Vorlesungsskript

Kryptologie 2

Sommersemester 2010

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

27. Juni 2010

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. In der Tat nehmen kryptografische Hashverfahren 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 Bildung von digitalen Signaturen sehr nützlich. Auf weitere Anwendungsmöglichkeiten werden wir später eingehen.

Den überaus meisten 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 daher gewissermaßen darauf abzielen, „charakteristische Merkmale“ von x in den Hashwert $h(x)$ einfließen zu 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)

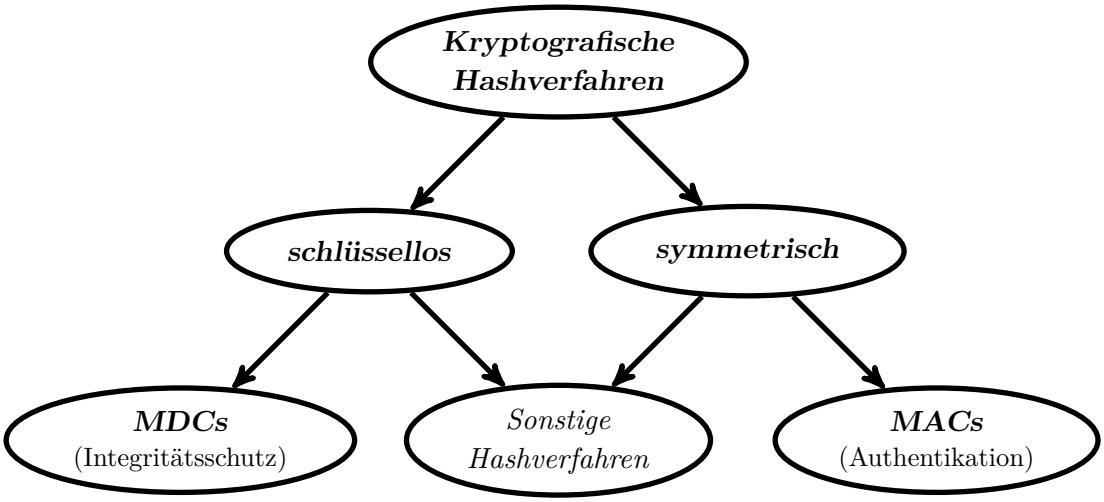

Abbildung 1.1: Eine grobe Einteilung von kryptografischen Hashverfahren.

  - 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 ungesicherten 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. Wie sich Nachrichten

mit einem MAC authentisieren lassen, ist in Abbildung 1.3 dargestellt. Man beachte, dass nun auch der Hashwert über den unsicheren Kanal gesendet wird.

> Möchte Bob eine Nachricht x an Alice übermitteln, so berechnet er den zugehörigen *MAC*-Hashwert $y = h_k(x)$ und fügt diesen der Nachricht x hinzu. Alice 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 Bob seine Nachricht x um den Hashwert $y = h_k(x)$ ergänzt, gibt er Alice 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.

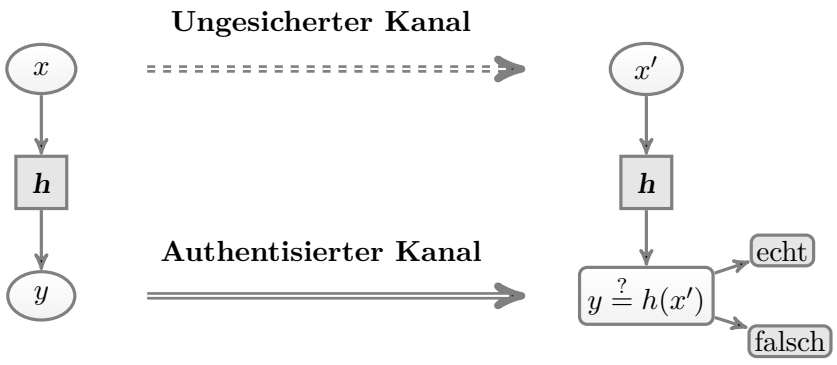

Abbildung 1.2: Einsatz eines MDC h zur Überprüfung der Integrität eines Datensatzes x.

