.4: Reduktion des Kollisionsroblems 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. □

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

Satz 2. *Sei $h\colon X \to Y$ 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 $\mathcal{C} = \{h^{-1}(y) \mid y \in Y\}$. Dann hat B eine Erfolgswahrscheinlichkeit von

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

□

1.2.2 Das Zufallsorakelmodell (ZOM)

Das ZOM dient dazu, den Aufwand verschiedener Angriffe auf eine Hashfunktion $h\colon X \to Y$ nach oben abzuschätzen. Sind X und Y vorgegeben, so können wir eine Hashfunktion $h\colon X \to Y$ 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 n^m 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 ∈ X
2 y := h(x)
3 x' := A(y)
4 if x ≠ 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, ..., x_q} ⊆ X
2 for each x_i ∈ X_0 do
3   if h(x_i) = y then return(x_i)
4 return(?)
```

Abbildung 1.6: Bestimmung eines Urbilds für einen Hashwert

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

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

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 $X_0 = \{x_1, \ldots, x_q\}$. Für $i = 1, \ldots, q$ bezeichne E_i das Ereignis "$h(x_i) = y$". Nach Proposition 3 sind diese Ereignisse stochastisch unabhängig und ihre Wahrscheinlichkeit ist $\Pr[E_i] = 1/m$ $(i = 1, \ldots, q)$. Also folgt

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

□

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 (dies erscheint zwar verblüffend, wird aber durch die Praxis mehr als bestätigt).

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

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

eine Kugel zweimal 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 $X_0 = \{x_1, \ldots, x_q\}$. Für $i = 1, \ldots, q$ bezeichne E_i das Ereignis

$$\text{“}h(x_i) \notin \{h(x_1, \ldots, h(x_{i-1})\}\text{.”}$$

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

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

Dies führt auf die Erfolgswahrscheinlichkeit

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

□

Prozedur `FindSecondPreimage`(h, x, q)

```
1  y := h(x)
2  wähle eine beliebige Menge X_0 = {x_1, ..., x_{q-1}} ⊆ X − {x}
3  for each x_i ∈ 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, ..., x_q} ⊆ X
2 for each x_i ∈ X_0 do y_i := h(x_i)
3 if ∃ i ≠ j : y_i = y_j then return(x_i, x_j) else return(?)
```

Abbildung 1.8: Bestimmung eines Kollisionspaares

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 $h\colon \{0,1\}^n \to \{0,1\}^m$ 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

$$h\colon \{0,1\}^{m+t} \to \{0,1\}^m$$

eine kollisionsresistente Hashfunktion

$$\hat{h}\colon \{0,1\}^* \to \{0,1\}^l$$

konstruieren lässt. Hierzu betrachten wir folgende kanonische Konstruktionsmethode.

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

$$y\colon \{0,1\}^* \to \bigcup_{r \geq 1} \{0,1\}^{rt}$$

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 $i = 1, \ldots, r$. Berechne eine Folge $z_0, \ldots, z_r$ von Strings $z_i \in \{0,1\}^m$ wie folgt:

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

Optionale Ausgabetransformation: Berechne den Hashwert $\hat{h}(x) = g(z_r)$, wobei $g\colon \{0,1\}^m \to \{0,1\}^l$ 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 $y\colon \{0,1\}^* \to \{0,1\}^*$ 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. Angenommen, es gelingt, ein Kollisionspaar $x, \tilde{x}$ für $\hat{h}$ mit $\hat{h}(x) = \hat{h}(\tilde{x})$ zu finden. Sei

$$y(x) = y_1 y_2 \dots y_{k-1} y_k \text{ und } y(\tilde{x}) = \tilde{y}_1 \tilde{y}_2 \dots \tilde{y}_{l-1} \tilde{y}_l \text{ mit } k \leq l.$$

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 $x = x_1 x_2 \dots x_{k-1} x_k$ 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 $y(x) = 0x0^d 1 bin_{t-1}(d)$ und für $k > 1$ ist $y(x) = y_1 \cdots y_{k+1}$, wobei

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

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 $y(x) = y_1 y_2 \dots y_{k+1}$ kein Suffix von $y(\tilde{x}) = \tilde{y}_1 \tilde{y}_2 \dots \tilde{y}_{l+1}$ 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 $|x| \equiv_{t-1} |x'|$. 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. □

Nun kommen wir zum Fall $t = 1$. Sei y die durch $y(x) := 11 f(x)$ definierte Funktion, wobei f wie folgt definiert ist:

$$f(x_1, \dots, x_n) = f(x_1) \dots f(x_2) \text{ mit } f(0) = 0 \text{ und } f(1) = 01.$$

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 (***M**essage **D**igest*) 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
$X \leftarrow\!\!\shortmid s$	zirkulärer Linksshift um s Stellen

Während die Ganzzahl-Addition bei MD4 und MD5 in *little endian* Architektur (d.h. ein aus 4 Bytes $a_3 a_2 a_1 a_0$, $0 \leq a_i \leq 255$ zusammengesetztes Wort repräsentiert die Zahl $a_0 2^{24} + a_1 2^{16} + a_2 2^8 + a_3$) ausgeführt wird, verwendet `SHA-1` eine *big endian* Architektur (d.h. $a_3 a_2 a_1 a_0$, $0 \leq a_i \leq 255$ repräsentiert die Zahl $a_3 2^{24} + a_2 2^{16} + a_1 2^8 + a_0$). 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
$j = 0, \ldots, 15$	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$
$j = 16, \ldots, 31$	$0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15$
$j = 32, \ldots, 47$	$0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15$

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

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

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

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

MD4(x)

input $x \in \{0,1\}^*, |x| = n$
$y := x10^k$**bin**$_{64}(n)$, $k \in \{0, 1, \ldots, 511\}$ **mit** $n + 1 + k + 64 \equiv 0 \pmod{512}$
$(H_1, H_2, H_3, H_4) := (67452301, efcdab89, 98badcfe, 10325476)$
sei $y = M_1 \cdots M_r$, $r = (n + 1 + k + 64)/512$
for $i := 1$ **to** r **do**
  sei $M_i = X[0] \cdots X[15]$
  $(A, B, C, D) := (H_1, H_2, H_3, H_4)$
  for $j := 0$ **to** 47 **do**
    $(A, B, C, D) := (D, (A + f_j(B, C, D) + X[z_j] + y_j) \hookleftarrow s_j, B, C)$
  $(H_1, H_2, H_3, H_4) := (H_1 + A, H_2 + B, H_3 + C, H_4 + D)$
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 ($j = 48, \ldots, 63$), 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 $f_j(X, Y, Z) := (X \wedge Z) \vee (Y \wedge \neg Z)$, $j = 16 \ldots 31$, ersetzt. Die y-Konstanten sind definiert als $y_j :=$ die ersten 32 Bit der Binärdarstellung von abs(sin($j + 1$)), $0 \leq j \leq 63$, und für z_j und s_j werden folgende Konstanten benutzt.

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

	s_j
$j = 0, \ldots, 15$	$7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22$
$j = 16, \ldots, 31$	$5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20$
$j = 32, \ldots, 47$	$4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23$
$j = 48, \ldots, 63$	$6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21$

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 $y := x10^k$ **bin**$_{64}(n)$, $k \in \{0, 1, \ldots, 511\}$ mit $n + 1 + k + 64 \equiv 0 \pmod{512}$
3 $(H_1, H_2, H_3, H_4) := (67452301, efcdab89, 98badcfe, 10325476)$
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 $(A, B, C, D) := (D, B + (A + f_j(B, C, D) + X[z_j] + y_j) \hookleftarrow s_j, B, C)$
10 $(H_1, H_2, H_3, H_4) := (H_1 + A, H_2 + B, H_3 + C, H_4 + D)$
11 **output** $H_1H_2H_3H_4$

1.2.7 Die SHA-1-Hashfunktion

Der *Secure Hash Algorithm* (SHA-1) ist eine Weiterentwicklung des MD4 bzw. MD5 Algorithmus. Er gilt in den USA als Standard und ist Bestandteil des von der US-Behörde NIST (National Institute of Standards and Technology) im August 1991 veröffentlichten DSS (Digital Signature Standard). Die Bitlänge von SHA-1 beträgt $l = 160$ Bit. Bei einer Wortlänge von 32 Bit entspricht dies 5 Wörtern. SHA-1 unterscheidet sich nur geringfügig von der SHA-0 Hashfunktion, in der eine Schwachstelle dazu führt, dass nach Berechnung von ca. 2^{61} Hashwerten ein Kollisionspaar gefunden werden kann (obwohl bei einem Geburtstagsangriff auf Grund der Hashwertlänge von 160 Bit ca. 2^{80} Berechnungen erforderlich sein müssten). Diese potentielle Schwäche von SHA-0 wurde im SHA-1 dadurch entfernt, dass SHA-1 in Zeile 8 einen zirkulären Shift um eine Bitstelle ausführt. Der SHA-1-Algorithmus benutzt die folgenden Konstanten K_j, $j = 0, \ldots, 79$

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

und folgende Funktionen f_j, $j = 0, \ldots, 79$

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

SHA-1(x)

input $x \in \{0,1\}^*, |x| = n$
$y := x10^k \mathbf{bin}_{64}(n)$, $k \in \{0,1,\ldots,511\}$ **mit** $n + 1 + k + 64 \equiv 0 \pmod{512}$
$(H_0, H_1, H_2, H_3, H_4) := (67452301, efcdab89, 98badcfe, 10325476, c3d2e1f0)$
sei $y = M_1 \cdots M_r$, $r = (n + 1 + k + 64)/512$
for $i := 1$ **to** r **do**
  sei $M_i = X[0] \cdots X[15]$
  for $t := 16$ **to** 79 **do**
    $X[t] := (X[t-3] \oplus X[t-8] \oplus X[t-14] \oplus X[t-16]) \hookleftarrow 1$
  $(A, B, C, D, E) := (H_0, H_1, H_2, H_3, H_4)$
  for $j := 0$ **to** 79 **do**
    $temp := (A \hookleftarrow 5) + f_j(B, C, D) + E + X[j] + K_j$
    $(A, B, C, D, E) := (temp, A, B \hookleftarrow 30, C, D)$
  $(H_0, H_1, H_2, H_3, H_4) := (H_0 + A, H_1 + B, H_2 + C, H_3 + D, H_4 + E)$
output $H_0H_1H_2H_3H_4$

1.2.8 Die SHA-2-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, $j = 0, \ldots, 63$ (in Hexadezimaldarstellung).

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

Dies sind jeweils die ersten 32 Bit der binären Nachkommastellen der dritten Wurzeln der ersten 64 Primzahlen $2, \ldots, 311$. SHA-256 arbeitet wie folgt.

SHA-256(x)