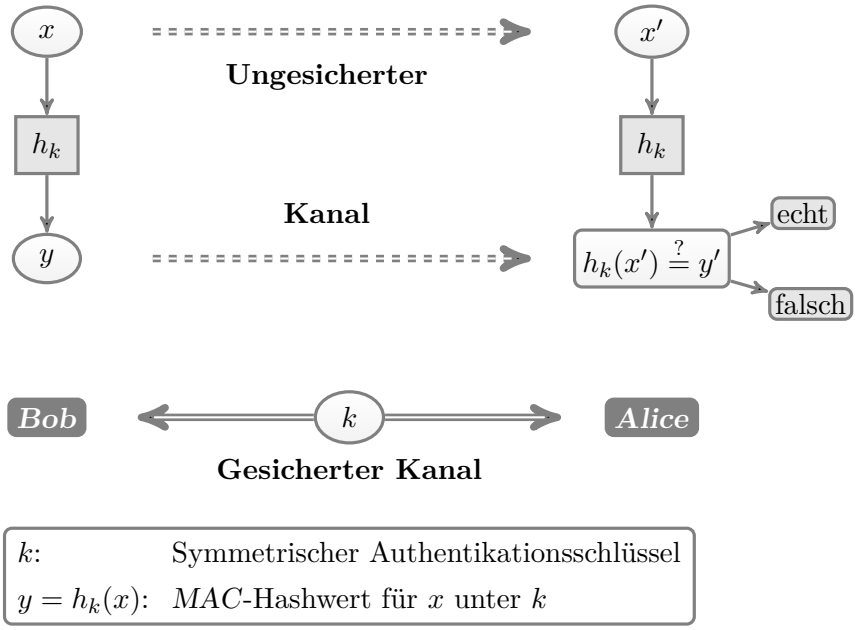

Abbildung 1.3: Verwendung eines MAC zur Nachrichtenauthentikation.

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.

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) else return(?)
```

Abbildung 1.4: Bestimmung eines Urbilds für einen Hashwert

1.2.1 Das Zufallsorakelmodell (ZOM)

Das ZOM dient dazu, die Effizienz 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 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).

Dass eine Zufallsfunktion h gute kryptografische Eigenschaften aufweist, rührt daher, dass der Hashwert $h(x)$ für einen neuen Text x auch dann noch schwer vorhersagbar ist, wenn der Gegner bereits die Hashwerte einer beliebigen Zahl von Texten kennt.

Proposition 1. *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 1.4 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) wider. Der Zeitaufwand der Berechnung ist dabei proportional zu q.

Satz 2. FindPreimage(h, y, q) *gibt 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 1 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 1.5 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.

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 1.5: Bestimmung eines 2. Urbilds für einen Hashwert

Satz 3. FindSecondPreimage(h, x, q) *gibt 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 1.6 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 4. Collision(h, q) *gibt mit Erfolgswahrscheinlichkeit*

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

ein Kollisionspaar (x, x') *für* h *aus.*

Prozedur `Collision`(h, q)

```
1  wähle eine beliebige Menge X_0 = {x_1, ..., x_q} ⊆ X − {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 1.6: Bestimmung eines Kollisionspaares

```
1 wähle zufällig x ∈ X
2 x' := A(x)
3 if x' ≠ ? then return(x, x') else return(?)
```

Abbildung 1.7: Reduktion des Kollisionsroblems auf das Problem, ein zweites Urbild zu bestimmen

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

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

□

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

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

Somit erhalten wir die Abschätzung

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

mit $c_\varepsilon = \sqrt{2\varepsilon}$. Für $\varepsilon = 1/2$ ergibt sich also $q \approx \sqrt{m}$. Besitzt also eine binäre Hashfunktion $h\colon \{0,1\}^n \to \{0,1\}^m$ die Hashwertlänge $m = 128$ Bit, so müssen im ZOM $q \approx \cdot 2^{64}$ 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.

1.2.2 Vergleich von Sicherheitsanforderungen

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

Satz 5. *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, 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. Dann ist klar, dass der in Abbildung 1.7 dargestellte Las-Vegas Algorithmus mit Wahrscheinlichkeit ε ein Kollisionspaar ausgibt. □

```
1 wähle zufällig x ∈ X
2 y := h(x)
3 x' := A(y)
4 if x ≠ x' then return(x, x') else return(?)
```

Abbildung 1.8: Reduktion des Kollisionsproblems auf das Urbildproblem

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

Satz 6. *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 1.8 dargestellten Las-Vegas Algorithmus B.
Sei $\mathcal{C} = \{h^{-1}(y) \mid y \in Y\}$. Dann hat B eine Erfolgswahrscheinlichkeit von

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

□

1.2.3 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-Damgard-Konstruktion

Merkle und Damgard 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 $i \in \{1, \ldots, k\}$ 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 $y(x) := 11f(x)$ definierte Funktion, wobei f wie folgt definiert ist:

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

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
$X \hookleftarrow s$	zirkulärer Linksshift um s Stellen

Während die Ganzzahl-Addition bei MD4 und MD5 in *little endian* Architektur (d.h. ein aus 4 Bytes $a_3a_2a_1a_0$, $0 \leq a_i \leq 255$ zusammengesetztes Wort repräsentiert die Zahl $a_0 2^{24} + a_1 2^{16} + a_2 2^8 + a_3$) ausgeführt wird, verwendet SHA-1 eine *big endian* Architektur (d.h. $a_3a_2a_1a_0$, $0 \leq a_i \leq 255$ 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, $j = 0, \ldots, 47$

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

	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 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 $x \in \{0,1\}^*, |x| = n$
$y := x10^k \texttt{bin}_{64}(n)$, $k \in \{0,1,\ldots,511\}$ 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 $H_1H_2H_3H_4$

1.2.8 Die SHA-2-Familie

Im Jahr 2001 veröffentlichte NIST 4 weitere Hashfunktionen der SHA-Familie: SHA-224, SHA-256, SHA-384, and SHA-512. Diese Funktionen werden auch als SHA-2 Hashfunktionen bezeichnet. In 2004 kam noch SHA-224 als fünfte 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) ⊕ ((note) ∧ g)
19       t1 := h + s1 + ch + k[i] + X[i]
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 and 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, and 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 and 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.

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 leicht auf 2^{39} Hashwertberechnungen verbessert.

Aufgrund der erfolgreichen Angriffe auf SHA-0 rieten mehrere Experten von einer weiteren Anwendung des SHA-1 ab. Daraufhin kündigte die amerikanische Behörde NIST an, SHA-1 in 2010 zugunsten der SHA-2 Varianten abzulösen.

In 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 and Frances Yao auf der CRYPTO 2005 eine weitere Reduktion auf 2^{63} Berechnungen.

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.

1.3 Nachrichten-Authentikationscodes (MACs)

Definition 10. *Eine **Hashfamilie** $\mathcal{H} = (\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ 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 $h_k \colon X \to Y$ spezifiziert.*

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 Alice eine Nachricht x zu schicken, die Alice als von Bob 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 (im Folgenden auch Oskar genannt) bekannt ist. Wir nehmen o.B.d.A. an, dass $p(x) > 0$ und $p(k) > 0$ für alle $x \in X$ und alle $k \in K$ gilt.

Erfolgswahrscheinlichkeit für Impersonation

α: Wahrscheinlichkeit mit der sich ein Gegner bei optimaler Strategie als Bob ausgeben kann, ohne dass Alice 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 Alice 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 ein Gegner bei optimaler Strategie Alice den Text x als von Bob stammend zukommen lassen kann.

Beispiel 11. *Sei $K = \{1, 2, 3\}$, $X = \{a, b, c, d\}$ und $Y = \{0, 1\}$.*

		0,1	0,2	0,3	0,4
	$h_k(x)$	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 $\alpha(x, y)$, falls er das Paar (x, y) an Alice sendet.