```
1  input x ∈ {0,1}*, |x| = n
2  y := x10^k bin_64(n), k ∈ {0,1,...,511} mit n+1+k+64 ≡ 0 (mod 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 ··· M_r, r = (n+1+k+64)/512
6  for i := 1 to r do
7    sei M_i = X[0] ··· X[15]
8    for t := 16 to 63 do
9      s0 := (X[t-15] ↪ 7) ⊕ (X[t-15] ↪ 18) ⊕ (X[t-15] → 3)
10     s1 := (X[t-2] ↪ 17) ⊕ (X[t-2] ↪ 19) ⊕ (X[t-2] → 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 ↪ 2) ⊕ (A ↪ 13) ⊕ (A ↪ 22)
15       maj := (A ∧ B) ⊕ (A ∧ C) ⊕ (B ∧ C)
16       t2 := s0 + maj
17       s1 := (E ↪ 6) ⊕ (E ↪ 11) ⊕ (E ↪ 25)
18       ch := (E ∧ F) ⊕ (¬E ∧ 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_0H_1H_2H_3H_4H_5H_6H_7
```

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

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

1.2.10 Die Sponge-Konstruktion

Die Konstruktionsidee hinter dem SHA-3-Gewinner Keccak wird von den Autoren als *Sponge* bezeichnet. Sie ähnelt oberflächlich der in 1.2.3 vorgestellten Konstruktion, weist aber einige Unterschiede auf. So ist ein Sponge nicht nur zur Konstruktion einer Hashfunktion gedacht, basiert statt auf einer Kompressionsfunktion h auf einer Permutation (oder Transformation) $f : \{0,1\}^b \to \{0,1\}^b$ und besitzt einen inneren Zustand, der nicht ausgegeben wird. Die Anzahl der Bits, die benötigt wird, um diesen inneren Zustand zu speichern, wird als **Kapazität** c des Sponges bezeichnet und ist sein wichtigster Sicherheitsparameter. Dagegen beschreibt die **Bitrate** $r = b - c$ die Anzahl der Bits des äußeren Zustands, über den Eingabe und Ausgabe des Sponges erfolgt.

Neben dem Kern f der Konstruktion ist auch wieder ein Preprocessing-Schritt notwendig. Die Anforderungen für diesen definieren wir vorab.

Definition 10. *Eine Funktion* $y\colon \{0,1\}^* \to \bigcup_{c\geq 1}\{0,1\}^{cr}$ *heißt* ***sponge-konformes Padding*** *für die Bitrate* r*, falls gilt:*

- $\forall n\,\forall (x,x') \in \{0,1\}^n \times \{0,1\}^n\,\exists z : y(x) = xz \wedge y(x') = x'z$,
- $\forall k \geq 0\,\forall x \neq x' : y(x) \neq y(x')0^{kr}$.

Es ist leicht zu sehen, dass die Paddingfunktion $\texttt{pad10*1}_r$ sponge-konform für r ist, wobei

$$\texttt{pad10*1}_r(x) = x10^d1, \quad d = \min\left\{i \;\middle|\; |x| + 2 + i \equiv_r 0\right\}.$$

Tatsächlich ist $\texttt{pad10*1}_r$ sogar für jedes $r' \geq 1$ sponge-konform. Ohne die abschließende 1 wäre dies nicht der Fall.

Definition 11. *Seien* $r \geq 1$*,* y *ein sponge-konformes Padding für* r *und* $f : \{0,1\}^b \to \{0,1\}^b$*. Die Funktion* $\texttt{Sponge}_{f,y,r} : \mathbb{N} \times \{0,1\}^* \to \{0,1\}^*$ *ist wie folgt definiert:*
Für $x \in \{0,1\}^*$ *sei* $y_1 \ldots y_k := y(x)$ *mit* $|y_i| = r\,(1 \leq i \leq k)$*. Wir definieren die Zustände* $s_i, i \geq 0$*:*

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

Weiter bezeichne z_i *die ersten* r *Bits von* $s_k + i - 1$*,* $i \geq 1$*, es sei* $c = \lfloor \frac{l}{r} \rfloor$ *und* z'_{c+1} *bezeichne die ersten* $l - cr$ *Bits von* z_{c+1}*. Dann ist*

$$\texttt{Sponge}_{f,y,r}(l,x) = z_1 \ldots z_c z'_{c+1}.$$

Für die Analyse definieren wir

$$\texttt{Absorb}_{f,y,r}(x) = s_k \textit{ und } \texttt{Squeeze}_{f,r}(l,s_k) = z_1 \ldots z_c z'_{c+1}.$$

Den Aufwand, für festes l ein Kollisionspaar $x \neq x'$ mit $\texttt{Sponge}_{f,y,r}(l,x) = \texttt{Sponge}_{f,y,r}(l,x')$ zu finden, können wir nach oben durch den Aufwand abschätzen, ein Paar $x \neq x'$ zu finden, so dass $\texttt{Absorb}_{f,y,r}(x) = \texttt{Absorb}_{f,y,r}(x')$. Da in der Absorbtionsphase der äußere Zustand (d.h. die Folge der ersten r Bits) beliebig und somit auch identisch gesetzt werden kann, genügt es, ein *inneres Kollisionspaar* zu finden, d.h. solche $x \neq x'$ so dass $\texttt{Absorb}^{\text{i}}_{f,y,r}(x) = \texttt{Absorb}^{\text{i}}_{f,y,r}(x')$, wobei $\texttt{Absorb}^{\text{i}}_{f,y,r}(x)$ die Folge der letzten c Bits von $\texttt{Absorb}_{f,y,r}(x)$ bezeichnet.

Um eine solche innere Kollision zu finden, hilft es, sich die 2^c inneren Zustände als Knoten eines gerichteten Multigraphen G vorzustellen, wobei jeder Knoten 2^r ausgehende Kanten mit Label 0^r bis 1^r hat. Ziel ist es dann, zwei verschiedene Pfade von 0^m zu demselben Knoten v zu finden, wobei zwei Pfade auch dann verschieden sind, wenn sich die Kanten nur in den Labeln unterscheiden. Anders als beim ZOM für eine Hashfunktion lohnt es sich hier für den Angreifer, die Argumente adaptiv nach einer Strategie $\mathcal{S}$ zu wählen. Der Algorithmus in Abb. 1.9 fasst dieses Vorgehen zusammen. Der Einfachheit halber gibt er ein Kollisionspaar nach dem Padding aus; für $\texttt{pad10*1}_r$ und alle y, deren Padding nur von $|x| \mod r$ abhängt, lässt sich dieses aber leicht auf ein Paar vor dem Preprocessing erweitern.

Satz 12. *Für jede Strategie* $\mathcal{S}$ *gibt* InnerCollision$(f,r,q,\mathcal{S})$ *im ZOM mit Erfolgswahrscheinlichkeit höchstens*

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

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