	0	1
a	0,7	0,3
b	0,25	0,75
c	0,55	0,45
d	0,45	0,55

Folglich ist $\alpha = 0,75$. ◁

Beispiel 12. *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** erhalten wir, indem wir die Zeilen mit den Schlüsseln $k \in K$ und die Spalten mit den Texten $x \in X$ indizieren und in Zeile k und*

Spalte x den Hashwert $h_k(x)$ eintragen.

	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

Angenommen, jeder Schlüssel (a,b) hat die gleiche Wk $p(a,b) = 1/9$. Versucht der Gegner dann eine Impersonation mit dem Paar (x,y), so akzeptieren genau 3 der 9 möglichen Schlüssel dieses Paar. Dies liegt daran, dass in jeder Spalte jeder Hashwert genau dreimal vorkommt. Also gilt $p(x \mapsto y) = 3/9 = 1/3$ für alle Paare $(x,y) \in X \times Y$, was für α ebenfalls den Wert $\alpha = 1/3$ ergibt.

Satz 13. *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 14. *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.

Erfolgswahrscheinlichkeit für Substitution

β: Wahrscheinlichkeit mit der ein Gegner bei optimaler Strategie eine von Bob gesendete Nachricht (x,y) durch eine andere Nachricht (x',y') ersetzen kann, ohne dass Alice dies bemerkt.

Angenommen, Bob sendet die Nachricht (x,y) und der Gegner ersetzt diese durch (x',y'). Dann ist die Erfolgswahrscheinlichkeit des Gegners gleich der bedingten Wk

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

dass ein zufällig gewählter Schlüssel k den Text x' auf y' abbildet, wenn bereits bekannt ist, dass er x auf y abbildet. Falls Bob also das Paar (x, y) sendet, so kann der Gegner bestenfalls die Erfolgswahrscheinlichkeit

$$\beta(x, y) = \max\{p(x' \mapsto y' \,|\, x \mapsto y) \mid x' \in X - \{x\}, y' \in Y\}$$

erzielen. Da Bob auf die Wahl von (x, y) keinen Einfluss hat, berechnet sich β als der erwartete Wert von $\beta(x, y)$, wobei das Paar (x, y) von Bob mit Wk

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

gesendet wird. Somit ergibt sich β zu

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

wobei

$$\beta'(x, y) = \max\{p(x \mapsto y, x' \mapsto y') \mid x' \in X - \{x\}, y' \in Y\}$$

ist.

Beispiel 15.

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

Für β erhalten wir also den Wert

$$\begin{aligned} \beta &= 0,1 \cdot (0,45 + 0,3) + 0,2 \cdot (0,25 + 0,45) + 0,3 \cdot (0,55 + 0,45) + 0,4 \cdot (0,45 + 0,55) \\ &= 0,915. \end{aligned}$$

Satz 16. *Für jeden MAC (X, Y, K, H) 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 für alle (x, y) mit $p(x, y) > 0$,

$$\beta(x, y) \;=\; \max\{p(x' \mapsto y' \,|\, x \mapsto y) \mid x' \in X - \{x\}, y' \in Y\} \;\geq\; \frac{1}{m}, \qquad (1.2)$$

was wiederum

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

impliziert. □

Lemma 17. *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 im Fall

$$\beta = \frac{1}{m}$$

für alle Paare $(x, y) \in X \times Y$

$$p(x \mapsto y) > 0$$

ist. Ist nämlich

$$p(w \mapsto z) = 0,$$

so ist auch

$$p(w \mapsto z \,|\, u \mapsto v) = 0,$$

wobei $(u, v) \in X \times Y$ ein beliebiges Paar mit

$$p(u \mapsto v) > 0$$

ist. Wegen

$$1 = \sum_{z' \in Y} p(w \mapsto z' \,|\, u \mapsto v) = \sum_{z' \in Y - \{z\}} p(w \mapsto z' \,|\, u \mapsto v)$$

impliziert dies die Existenz eines Hashwertes z' mit

$$p(w \mapsto z' \,|\, u \mapsto v) \geq 1/(m-1) > 1/m.$$

Dann ist aber auch

$$\beta(u, v) = \max\{p(u' \mapsto v' \,|\, u \mapsto v) \mid u' \in X - \{u\}, v' \in Y\} > 1/m.$$