```
1  c := b − r, wobei f : {0,1}^b → {0,1}^b
2  initialisiere den gerichteten Multigraphen G = (V, A) := ({0,1}^c, ∅)
3  for i := 1 to q do
4    wähle v ∈ V und x ∈ {0,1}^r nach Strategie S
5    x'v' := f(xv)
6    A := A ∪ {(v, v', x, x')}
7  if ∃ verschiedene Pfade (0^c, u_1, x_1, x'_1), ..., (u_{k−1}, u_k, x_k, x'_k) und
8                (0^c, v_1, y_1, y'_1), ..., (v_{l−1}, v_l, y_l, y'_l) mit u_k = v_l in G
9    return(x_1(x_2 ⊕ x'_1)...(x_k ⊕ x'_{k−1}), y_1(y_2 ⊕ y'_1)...(y_k ⊕ y'_{k−1}))
10 else
11   return(?)
```

Abbildung 1.9: Bestimmung eines inneren Kollisionspaares

ein Kollisionspaar (x, x') für $\mathtt{Absorb}^i_{f,\mathrm{id},r}(x)$ aus. Wählt $\mathcal{S}$ nur von 0^c erreichbare Knoten v und kein Paar (v, x) mehrmals, so ist die Erfolgswahrscheinlichkeit exakt ε.

Beweis. Sei E_i das Ereignis "G enthält nach i Durchläufen keine zwei verschiedenen Pfade von 0^c zu einem Knoten v". Da nur durch eine Kante zwischen zwei von 0^c aus erreichbaren Knoten ein zweiter Pfad von 0^c aus geschlossen werden kann und nach $i-1$ Durchläufen höchstens i von 2^c Knoten erreichbar sind, gilt (unabhängig von $\mathcal{S}$):

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

Wählt $\mathcal{S}$ nur erreichbare Knoten und keine (v, x) mehrfach, so sind unter Annahme von $E_1 \cap \ldots \cap E_{i-1}$ auch i Knoten erreichbar (sonst gäbe es bereits zwei Pfade von 0^c zu einem Knoten in G) und es gilt Gleichheit. Analog zum Beweis vom Satz 6 folgt der behauptete Wert ε, mit Gleichheit im Fall der Wahl erreichbarer Knoten durch $\mathcal{S}$. □

Auch hier lässt sich q in Abhängigkeit von ε mittels $1 - x \approx e^{-x}$ abschätzen und es folgt:

$$q \approx c_\varepsilon 2^{\frac{c}{2}}, \quad c_\varepsilon = \sqrt{2 \ln \frac{1}{1-\varepsilon}} \quad .$$

1.2.11 SHA-3

Der Standard SHA-3 definiert die oben beschriebene Sponge-Konstruktion, 7 verschiedene bijektive Funktionen $f_w, w = 2^i, i \in \{0, \ldots, 6\}$ als Kern des Sponges $\mathtt{Sponge}_{f_w, \mathtt{pad10^*1}_r, r}$, sowie verschiedene Kombinationen von Bitraten r und Ausgabelängen l (c ist durch $25w - r$ bestimmt).

Jede Funktion $f_w : \{0,1\}^{5\times5\times w} \to \{0,1\}^{5\times5\times w}$ bildet ein zweidimensionales Feld A aus w-Bit-Wörtern auf ein ebensolches Feld $f_w(A)$ ab. Dabei wird $(12 + \log_2 w)$-mal eine Rundenfunktion $f'_w : \{0,1\}^{5\times5\times w} \times \{0,1\}^w \to \{0,1\}^{5\times5\times w}$ aufgerufen, die A und eine Rundenkonstante RC_i auf A' abbildet.

Es gilt

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

wobei θ, ρ, π, χ und ι_{RC} Bijektionen von $\{0,1\}^{5\times5\times w}$ nach $\{0,1\}^{5\times5\times w}$ sind. Die Funktion θ besteht aus $\oplus$-Operationen und ist so gewählt, dass sich $\theta^{-1}(A)$ an möglichst vielen Bits ändert, falls eines in A geflippt wird. Danach permutieren die Funktionen ρ und π die Bits von A innerhalb und zwischen den Wörtern. Ähnlich einer S-Box im SPN ist χ eine nichtlineare Funktion (die einzige solche in der Definition von f'_w), die nur auf 5-Bit-Blöcken arbeitet (jedes Bit hängt sogar nur von 2 anderen ab). Schlussendlich setzt ι_{RC} das Wort $A_{0,0}$ auf $A_{0,0} \oplus RC$.
Für die Werte $l \in \{224, 256, 384, 512\}$ definiert der Standard FIPS 202:

$$\texttt{SHA3-}l(x) = \texttt{Sponge}_{f_{1600},\texttt{pad10*1}_r,r}(l, x01), \qquad \text{wobei } r = 1600 - 2l.$$

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

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 $h_k \colon X \to Y$ 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 **$(\mathbf{n}, \mathbf{m}, \mathbf{l})$-Hashfamilie**.

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 $(x_1, h_k(x_1)), \ldots, (x_n, h_k(x_n))$ 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 $x_1, \ldots, x_n$ gegen unbefugt vorgenommene Veränderungen zu schützen, legt man sie zusammen mit ihren Hashwerten $y_1 = h_k(x_1), \ldots, y_n = h_k(x_n)$ 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 Hashwerten zum Ziel. Das heißt, der Gegner versucht, Hashwerte $h_k(x)$ 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 $x_1, \ldots, x_r$ (die er nicht selbst wählen konnte) die zugehörigen MAC-Werte $h_k(x_1), \ldots, h_k(x_r)$ und versucht, ein Paar (x', y') mit $h_k(x') = y'$ und $x' \notin \{x_1, \ldots, x_r\}$ 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.

Wechseln die Anwender nach jeder Hashwertberechnung den Schlüssel, so genügt es, dass $\mathcal{H}$ einem Impersonationsangriff widersteht.

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) sei $p(x \mapsto y)$ die Wahrscheinlichkeit, dass ein zufällig gewählter Schlüssel den Text x auf den Hashwert y abbildet:

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

wobei $K(x, y) = \{k \in K \mid h_k(x) = y\}$ alle Schlüssel enthält, die x auf y abbilden. D.h. $p(x \mapsto y)$ ist die Wahrscheinlichkeit, dass Bob das (vom Gegner gewählte) Paar (x, y) als echt akzeptiert. Dann gilt $\alpha = \max\{\alpha(x) \mid x \in X\}$, wobei

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

die Wahrscheinlichkeit ist, mit der einem Gegner bei optimaler Strategie eine Impersonation mit dem Text x gelingt.

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 Hashwert 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 (Beweis siehe Übungen).

Lemma 17. *Sei $\mathcal{X}$ eine Zufallsvariable mit endlichem Wertebereich $W(\mathcal{X}) \subseteq \mathbb{R}^+$. Dann gilt $\log E(\mathcal{X}) \geq E(\log \mathcal{X})$.*

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 Hashwerte 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. Bezeichne $\alpha(x, y) = p(x \mapsto y)$ die Wahrscheinlichkeit, mit der dem Gegner eine Impersonation mit dem Paar (x, y) gelingt. Da $\alpha = \max_{x,y} \alpha(x, y)$ ist, folgt $E(\alpha(\mathcal{X}, \mathcal{Y})) = \sum_{x,y} p(x, y)\alpha(x, y) \leq \alpha$ und 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 $H(\mathcal{Y} \,|\, \mathcal{X}) = H(\mathcal{K}) - H(\mathcal{K} \mid \mathcal{X}, \mathcal{Y})$ und somit $\log \alpha \geq H(\mathcal{K} \mid \mathcal{X}, \mathcal{Y}) - H(\mathcal{K})$. □

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 bedingten Verteilungen $\mathcal{K}_{x,y}$ für alle Paare $(x, y) \in X \times Y$.

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

Hierbei gilt $p(x, y) = p(x)p(y|x) = p(x)p(x \mapsto y)$. Zusammen ergibt sich

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

Erfolgswahrscheinlichkeit für Substitution

Bezeichne β die Wahrscheinlichkeit, mit der ein Gegner bei optimaler Strategie eine von Alice gesendete Nachricht durch eine andere Nachricht ersetzen kann, ohne dass Bob dies bemerkt.

Betrachten wir den Fall, dass der Gegner ein von Alice gesendetes Paar (x, y) durch (x', y') ersetzt. Dann 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 gleich

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

Da der Gegner keinen Einfluss auf die Wahl von (x, y) hat, ist β gleich dem erwarteten Wert von $\beta(x, y)$ unter der Verteilung

$$p(x, y) = p(x)p(y \,|\, x) = p(x)p(x \mapsto y).$$

unter der die Paare gesendet werden. 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*

$$h_k(x) = ax + b \bmod 3.$$

*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 Hashwertpaar genau einmal vorkommt. Folglich ist $p(x' \mapsto y' \,|\, x \mapsto y) = 1/3$ und somit $\beta = 1/3$. ◁

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 Hashwert 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 $\beta = 1/m$. □

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. □

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 Authentikationsmatrix jedes Hashwertpaar 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:*

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

Bemerkung 27. *Bei der Konstruktion von 2-universalen Hashfamilien 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 eine 2-universale (n, m, l)-Hashfamilie 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*

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

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

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

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

Die Hashfunktionen des vorigen Satzes erfüllen wegen $n = m = p$ nicht die Kompressionseigenschaft. Zwar lässt sich n noch geringfügig von p auf $p + 1$ vergrößern, ohne K und Y (und damit λ) 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 $l = m^2 \geq (n - 1)^2$.

Beweis. O.B.d.A. sei $\|K\| = \{1, \ldots, l\}$ und $Y = \{1, \ldots, m\}$. Es ist leicht zu sehen, dass eine (bijektive) Umbenennung $\pi \colon Y \to Y$ der Hashwerte 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 $i = 2, \ldots, m^2$ 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 $i = 2, \ldots, m^2$ 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. □

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. □

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

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

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 $k = (k_1, \ldots, k_d)$ mit dieser Eigenschaft existieren. □

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 von $\frac{p^d-1}{p(p-1)}$ bei einem λ-Wert von $\lambda = p^{d-2}$. Wie der nächste Satz zeigt, lässt sich dies nicht mit einem kleineren λ-Wert erreichen.*

Im Beweis des nächsten Satzes benötigen wir folgendes Lemma.

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

Beweis. Siehe Übungen. □

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

$$\lambda \geq \frac{n(m-1)+1}{m^2}$$

und somit $l \geq n(m-1)+1$.

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

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

Sei z_i die Anzahl von Indexpaaren (j, j') mit $j \neq j'$ und $A[i, j] = A[i, j'] = 1$ in Zeile i. Dann gibt es in den Zeilen $i = 2, \ldots, l$ insgesamt

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

solche Paare. 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 Hashwert-Paar $(1, 1)$ in genau λ Zeilen vorkommt (genauer: einmal in Zeile 1 und $(\lambda - 1)$-mal in den Zeilen $i = 2, \ldots, l$), und da $n(n-1)$ solche Spaltenpaare existieren, ergibt sich andererseits die Gleichung

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

Somit erhalten wir

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

□

1.3.3 MACs auf der Basis einer Kompressionsfunktion

Sei $h\colon \{0,1\}^{m+t} \to \{0,1\}^m$ die Kompressionsfunktion einer schlüssellosen Hashfunktion $\hat{h}$ (etwa MD5). Dann können wir mithilfe von h einen MAC konstruieren, indem wir als Initialisierungsvektor IV den symmetrischen Schlüssel $k \in K$ benutzen. Wir betrachten zunächst den Fall, dass auf das Preprocessing verzichtet wird.

Sei $\mathcal{H} = (X, Y, K)$ die Hashfamilie mit $X = \cup_{n\geq 1}\{0,1\}^{n\cdot t}$, $Y = \{0,1\}^m = K$ und $H = \{h_k \mid k \in K\}$, wobei $h_k(x)$ wie folgt berechnet wird:

---

1 **Sei** $x = x_1, \ldots, x_n, |x_i| = t$ **für** $i = 1, \ldots, n$
2 $z_0 := k$
3 **for** $i := 1$ **to** n **do**
4 $\quad z_i := h(z_{i-1}x_i)$
5 **output** z_n

Bei diesem MAC führt beispielsweise folgender Substitutionsangriff zum Erfolg.

Sei (x, z) ein Paar mit $h_k(x) = z$, wobei k der dem Gegner unbekannte Schlüssel ist. Dann lässt sich für einen beliebigen String $u \in \{0,1\}^t$ leicht der MAC-Wert des Textes $x' = xu$ mittels $h_k(x') = h(zu)$ berechnen.

Ein ähnlicher Angriff ist auch bei Verwendung einer Preprocessing-Funktion möglich. Hat diese beispielsweise die Form $y(x) = x\text{pad}(x)$, so lässt sich obiger Angriff entsprechend modifizieren (siehe Übungen).

1.3.4 CBC-MACs

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

Sei (M, C, K, E, D) ein 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.

Berechnung von $h_k(x)$:

---

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

Die Hashwertlänge 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 $x = x_1 \cdots x_n$ und $x' = (x_{n+1} \oplus IV \oplus z)x_{n+2} \cdots x_{n+m}$, wobei $|x_i| = t$ für $i = 1, \ldots, n+m$ ist, so muss auch der Text $x'' = x_1 \cdots x_{n+m}$ den MAC-Wert $h_k(x'') = z'$ haben.

Diesen Angriff kann man zwar ausschließen, indem man eine feste Länge für die Texte x vorschreibt. Dies schränkt jedoch die Anwendbarkeit des CBC-MACS erheblich ein. Zudem ist dann immer noch folgender Geburtstagsangriff auf den CBC-MAC möglich.

Geburtstagsangriff auf einen CBC-MAC

Dieser Angriff ermöglicht es, mit $q+1$ Hashwertfragen (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 $x^i = x_1^i x_2^i x_3 \cdots x_n$, $i = 1, \dots, q$.
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.
Befindet sich unter den erfragten Texten ein Kollisionspaar (x^i, x^j) mit $z^i = z^j$, so erfragt der Gegner für einen beliebigen Bitblock $u \in \{0,1\}^t - \{0^t\}$ den MAC-Wert $\bar{z}_i = h_k(\bar{x}^i)$ für den Text $\bar{x}^i = x_1^i(x_2^i \oplus u)x_3 \cdots x_n$, welcher zugleich MAC-Wert des Textes $\bar{x}^j = x_1^j(x_2^j \oplus u)x_3 \cdots x_n$ ist, 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 eine Hashfamilie $\mathcal{H}$ 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 Hashwerten 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 Hashwert für einen Text seiner Wahl generieren) und existientiellen Fälschungen (d.h. der Gegner kann den Hashwert 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 Hashwertfrage adaptiv und der Text x kann abgesehen vom ersten t-Bitblock beliebig vorgegeben werden.* ◁

1.3.5 Kombination einer Hashfunktion mit einem MAC (HMAC)

Falls der Textraum einer Hashfamilie den Hashwertraum einer anderen Hashfamilie enthält, lassen sich diese leicht komponieren (Nested-MAC).

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 $G = \{g_k \mid k \in K_2\}$ Hashfamilien. 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 $\mathcal{H}_2$ eine 2-universale Hashfamilie und für $\mathcal{H}_1$ eine schlüssellose Hashfunktion (etwa* `SHA-1`*), so erhält man einen so genannten HMAC (Hash-MAC).* ◁

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

$$ipad = \underbrace{36\ldots36}_{64mal} \text{ und } opad = \underbrace{5C\ldots5C}_{64mal}$$

512 Bit Konstanten. Dann berechnet sich HMAC wie folgt:

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

Hierbei fungiert die Funktion $f_k(x) = \texttt{SHA-1}((k \oplus ipad)x)$ als Hashfunktion mit Schlüssel, die beliebig lange Texte hasht, und der MAC $g_k(y) = \texttt{SHA-1}((k \oplus opad)y)$ 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 eine Hashfamilie $\mathcal{H} = (X, Y, K, H)$ ist ein probabilistischer Algorithmus $\mathcal{A}$, der q Fragen $x_1, \ldots, x_n$ 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 unbekannte (und zufällig gewählte) Schlüssel ist.*

Da der Gegner den Schlüssel k nicht kennt, ist ein Kollisionsangriff gegen eine Hashfamilie $\mathcal{H}$ schwieriger zu realisieren als ein Kollisionsangriff gegen eine schlüssellose Hashfunktion.

Satz 40. *Seien $\mathcal{H}_1 = (X, Y, K_1, F)$, $\mathcal{H}_2 = (X, Y, K_2, G)$ und $\mathcal{H} = (X, Z, K, H) = \mathcal{H}_1 \circ \mathcal{H}_2$ Hashfamilien. 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 gilt für jeden adaptiven (ε, q)-Fälscher für $\mathcal{H}$, dass $\varepsilon \leq \varepsilon_1 + \varepsilon_2$ ist.*

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

$$\Pr[x \notin \{x_1, \ldots, x_q\} \wedge g_{k_2}(f_{k_1}(x)) = z] \geq \varepsilon.$$

Hierbei wird (k_1, k_2) zufällig aus $K = K_1 \times K_2$ gewählt. Wir müssen zeigen, dass $\varepsilon \leq \varepsilon_1 + \varepsilon_2$ ist.

Behauptung 41. $\Pr[f_{k_1}(x) \in \{f_{k_1}(x_1), \ldots, f_{k_1}(x_q)\}] < \varepsilon_1$.

Hierzu betrachten wir folgenden adaptiven Kollisionsangreifer A' gegen $\mathcal{H}_1$: A' wählt zufällig einen Schlüssel $k_2 \in K_2$ und simuliert A, wobei A' für jede Anfrage x_i von A das Orakel f_{k_1} (mit unbekanntem, aber zufällig gewähltem Schlüssel k_1) nach dem Wert $y_i = f_{k_1}(x_i)$ fragt und an A die Antwort $z_i = g_{k_2}(y_i)$ zurückgibt. Sobald A ein Paar (x, z) ausgibt, fragt A' das Orakel f_{k_1} nach dem Hashwert $y = f_{k_1}(x)$ und gibt im Fall $y \in \{y_1, \ldots, y_q\}$ das Paar (x, x_i) für einen beliebigen Index i mit $y = y_i$ aus.
Da A' genau im Fall $y \in \{y_1, \ldots, y_q\}$ Erfolg hat, tritt dieser Fall mit Wahrscheinlichkeit kleiner ε_1 ein, womit Behauptung 41 bewiesen ist.

Behauptung 42. $\Pr[f_{k_1}(x) \notin \{f_{k_1}(x_1), \ldots, f_{k_1}(x_q)\} \wedge g_{k_2}(f_{k_1}(x)) = z] > \varepsilon - \varepsilon_1$.

Dies folgt direkt aus $\Pr[x \notin \{x_1, \ldots, x_q\} \wedge g_{k_2}(f_{k_1}(x)) = z] \geq \varepsilon$ und Behauptung 41.

Behauptung 43. $\Pr[f_{k_1}(x) \notin \{f_{k_1}(x_1), \ldots, f_{k_1}(x_q)\} \wedge g_{k_2}(f_{k_1}(x)) = z] < \varepsilon_2$.

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. Dann hat A'' genau im Fall $f_{k_1}(x) \notin \{f_{k_1}(x_1), \ldots, f_{k_1}(x_q)\} \wedge g_{k_2}(f_{k_1}(x)) = z$ Erfolg. Da es nach Voraussetzung keinen adaptiven (ε_2, q)-Fälscher gegen $\mathcal{H}_2$ gibt, muss $\varepsilon - \varepsilon_1 < \varepsilon_2$ sein. □

2 Elliptische Kurven

2.1 Elliptische Kurven über den reellen Zahlen

Definition 44. *Seien $a, b \in \mathbb{R}$. Eine elliptische Kurve E 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 45. *Betrachte die durch $y^2 = x^3 - 4x$ definierte elliptische Kurve E. Punkte: $(-2, 0)$, $(0, 0)$, $(2, 0)$, $(-1, 2)$, $(-1, -2)$.*

Auf den nicht-singulären Punkten von E lässt sich eine additive Gruppenoperation $+$ definieren. Die Idee dabei ist, dass die Summe aller auf einer Geraden g liegenden Punkte von E gleich dem neutralen Element $\mathcal{O}$ sein soll. Hierbei werden Tangentialpunkte doppelt und Wendepunkte dreifach gezählt und nur solche Geraden g berücksichtigt, auf denen bei dieser Zählweise 3 Punkte von E liegen, wobei im Fall, dass g parallel zur y-Achse verläuft, zusätzlich noch der Punkt $\mathcal{O}$ hinzugerechnet wird.

Am einfachsten ist der Fall, dass die Gerade g parallel zur y-Achse verläuft, also g den Punkt $\mathcal{O}$ enthält. Besteht die Schnittmenge S von g und $E \setminus \{\mathcal{O}\}$ aus 2 Punkten $P = \{x_1, y_1\}$ und $Q = \{x_2, y_2\}$, so gilt offensichtlich $x_1 = x_2$ und $y_1 = -y_2$ und wir erhalten $P + Q + \mathcal{O} = \mathcal{O}$ bzw. $-P = (x_1, -y_1)$. Diese Gleichung gilt auch für den Fall, dass S nur aus einem Punkt $P = \{x_1, y_1\}$ besteht, da P dann wegen $y_1 = 0$ ein Tangentialpunkt ist und daher doppelt gezählt wird.

Es bleibt der Fall, dass g nicht parallel zur y-Achse verläuft. Hier gibt es 2 Unterfälle:

$P \neq Q$: 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

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

ist, wobei $R = (x_3, y_3)$ folgende Koordinaten hat:

$$x_3 = \lambda^2 - x_1 - x_2 \text{ und } y_3 = \lambda(x_3 - x_1) + y_1.$$

Für alle $(x, y) \in E \cap g$ gilt

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

p lässt sich in $\mathcal{C}$ 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

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

wobei $x_3 = \lambda^2 - 2x_1$ und $y_3 = \lambda(x_3 - x_1) + y_1$ ist. 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 $F(x, y)$ 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 46. *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 47. *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*

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

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* $-P = \overline{P} = (x, -y)$.
- *Für* $P, Q \in E \setminus \{\mathcal{O}\}$ *ist*

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

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

Beispiel 49. $p = 11$, *E definiert durch* $y^2 = x^3 + x + 6$. *Zur Erinnerung: Im Fall* $p \equiv_4 3$ *lassen sich für* $z \in QR_p$ *die Wurzeln* y *durch* $\pm z^{\frac{p+1}{4}}$ *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
$y = \pm\sqrt{z} \bmod 11$	–	–	4; 7	5; 6	–	2; 9	–	2; 9	3; 8	–	2; 9

Da die Gruppe $(E, \mathcal{O}, +)$ $\|E\| = 13$ *Elemente enthält, und* 13 *eine Primzahl ist, haben alle Elemente entweder die Ordnung* 1 *oder* 13. *Da nur das neutrale Element* $\mathcal{O}$ *die Ordnung* 1 *hat, haben alle anderen Elemente* $P \in E - \{\mathcal{O}\}$ *die Ordnung* 13, *sind also 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 $2g = (2, 7) + (2, 7)$*:*

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

$\Rightarrow 2g = (5, 2)$
Berechnung von $3g = 2g + g = (5, 2) + (2, 7)$*:*

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

$\Rightarrow 3g = (8, 3)$

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

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

Bemerkung 51. *Es gibt einen effizienten Algorithmus (von Schoof) mit Zeitkomplexität $O(\log^8 q)$, der $\|E\|$ bei Eingabe von a, b und q berechnet.*

Satz 52. *Sei E eine elliptische Kurve über $\mathbb{F}_q$. Dann ist $(E, \mathcal{O}, +)$ isomorph zu $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$, wobei $n_1, n_2 \in \mathbb{N}^+$ sind und n_1 Teiler von n_2 und von $q - 1$ ist (ohne Beweis).*

Bemerkung 53. *Wie jede Gruppe muss E im Fall $\|E\|$ prim zyklisch sein. Wegen $\|E\| = n_1 \cdot n_2$ und da n_1 Teiler von n_2 ist, muss E auch dann zyklisch sein, wenn $\|E\|$ das Produkt von zwei verschiedenen Primzahlen (da dann $n_1 = 1$ sein muss).*

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 $\pm\sqrt{z} \bmod p$ von $z \in QR_p = \{x^2 \bmod p \mid x \in \mathbb{Z}_p^*\}$ (QR steht für quadratischer Rest) effizient mittels $\pm\sqrt{z} = \pm z^{(p+1)/4} \bmod p$ berechnen.

Folgende Funktion liefert dann eine kompakte Darstellung.

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

Für die Rekonstruktion können wir folgende Prozedur benutzen. Sei E eine elliptische Kurve $y^2 = x^3 + ax + b$ über $\mathbb{F}_q$ und sei $p(x) = x^3 + ax + b$.

Prozedur `PointDeCompress`(x, i)