Da

$$\beta(x, y) \geq 1/m$$

für alle Paare (x, y) gilt (siehe (1.2)) und da

$$p(u, v) = p(u)p(u \mapsto v) > 0$$

ist, folgt

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

Ist nun

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

für ein Doppelpaar (x, y, x', y') mit $x \neq x'$, so muss wegen

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

auch ein Doppelpaar (x, z', x', y') mit

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

existieren, was genau wie im ersten Teil des Beweises zu einem Widerspruch führt. □

Satz 18. *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 $\mathcal{H}$ tatsächlich die Bedingung

$$\beta = \frac{1}{m}$$

erfüllt, wenn

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

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

Bemerkung 19. *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 20. *Ein MAC (X, Y, K, H) heißt* 2-***universal****, falls für alle $x, x' \in M$ mit $x \neq x'$ und alle $y, y' \in Y$ gilt:*

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

Bemerkung 21. *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.

Beispiel 22. *Betrachten wir den MAC (X, Y, K, H) mit $X = \{0, 1, 2, 3\}$, $Y = \{0, 1, 2\}$, $K = \{0, 1, \ldots, 8\}$, wobei H durch folgende Authentikationsmatrix beschrieben wird.*

	0	1	2	3
0	0	0	0	0
1	1	1	1	0
2	2	2	2	0
3	0	1	2	1
4	1	2	0	1
5	2	0	1	1
6	0	2	1	2
7	1	0	2	2
8	2	1	0	2

Da in je zwei Spalten jedes Hashwertpaar genau einmal vorkommt, ist (X, Y, K, H) ein $(4, 3, 9, 1)$-MAC.

Auf Grund von Bemerkung 19 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 14 nimmt in diesem Fall auch α den optimalen Wert $\frac{1}{m}$ an.
Der nächste Satz zeigt für primes p eine Konstruktionsmöglichkeit von 2-universalen MACs mit dem Parameterwert $\lambda = 1$.

Satz 23. *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), aber eine stärkere Kompression ist mit dem Parameterwert $\lambda = 1$ nicht realisierbar.

Satz 24. *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 weiterhin 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 25. *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 26. *Sei p prim und für $x = (x_1, \ldots, x_\ell) \in \{0,1\}^\ell$ und $k = (k_1, \ldots, k_\ell) \in \mathbb{Z}_p^\ell$ sei*

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

Dann ist (X, Y, K, H) mit $X = \{0,1\}^\ell - \{0^\ell\}$, $Y = \mathbb{Z}_p$ und $K = \mathbb{Z}_p^\ell$ ein $(2^\ell - 1, p, p^\ell, p^{\ell-2})$-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 $x = x_1 \cdots x_\ell$ und $x' = x'_1 \cdots x'_\ell$ zu einer Matrix A zusammen, so ist dies äquivalent zu

$$\begin{pmatrix} x_1 \cdots\ x_\ell \\ x'_1 \cdots\ x'_\ell \end{pmatrix} \cdot \begin{pmatrix} k_1 \\ \vdots \\ k_\ell \end{pmatrix} = \begin{pmatrix} y \\ y' \end{pmatrix}.$$

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

Bemerkung 27. *Obige Konstruktion liefert einen λ-Wert von $\frac{\|K\|}{m^2} = p^{\ell-2}$. Durch Erweiterung von X auf eine geeignete Teilmenge $X' \subseteq \mathbb{Z}_p^\ell$ lässt sich der Textraum von $2^\ell - 1$ auf $\frac{p^\ell-1}{p-1}$ vergrößern (siehe Übungen). Dies führt auf einen beliebig groß wählbaren Kompressionsfaktor von $\frac{p^\ell-1}{p(p-1)}$ bei einem λ-Wert von $\lambda = p^{\ell-2}$. 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 28. *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 29. *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 x_i die Anzahl der Einsen in dieser Zeile (also $x_1 = n$). Da in jeder Spalte jeder Hashwert genau λm-mal vorkommt, gilt

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

Nun ist die Anzahl z der Vorkommen von Indexpaaren (j, j') mit $A[i, j] = A[i, j'] = 1$ in den Zeilen $i = 2, \ldots, l$ gleich

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

Mit obigem Lemma ergibt sich

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

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, ist die Anzahl z der Vorkommen von Indexpaaren (j, j') mit $A[i, j] = A[i, j'] = 1$ in den Zeilen $i = 2, \ldots, l$ gleich

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

Somit ergibt sich

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

□

Für den Beweis des nächsten Satzes benötigen wir folgendes Lemma (Beweis siehe Übungen).