```
1  z := p(x) mod p
2  if z ∈ QR_p then
3     y := √z mod p
4     if y ≢_2 i 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 $a^e \bmod m$ durch 'wiederholtes Quadieren 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 54. $(c_{l-1}, \ldots, c_0) \in \{-1, 0, 1\}^l$ *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 *mindestens eine* 0*, so heißt* $(c_{l-1}, \ldots, c_0)$ ***NAF-Darstellung*** *von* c*.*

Beispiel 55. *Sowohl* $(0, 1, 0, 1, 1)$ *als auch* $(1, 0, -1, 0, -1)$ *sind SBR-Darstellungen von* $c = 1 + 2 + 8 = 11 = -1 - 4 + 16$*.* ◁

Satz 56. *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 $(0, 1, \ldots, 1)$ von rechts beginnend durch den Teilstring $(1, 0, \ldots, 0, -1)$.
Zur effizienten Berechnung von $Q = cP$ benutzen wir das Horner-Schema

$$c = \sum_{i=0}^{l-1} c_i 2^i = (\ldots(c_{l-1}2 + c_{l-2})2 + \cdots + c_1)2 + c_0,$$

welches auf folgendes iteratives Schema zur Berechnung der Vielfachen $Q_i = \sum_{j=i}^{l-1} c_j 2^j P$ führt:

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

Dies führt auf folgende Algorithmen zur Berechnung von Vielfachen von Punkten auf E:

Prozedur `DoubleAdd`$(P, c_{l-1}, \ldots, c_0)$

```
1  Q := O
2  for i := l − 1 to 0 do
3     Q := 2 · Q
4     if c_i = 1 then Q := Q + P
5  output (Q)
```

Prozedur `DoubleAddSub`$(P, c_{l-1}, \ldots, c_0)$

```
1  Q := O
2  for i := l − 1 to 0 do
3     Q := 2 · Q
4     if c_i = 1 then Q := Q + P
5     if c_i = −1 then Q := Q + (−P)
6  output (Q)
```

Da eine l-Bitzahl im Durchschnitt $\frac{l}{2}$-Nullen in Binärdarstellung und $\frac{2l}{3}$-Nullen in NAF-Darstellung enthält, ist DoubleAddSub mit ca. $l + l/3$ Additionen/Subtraktionen um 11 Prozent effizienter als DoubleAdd mit ca. $l + l/2$ Additionen (siehe Übungen).

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 Unterschrift wird nicht physikalisch, sondern logisch (inhaltlich) an ein elektronisches Dokument gebunden und die Fähigkeit, einen individuellen Schriftzug auszuführen, wird durch geheimes Wissen ersetzt.

Definition 57. *Ein digitales Signaturverfahren besteht aus:*

- *einer Menge X von Dokumenten,*
- *einer endlichen Menge Y von Unterschriften,*
- *einer endlichen Menge K von Schlüsseln,*
- *einer Menge $S \subseteq K \times K$ von Schlüsselpaaren $(\hat{k}, k)$,*
- *einem Signaturalgorithmus $sig : K \times X \to Y$ und*
- *einem Verifikationsalgorithmus $ver : K \times X \times Y \to \{0, 1\}$*

mit

$$ver(k, x, y) = \begin{cases} 1, & sig(\hat{k}, x) = y, \\ 0, & sonst \end{cases}$$

für alle $(\hat{k}, k) \in S$.

Klassifikation von Angriffen gegen Signaturverfahren

Angriff bei bekanntem Verifikationsschlüssel (key-only attack)

Angriff bei bekannter Signatur (known signature attack): für eine Reihe von Dokumenten x ist die zugehörige Signatur $y = sig(\hat{k}, x)$ bekannt, auf deren Auswahl der Gegner keinen Einfluss hat.

Angriff bei frei wählbaren Dokumenten (chosen document attack): d.h. der Gegner war für eine gewisse Zeit in der Lage, für von ihm gewählte Dokumente die zugehörige Signatur in Erfahrung zu bringen und versucht nun, für ein "neues" Dokument die Unterschrift zu bestimmen.

adaptiver Angriff bei frei wählbaren Dokumenten: d.h. der Gegner wählt jeweils das nächste Dokument in Abhängigkeit von der Signatur des vorigen.

Erfolgskriterien für die Fälschung digitaler Signaturen

uneingeschränktes Fälschungsvermögen (total break): Der Gegner hat einen Weg gefunden, die Funktion $x \mapsto sig(\hat{k}, x)$, effizient zu berechnen ohne $\hat{k}$ als Eingabe zu benutzen (k ist ohnehin bekannt).

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

nichtselektives (existentielles) Fälschungsvermögen: Der Gegner kann für irgendein Dokument x die zugehörige digitale Signatur bestimmen.

Beim **RSA-Signaturverfahren** ist $K = \{(a, n) | n = pq$ für Primzahlen p, q 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

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

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

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

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

Wir betrachten unterschiedliche Angriffsmöglichkeiten gegen das RSA-Signaturverfahren.

- Es ist nicht schwer, eine nichtselektive Fälschung bei bekanntem Verifikationsschlüssel durchzuführen. Hierzu wählt der Gegner zu einer beliebigen Signatur $y \in Y$ das Dokument $x = y^e \bmod n$.
- Zudem ist eine existentielle Fälschung bei bekannten Signaturen möglich, falls der Gegner zwei signierte Dokumente $(x_1, y_1), (x_2, y_2)$ 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 Dokumenten sogar eine selektive Fälschung durchführen. Ist bereits die Signatur für ein beliebiges Dokument $x' \in \mathbb{Z}^*_n$ bekannt und kann sich der Gegner die Signatur für das Dokument $x'' = x \cdot x'^{-1} \bmod n$ beschaffen, so kann er daraus wie oben eine gültige Signatur für das Dokument x berechnen.

Diese Angriffe kann man vereiteln, indem man das Dokument x mit Redundanz versieht (indem man z.B. anstelle von x den Text xx signiert). Um auch längere Dokumente effizient signieren zu können, wird i.a. jedoch eine geeignete Hashfunktion h benutzt und nicht das gesamte Dokument 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 $y \in Y$ ein passendes Dokument 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 Dokumente x die zugehörige Signatur y beschaffen kann, so sollte h sogar kollisionsresistent sein. Andernfalls könnte der Gegner ein Kollisonspaar (x, x') für h finden und sich das (unverdächtige) Dokument x signieren lassen. Die erhaltene Signatur y für x' verwenden.

3.1 Das ElGamal-Signaturverfahren

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

Wir beschreiben nun das Signaturverfahren von ElGamal. Sei p eine große Primzahl und α ein Erzeuger von $\mathbb{Z}_p^*$ (p und α sind öffentlich). Jeder Teilnehmer B erhält als geheimen Signierschlüssel eine Zahl $a \in \mathbb{Z}_{p-1} = \{0, \ldots, p-2\}$ und gibt $\beta = \alpha^a \bmod p$ als öffentlichen Verifikationsschlüssel bekannt:

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

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

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

Lemma 59. *Die Bedingung $\beta^\gamma \gamma^\delta \equiv_p \alpha^x$ ist genau dann erfüllt, wenn es ein $z \in \mathbb{Z}_{p-1}^*$ mit $sig(\hat{k}, x, z) = (\gamma, \delta)$ gibt.*

Beweis. Wegen $\gamma \equiv \alpha^z \bmod p$ 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. □

Zur Sicherheit des ElGamal-Systems

1. Falls der Gegner den diskreten Logarithmus 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 gegebenem Klartext x.
   a) Der Gegner wählt zuerst γ und versucht, ein passendes δ zu finden. Mit $\alpha^x \equiv \beta^\gamma \gamma^\delta \bmod p$ folgt $\delta = \log_\gamma \alpha^x \beta^{-\gamma}$. D.h. die Bestimmung von δ ist eine Instanz des dikreten Logarithmus Problems (kurz: DLP).
   b) Der Gegner wählt zuerst δ und versucht dann γ aus $\alpha^x \equiv \beta^\gamma \gamma^\delta \bmod p$ zu bestimmen. Dazu ist kein effizientes Verfahren bekannt.
   c) Der Gegner wählt γ und δ gleichzeitig. Auch hierfür ist kein effizientes Verfahren bekannt.

3. Versucht der Gegner bei einem nichtselektiven Angriff, zuerst γ und δ zu wählen und dazu ein passendes Dokument 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. Wähle beliebige Zahlen $u \in \mathbb{Z}_{p-1}$, $v \in \mathbb{Z}^*_{p-1}$ und berechne $\gamma = \alpha^u \beta^v \bmod p$. Dann ist (γ, δ) genau dann eine gültige Signatur für ein Dokument 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 für das Dokument $x = u\delta \bmod p-1$ mittels $\delta = -\gamma v^{-1} \bmod p-1$ erfüllen lässt. Bei Wahl von $v = 1$ führt z.B. jedes $u \in \mathbb{Z}_{p-2}$ mittels $\gamma = \alpha^u \beta \bmod p$ und $\delta = -\gamma \bmod p-1$ auf eine gültige Signatur (γ, δ) für das Dokument $x = u\delta \bmod p-1$.

Bemerkung 60. *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 sie wegen $\delta \equiv_{p-1} (x - a\gamma)z^{-1}$ im Fall $\mathrm{ggT}(\gamma, p-1) = 1$ die geheime Zahl $a = (-z\delta + x)\gamma^{-1} \bmod p-1$ berechnen. Ist $ggT(\gamma, p-1) = d > 1$, so lassen sich aus dieser Kongruenz d Kandidaten für a gewinnen.

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 $d = ggT(\delta_1 - \delta_2, p-1)$ Kandidaten für z gewinnen und daraus wie oben a berechnen, falls d nicht zu groß ist.

3.2 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 $ord_p(\alpha) = q$. Weiter sei $h : \{0,1\}^* \to \mathbb{Z}_q$ eine Hashfunktion, die jedem Dokument $x \in X = \{0,1\}^*$ einen Hashwert in $\mathbb{Z}_q$ zuordnet.

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

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

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

wobei $\gamma = h(x \text{ bin}(\alpha^z \bmod p))$ und $\delta = (z + a\gamma) \bmod q$ ist. Der Signaturraum ist also $Y := \mathbb{Z}_q \times \mathbb{Z}_q$.
Verifikation: $ver(k, \gamma, \delta) = 1$, falls $h(x bin(\alpha^\delta \beta^{-\gamma} \bmod p)) = \gamma$ ist.

3.3 Der Digital Signature Algorithm (DSA)

(Standard in USA seit 1994)
Hierbei wird das ElGamal-Verfahren wie folgt modifiziert:

1. δ als Lösung von $z\delta - a\gamma \equiv_{p-1} x$ (d.h. $\delta = (x + a\gamma)z^{-1}$) $\rightsquigarrow$ 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 $\delta^{-1} = (x + a\gamma)^{-1} z \bmod p - 1 \rightsquigarrow$ 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 $ord_p(\alpha) = q$. 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 wird eine 512-1024 Bit Primzahl p der Form $rq + 1$ benutzt, wobei q eine 160 Bit Primzahl und $\alpha \in \mathbb{Z}^*_p$ mit $ord_p(\alpha) = q$ ist. Weiter ist $X = \mathbb{Z}^*_p$ und $Y = \mathbb{Z}_q \times \mathbb{Z}^*_q$. Der Signierschlüssel hat die Form $\hat{k} = (p, q, \alpha, a)$, wobei $a \in \mathbb{Z}^*_q$ ist. Der zugehörige Verifikationsschlüssel ist $k = (p, q, \alpha, \beta)$ mit $\beta = \alpha^a \bmod p$.
Zu gegebenem $x \in X$ wird zufällig eine geheime Zahl $z \in \mathbb{Z}^*_p$ gewählt.