Lemma 30. *Sei $\mathcal{X}$ eine Zufallsvariable mit endlichem Wertebereich $W(\mathcal{X}) \supseteq \mathbb{R}^+$. Dann gilt $\log E(\mathcal{X}) \geq E(\log \mathcal{X})$.*

Satz 31. *Für jeden MAC (X, Y, K, H) gilt:*

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

Hierbei sind $\mathcal{X}, \mathcal{Y}, \mathcal{K}$ Zufallsvariablen, die die Verteilungen der Nachrichten, der Hashwerte und der Schlüssel beschreiben.

Beweis. Wir zeigen: $\log \alpha \geq H(\mathcal{K} \mid \mathcal{X}, \mathcal{Y}) - H(\mathcal{K})$. Es gilt: $\alpha = \max_{x,y} p(x \mapsto y)$, wobei

$$\begin{aligned}
p(x \mapsto y) &= Prob_k[h_k(x) = y] \\
&= Prob[\mathcal{Y} = y \mid \mathcal{X} = x] \\
&=: p_{y|x} \\
\Rightarrow \alpha &\geq \sum_{x,y} Prob[\mathcal{X} = x, \mathcal{Y} = y] \cdot p(x \mapsto y) \\
&= E(\alpha(\mathcal{X}, \mathcal{Y})) \\
\Rightarrow \log \alpha &\geq \log E(\alpha(\mathcal{X}, \mathcal{Y})) \\
&\geq E(\log \alpha(\mathcal{X}, \mathcal{Y}))(*) \\
&= \sum_{x,y} p_{x,y} \cdot \log p_{y|x} \\
&= \sum_{x,y} p_x \cdot p_{y|x} \cdot \log p_{y|x} \\
&= -H(\mathcal{Y} \mid \mathcal{X}) \\
&\geq H(\mathcal{K} \mid \mathcal{X}, \mathcal{Y}) - H(\mathcal{K})(**)
\end{aligned}$$

Hierbei gilt (*) wegen obigem Lemma und (**) ergibt sich aus

$$\begin{aligned}
H(\mathcal{K}, \mathcal{Y}, \mathcal{X}) &= H(\mathcal{X}) + H(\mathcal{Y} \mid \mathcal{X}) + H(\mathcal{K} \mid \mathcal{X}, \mathcal{Y}) \\
&= \underbrace{H(\mathcal{K}, \mathcal{X})}_{=H(\mathcal{K})+H(\mathcal{X})} + \underbrace{H(\mathcal{Y} \mid \mathcal{K}, \mathcal{X})}_{=0}.
\end{aligned}$$

□

1.3.3 MACs auf der Basis einer schlüssellosen Hashfunktion

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 wie folgt modifizieren.

Sei (x, z) gegeben mit $h_k(y(x)) = z$ und sei $y(x) = x\text{pad}(x) = y_1 \dots y_n$. Dann können wir für einen beliebigen String $u \in \{0,1\}^*$ den MAC-Wert $h_k(y(x'))$ für den Text $x' = x\text{pad}(x)u$ wie folgt berechnen. Wegen

$$y(x') = x'\text{pad}(x') = x\text{pad}(x)u\text{pad}(x') = y_1 \dots y_n u\text{pad}(x')$$

lässt sich das Suffix $u\text{pad}(x')$ in eine Folge $u_1 \dots u_m$ von Blöcken u_i der Länge $|u_i| = t$ zerlegen. Setzen wir nun $z_n = z$ und

$$z_{n+i} \quad := \quad h(z_{n+i-1}u_{n+i})$$

für $i = 1, \dots, m$, so erhalten wir den gewünschten MAC-Wert $h_k(y(x')) = z_{n+m}$.

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 endomorphes Kryptosystem (d.h. $M = C$) mit $M = \{0,1\}^t$. 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 $y := y(x) = y_1 \dots y_n$, $n \geq 1$, $|y_i| = t$
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 $i = 1, \dots, n+m$ 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 $q+1 \approx 2^{\frac{t}{2}}$ Hashwertfragen den MAC-Wert $h_k(x)$ für einen zuvor nicht erfragten Text x zu finden, wobei $x = x_1, \ldots, x_n \in \{0,1\}^{tn}$ abgesehen vom ersten t-Bitblock x_1 beliebig wählbar ist. Hierzu wählt der Gegner zunächst $n-2$ beliebige Blöcke $x_3, \ldots, x_n \in \{0,1\}^t$ und $q \approx 1,17 \cdot 2^{\frac{t}{2}}$ paarweise verschiedene Blöcke $x_1^1, \ldots, x_1^q \in \{0,1\}^t$. Anschließend wählt er zufällig q weitere Blöcke $x_2^1, \ldots, x_2^q \in \{0,1\}^t$ und erfragt die MAC-Werte $z_i = h_k(x^i)$ für die Texte $x^i = x_1^i x_2^i x_3 \cdots x_n$, $i = 1, \ldots, q$.

Wegen $x_1^i \neq x_1^j$ für $i \neq j$ sind auch die Texte $x^1, \ldots, x^q$ paarweise verschieden. Seien $z_1^1, \ldots, z_1^q$ 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 für die 2. Iteration mit der Gleichheit der Ausgangsblöcke für die n-te 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 32. *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 $x_1, \ldots, x_q$ 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 $x \notin \{x_1, \ldots, x_q\}$ und $h_k(x) = z$.*

Wir unterscheiden zwischen adaptiven Fragen (d.h. der Text x_i darf von den Hashwerten der Texte $x_1, \ldots, x_{i-1}$ 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 $x \notin \{x_1, \ldots, x_q\}$ generieren, auf dessen Wahl er keinen Einfluss hat).

Beispiel 33. *Der betrachtete 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 34. *Seien $\mathcal{H}_1 = (X, Y, K_1, F)$ mit $F = \{f_k \mid k \in K_1\}$ und $\mathcal{H}_2 = (X, Y, K_2, G)$ 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 35. *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 H-MAC, 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 H-MAC wie folgt:

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

Hierbei fungiert die Funktion $f_k(x) = \text{SHA-1}((k \oplus ipad)x)$ als Hashfunktion mit Schlüssel, die beliebig lange Texte hasht, und der MAC $g_k(x) = \text{SHA-1}((k \oplus ipad)x)$ 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 36. *Ein (ε, q)-Kollisionsangreifer für eine Hashfamilie $\mathcal{H}$ ist ein probabilistischer Algorithmus $\mathcal{A}$, der q Fragen $x_1, \ldots, x_n$ stellt und aus den Antworten 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 37. *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 $\mathcal{H}$. Wir müssen zeigen, dass $\varepsilon \leq \varepsilon_1 + \varepsilon_2$ ist. Wir betrachten zunächst 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 $< \varepsilon_1$ ein. Da A aber ein (ε, q)-Fälscher für $\mathcal{H}$ ist, muss mit Wahrscheinlichkeit $\geq \varepsilon$ $z = g_{k_2}(y)$ gelten. Folglich sind mit Wahrscheinlichkeit $\geq \varepsilon - \varepsilon_1$ die beiden Bedingungen $y \notin \{y_1, \ldots, y_q\}$ und $z = g_{k_2}(y)$ erfüllt. In diesem Fall hat jedoch der adaptive Fälscher A'' gegen $\mathcal{H}_2$ Erfolg, der zufällig einen Schlüssel $k_1 \in K_1$ wählt und A wie folgt simuliert. A'' gibt bei jeder Anfrage x_i von A die Antwort des Orakels g_{k_2} auf die Frage $y_i = f_{k_1}(x_i)$ zurück und sobald A ein Paar (x, z) ausgibt, gibt A'' das Paar $(f_{k_1}(x), z)$ aus. Da 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 38. *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 $\mathcal{O}$. Im Fall $4a^3 + 27b^2 = 0$ heißt E singulär, sonst nicht-singulär.*