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

(falls $\delta \equiv_q 0$ muss ein neues z gewählt werden). Die Verifikationsbedingung ist

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

wobei $e = x\delta^{-1} \bmod q$ und $d = \gamma\delta^{-1} \bmod q$ ist.
Im Fall $sig(\hat{k}, z, x) = (\gamma, \delta)$ 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 61. $q = 101$, $p = 78q + 1 = 7879$, $g = 3$ *($ord_p(3) = p - 1$)*

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

Wir wählen $a = 75 \in \mathbb{Z}^*_q$, *d.h.* $\beta = \alpha^a \bmod p = 170^{75} \bmod p = 4547$ *Um das Dokument* $x = 1234 \in \mathbb{Z}^*_p$ *zu signieren, wählen wir die geheime Zufallszahl* $z = 50 \in \mathbb{Z}^*_p$ *($\rightsquigarrow z^{-1} = 99$)*

und erhalten dann

$$\begin{aligned} \gamma &= (170^{50} \bmod 7879) \bmod 101 \\ &= 2518 \bmod 101 \\ &= 94 \\ \delta &= (1234 + 75 \cdot 94) \cdot 99 \bmod 101 \\ &= 97 \ (\rightsquigarrow \delta^{-1} = 25) \end{aligned}$$

d.h. $sig(p, q, \alpha, z, x) = (94, 97)$, *wobei* $\hat{k} = (p, q, \alpha, a)$
Um diese Signatur zu prüfen berechnen wir:

$$\begin{aligned} e &= x\delta^{-1} \bmod q \\ &= 1234 \cdot 25 \bmod 101 \\ &= 45 \\ d &= \gamma\delta^{-1} \bmod q \\ &= 94 \cdot 25 \bmod 101 \\ &= 27 \end{aligned}$$

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

3.4 ECDSA (Elliptic Curve DSA)

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

Definition 62. *Sei E eine elliptische Kurve über einem endlichen Körper. 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 eine kryptografische Hashfunktion (z.B.* SHA-2*).*

$X = \{0,1\}^*$, $Y = \mathbb{Z}_q^* \times \mathbb{Z}_q^*$. öffentlicher Verifikationsschlüssel: (p, q, E, A, B), wobei $B = m \cdot A$ geheimer Signierschlüssel: (p, q, E, A, m), $m \in \mathbb{Z}_q^*$.

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

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

$$ver(k, x, \gamma, \delta) = \begin{cases} 1, & u \bmod q = \gamma \\ 0, & sonst \end{cases} \quad wobei$$

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

Korrektheit der Verifikation beim ECDSA:

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

Beispiel 63. *Signieren und Verifizieren: Sei E über $\mathbb{Z}_{11}$ definiert durch $\gamma^2 = x^3 + x + 6$ Wir wählen $A = (2,7)$, $m = 7 \to p = 11, q = 13, B = 7A = (7,2)$*

Annahme: Wollen Dokument x mit $h(x) = 4$ mit dem Signierschlüssel $\hat{k} = (p,q,E,A,m)$ und der Zufallszahl $r = 3$ signieren.

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

Verifikation von $(\gamma,\delta) = (8,7)$ unter $k = (p,q,E,A,B)$:

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

$\rightsquigarrow u \bmod q = 8 = \gamma$. ◁

3.5 One-time Signatur (Lamport)

Sei $f : U \to V$ eine injektive Einwegfunktion. Der Dokumentenraum ist $X = \{0,1\}^n$ und der Signaturraum ist $Y = U^n$.

Der Signierschlüssel ist eine beliebige Folge $\hat{k} = (u_{i,b})_{i=1,\ldots,n;b=0,1}$ von $2n$ paarweise verschiedenen Elementen aus U.

Der zugehörige Verifikationsschlüssel ist dann $k = (v_{i,b})_{i=1,\ldots,n;b=0,1}$ mit $v_{i,b} = f(u_{i,b})$ für alle $(i,b) \in \{1,\ldots,n\} \times \{0,1\}$.

Signaturerstellung: Die Signatur für ein Dokument $x = x_1 \ldots x_n \in X$ ist

$$sig(\hat{k}, x) = \underbrace{u_{1,x_1} \ldots u_{n,x_n}}_{y}.$$

Verifikation:

$$ver(k, x, \underbrace{u_1, \ldots, u_n}_{y}) := \begin{cases} 1, & f(u_i) = v_{i,x_i} \text{ für } i = 1,\ldots,n, \\ 0, & sonst. \end{cases}$$

Beispiel 64. *Wir wählen als Einwegfunktion eine Funktion der Form $f : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ 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 $n = 3$.

Dann erhalten wir für den Signierschlüssel $\hat{k} = (u_{1,0}, u_{1,1}, u_{2,0}, u_{2,1}, u_{3,0}, u_{3,1})$, wobei $u_{1,0} = 5831$, $u_{1,1} = 803$, $u_{2,0} = 4285$, $u_{2,1} = 735$, $u_{3,0} = 2467$, $u_{3,1} = 6449$ den zugehörigen

Verifikationsschlüssel $k = (v_{1,0}, v_{1,1}, v_{2,0}, v_{2,1}, v_{3,0}, v_{3,1})$*, wobei* $v_{1,0} = 2009$*,* $v_{1,1} = 4672$*,* $v_{2,0} = 268$*,* $v_{2,1} = 3810$*,* $v_{3,0} = 4721$ *und* $v_{3,1} = 5731$ *ist. Die Signatur für das Dokument* $x = 110$ *ist dann*

$$sig(\hat{k}, x) = (u_{1,1}, u_{2,1}, u_{3,0}) = (u_1, u_2, u_3) = (803, 735, 2467).$$

Die Verifikation ergibt den Wert $ver(k, x, u_1, u_2, u_3) = 1$*, da Folgendes gilt:*

$$i = 1 : f(u_1) = f(803) = 3^{803} \bmod 7879 = 4672 = v_{1,x_1}$$
$$i = 2 : f(u_2) = f(735) = 3^{735} \bmod 7879 = 3810 = v_{2,x_2}$$
$$i = 3 : f(u_3) = f(2467) = 3^{2467} \bmod 7879 = 4721 = v_{3,x_3}$$ ◁

Zum Nachweis der Sicherheit des Signaturverfahrens nehmen wir an, dass $f : U \to V$ eine Bijektion ist und dass ein deterministischer Algorithmus LAMPORT-FÄLSCHUNG(k) existiert, der bei Eingabe eines Verifikationsschlüssels k eine existentielle Fälschung (x, y) mit $ver(k, x, y) = 1$ berechnet. Betrachte folgenden probabilistischen Algorithmus:

Prozedur `Lamport-Urbild`(v)

```
1 wähle zufällig einen Verifikationsschlüssel k = (v_{i,b})_{i=1,...,n;b=0,1}
2 falls v nicht in k vorkommt, ersetze für ein zufällig gewähltes
     Indexpaar (j, a) den Wert v_{j,a} durch v
3 (x_1, ..., x_n, u_1, ..., u_n) =: Lamport-Fälschung(k)
4 if x_j = a then
5   output (u_j)
6 else
7   output ('?')
```

Satz 65. *Unter den genannten Voraussetzungen gibt* LAMPORT-URBILD(v) *für ein zufällig aus* V *gewähltes* v *mit Wahrscheinlichkeit* $\frac{1}{2}$ *ein Urbild* u *von* v *aus.*

Beweis. Im Fall $x_j = a$ gibt der Algorithmus LAMPORT-URBILD ein Urbild $u = u_j$ von v aus:

$$f(u_j) = v_{j,x_j} = v_{j,a} = v.$$

Daher reicht es zu zeigen:

$$p = Prob_{v \in_R V}[\text{LAMPORT-URBILD}(v) \neq \text{'?'}] = 1/2.$$

Sei $\mathcal{S}$ die Menge aller möglichen Verifikationsschlüssel k und für $v \in V$ sei $\mathcal{S}_v$ die Menge aller $k \in \mathcal{S}$, die v enthalten. $\mathcal{T}_v$ bezeichne die Menge aller $k \in \mathcal{S}_v$, für die LAMPORT-FÄLSCHUNG(k) ein Urbild von v liefert. Weiter sei $t_v = \|\mathcal{T}_v\|$, $s_v = \|\mathcal{S}_v\|$ und $s = \|\mathcal{S}\|$.

Da jeder der s Verifikationsschlüssel $k \in \mathcal{S}$ zu der Summe $\sum_{v \in V} t_v$ einen Wert von genau n beiträgt (für jedes $i = 1, \ldots, n$ ist $k = (v_{i,b})_{i=1,\ldots,n;b=0,1}$ in genau einer der beiden Mengen $T_{v_{i,0}}$ und $T_{v_{i,1}}$ enthalten), ist $\sum_{v \in V} t_v = ns$. Dagegen trägt jedes k zu der Summe $\sum_{v \in V} s_v$ den Wert $2n$ bei ($k = (v_{i,b})_{i=1,\ldots,n;b=0,1}$ ist genau in den $2n$ Mengen $S_{v_{i,b}}$ enthalten), weshalb $\sum_{v \in V} s_v = 2ns$ ist. Da aus Symmetriegründen die Zahlen s_v alle gleich sind, folgt $s_v = 2ns/\|V\|$.

Sei nun p_v die Erfolgswahrscheinlichkeit von LAMPORT-URBILD(v), d.h. $p_v = t_v/s_v$. Dann ergibt sich die durchschnittliche Erfolgswahrscheinlichkeit p zu

$$p = \frac{1}{\|V\|} \sum p_v = \frac{1}{\|V\|} \sum t_v/s_v = \frac{1}{2ns} \sum t_v = \frac{ns}{2ns} = \frac{1}{2}.$$

□

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.

3.6 Full Domain Hash (FDH) Signaturen

Sei $\mathcal{F} = \{f_k | k \in \mathcal{K}\}$ eine Familie von Falltür-Permutationen auf $\{0,1\}^n$, d.h. für jedes $k \in \mathcal{K}$ gilt:

- f_k ist Einweg-Permutation auf $\{0,1\}^n$.
- Es existiert ein $\hat{k} \in \mathcal{K}$ mit $f_k(f_{\hat{k}}(x)) = x$ für alle $x \in \{0,1\}^n$.

Weiter sei $G : \{0,1\}^* \to \{0,1\}^n$ eine Zufallsfunktion, d.h. die Zufallsvariablen $X_x = G(x)$ sind stochastisch unabhängig und es gilt

$$Prob[G(x) = y] = 2^{-n} \ \forall x \in \{0,1\}^* \text{ und } y \in \{0,1\}^n.$$

G modelliert eine Hashfunktion $H : \{0,1\}^* \to \{0,1\}^n$ mit optimalen kryptographischen Eigenschaften (vgl. Zufalls-Orakel-Modell, ZOM), deren Wertebereich den gesamten Definitionsbereich der Funktionen f_k ausfüllt (full domain hash). In der Praxis wird anstelle von G eine konkrete Hashfunktion (etwa `SHA-1`) eingesetzt, die meist nicht den gesamten Definitionsbereich der Funktionen f_k ausschöpft.

Die auf $\mathcal{F}$ und G basierende FDH-Signatur funktioniert wie folgt. Um für ein Dokument $x \in X = \{0,1\}^*$ eine Signatur $y \in Y = \{0,1\}^n$ zu berechnen, wird ein Signierschlüssel $\hat{k}$ benutzt:

$$sig(\hat{k}, x) := f_{\hat{k}}(G(x)).$$

Diese wird unter Verwendung des zugehörigen Verifikationsschlüssels k wie folgt überprüft:

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

Z.B. beruht das RSA-Signaturverfahren in Verbindung mit einer Hashfunktion auf diesem Prinzip. Ein Problem hierbei ist allerdings, dass der Wertebereich von in der Praxis verwendeten Hashfunktionen die Menge $\{0,1\}^{160}$ ist und für die RSA-Falltür-Permutation ein Definitionsbereich von $\{0,1\}^n$ mit $n \approx 1024$ zu wählen ist, um eine ausreichend große Sicherheit zu erreichen. 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 mindern kann.

Sicherheitsanalyse der FDH-Signatur im ZOM

Sei `FDH-Fälschung` ein probabilistischer Algorithmus, der bei Eingabe des öffentlichen Verifikationsschlüssels k mit Wahrscheinlichkeit ε eine existentielle Fälschung (x, y) mit $ver(x, y) = 1$ ausgibt und sei q die Anzahl der verschiedenen Orakelfragen $x_1, \ldots, x_q$ von `FDH-Fälschung` an G. Wir nehmen an, dass $\varepsilon > 2^{-n}$ ist, da für ein beliebiges Dokument $x \in \{0,1\}^*$ ein zufällig gewähltes $y \in \{0,1\}^n$ bereits mit Wahrscheinlichkeit 2^{-n} eine gültige Signatur liefert.

Betrachte folgenden Invertierungsalgorithmus für f_k.

Prozedur `FDH-Invert`(k, z_0)

```
1 wähle zufällig j ∈ {1,...,q}
2 simuliere FDH-Fälschung(k), wobei die i-te Orakelfrage x_i, 1 ≤ i ≤ q,
    im Fall i = j durch z_0 und sonst durch ein zufällig gewähltes
    z ∈ {0,1}^n beantwortet wird.
3 if FDH-Fälschung(k) = (x,y) ∧ f_k(y) = z_0 then output (y)
4 else output ('?')
```

Der nächste Satz zeigt, dass `FDH-Invert` bei Eingabe eines beliebigen Verifikationsschlüssels $k \in \mathcal{K}$ die Funktion f_k an einem zufällig gewählten Wert $z_0 \in \{0,1\}^n$ mit einer von ε und q abhängigen Erfolgswahrscheinlichkeit ε' invertiert.

Satz 66. *Falls* `FDH-Fälschung` *bei Eingabe k nach genau q Fragen an G eine gültige Fälschung (x, y) mit Wahrscheinlichkeit $\varepsilon > 2^{-n}$ ausgibt, findet* `FDH-Invert` *bei Eingabe von k und einem zufällig gewählten String $z_0 \in \{0,1\}^n$ mit Wahrscheinlichkeit*

$$\varepsilon' \geq \frac{\varepsilon - 2^{-n}}{q}$$

ein Urbild y von z_0 für die Funktion f_k.

Beweis. Da die Eingabe z_0 zufällig gewählt wird, erhält `FDH-Fälschung` als Antwort auf seine Orakelfragen $x_1, \ldots, x_q$ zufällig gewählte Strings z, was dem ZOM entspricht. Daher ist die Wahrscheinlichkeit, dass `FDH-Fälschung`(k) bei der Simulation Erfolg hat, also ein Paar (x, y) mit $G(x) = f_k(y)$ ausgibt, genau ε. Falls `FDH-Fälschung` das Paar (x, y) ausgibt, ohne den Wert $G(x)$ zu erfragen (d.h. $x \notin \{x_1, \ldots, x_q\}$), so nimmt $G(x)$ den Wert $f_k(y)$ nur mit Wahrscheinlichkeit 2^{-n} an, d.h.

$$\Pr[\texttt{FDH-Fälschung}(k) \text{ hat Erfolg} \,|\, x \notin \{x_1, \ldots, x_q\}] = 2^{-n},$$

was $\Pr[\texttt{FDH-Fälschung}(k) \text{ hat Erfolg } \wedge x \notin \{x_1, \ldots, x_q\}] \leq 2^{-n}$ impliziert. Wegen

$$\begin{aligned} \varepsilon &= \Pr[\texttt{FDH-Fälschung}(k) \text{ hat Erfolg} \wedge x \in \{x_1, \ldots, x_q\}] \\ &\quad + \Pr[\texttt{FDH-Fälschung}(k) \text{ hat Erfolg} \wedge x \notin \{x_1, \ldots, x_q\}] \\ &\leq \Pr[\texttt{FDH-Fälschung}(k) \text{ hat Erfolg} \wedge x \in \{x_1, \ldots, x_q\}] + 2^{-n}, \end{aligned}$$

erhalten wir

$$\Pr[\texttt{FDH-Fälschung} \text{ hat Erfolg} \wedge x \in \{x_1, \ldots, x_q\}] \geq \varepsilon - 2^{-n}.$$

Da die Frage $x_j \in \{x_1, \ldots, x_q\}$, die mit z_0 beantwortet wird, zufällig ausgewählt wird und `FDH-Fälschung` keinerlei Information über j erhält, folgt

$$\begin{aligned} \Pr[\texttt{FDH-Invert} \text{ hat Erfolg}] &\geq \Pr[\texttt{FDH-Fälschung} \text{ hat Erfolg} \wedge x = x_j] \\ &= \frac{1}{q} \sum_{i=1}^{q} \Pr[\texttt{FDH-Fälschung} \text{ hat Erfolg} \wedge x = x_i] \\ &= \Pr[\texttt{FDH-Fälschung} \text{ hat Erfolg} \wedge x \in \{x_1, \ldots, x_q\}]/q \\ &\geq (\varepsilon - 2^{-n})/q \end{aligned}$$

□

Falls sich also f_k nur mit einer sehr kleinen Wahrscheinlichkeit ε' effizient invertieren lässt, so gelingt einem ähnlich effizienten Gegner, der nicht mehr als q Hashwertberechnungen durchführt im ZOM höchstens mit Wahrscheinlichkeit $q\varepsilon' + 2^{-n}$ eine existentielle Fälschung für die FDH-Signatur. Ein ähnliches Resultat lässt sich auch für den Fall beweisen, dass der Gegner einen Angriff mit frei wählbaren Dokumenten ausführt.

3.7 Verbindliche Signaturen (undeniable signatures)

In manchen Fällen ist es für den Unterzeichner eines Dokumentes nicht wünschenswert, dass jeder die von ihm geleistete Unterschrift verifizieren kann.

Zum Beispiel könnte eine Softwarefirma ihre Produkte mit einer Signatur versehen, die u.a. Virusfreiheit garantiert.

Problem: Auch SW-Piraten, die ein Produkt unrechtmäßig erworben haben, können sich von der Gültigkeit der Signatur überzeugen.

Lösung: Die Signatur wird so erstellt, dass ihre Verifikation nur unter Mitwirkung der Softwarefirma möglich ist.

Neues Problem: Die Softwarefirma könnte sich absichtlich unkooperativ verhalten, um eine von ihr erzeugte echte Signatur als gefälscht abzuleugnen.

Lösung: Es gibt ein *Ableugnungsprotokoll (disavowal protocol)*, mit dem die Softwarefirma gefälschte Signaturen als solche entlarven kann. Verweigert die Softwarefirma auch hier ihre Mitwirkung, so liegt der Verdacht nahe, dass die vorliegende Signatur echt ist.