Auf den nicht-singulären Punkten von E lässt sich eine additive Gruppenoperation $+$ definieren. Die Idee dabei ist, dass die Summe aller Punkte von E, die auf einer Geraden liegen gleich dem neutralen Element $\mathcal{O}$ sein soll (hierbei werden Tangentialpunkte doppelt gezählt). Da maximal drei Punkte von E auf einer Geraden liegen können wir auf der Basis dieser Regel die Summe $P + Q$ zweier Punkte P und Q leicht bestimmen.

Am einfachsten ist der Fall, dass die Gerade g parallel zur y-Achse verläuft. Besteht die Schnittmenge S von g und E 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 $P + Q = \mathcal{O}$ 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.

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

$P \neq Q$, d.h. $x_1 \neq x_2$: Dann 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:

$$x_3 = \lambda^2 - x_1 - x_2 \text{ und } y_3 = \lambda(x_3 - x_1) + y_1, \text{ mit } \lambda = \frac{y_2 - y_1}{x_2 - x_1}.$$

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 $x_1, x_2 \in \mathbb{R}$ sind, muss auch $x_3 \in \mathbb{R}$ sein. Der Koeffizient $-\lambda^2$ von x^2 berechnet sich aus der linearen Zerlegung von $p(x)$ zu

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

Wegen $\lambda = \frac{y_3 - y_1}{x_3 - x_1}$ erhalten wir dann $y_3 = \lambda(x_3 - x_1) + y_1$.

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

$P = Q$**, d.h.** $x_1 = x_2, y_1 = y_2 \neq 0$**:** Sei t die Tangente durch P an E. Wir zeigen, dass es einen Punkt $R = (x_3, y_3) \in \mathbb{R}^2$ gibt mit

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

wobei $x_3 = \lambda^2 - 2x_1$ und $y_3 = \lambda(x_3 - x_1) + y_1$ ist. Die Steigung λ von t erhalten wir durch implizites Differenzieren:

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

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

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

und

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

t ist dann der Schnitt von T mit der x, y-Ebene, d.h.

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

woraus sich $\lambda = -\frac{c}{d}$ erbibt. 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 39. *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 40. *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. $\forall P \in E : P + \mathcal{O} = \mathcal{O} + P = P$.*

- *Für $P, Q \in E - \{\mathcal{O}\}$ 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:

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

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

Beweis. ohne Beweis □

Beispiel 42. *$p = 11$, 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
$z \in QR_{11}$	–	–	x	x	–	x	–	x	x	–	x
y	–	–	4; 7	5; 6	–	2; 9	–	2; 9	3; 8	–	2; 9

Da die Gruppe $(E, \mathcal{O}, +)$ $\sharp E = \|E\| = 13$ 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 $\mathbb{Z}_{13}$: $(E, \mathcal{O}, +) \cong (\mathbb{Z}_{13}, 0, +)$. Da die Gruppenordnung prim ist, ist sogar jedes Element $p \in E - \{\mathcal{O}\}$ ein Erzeuger. Folglich

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

$$q + 1 - 2\sqrt{q} \leq \sharp E \leq q + 1 + 2\sqrt{q}.$$

Beweis. (ohne Beweis) □

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

Prozedur `PointDeCompress`(x, i)

```
1 z := x^3 + ax + b mod p
2 if z ∈ QR_p then
3   y := √z mod p
4   if y ≢_2 i then
5     y := p − y
6     output (x, y)
7 output (''error'')
```

Bemerkung 44. *Es gibt einen effizienten Algorithmus (von Schoof) mit Zeitkomplexität $O(\log^8 q)$, der $\sharp E$ bei Eingabe von a, b und q berechnet.*

Satz 45. *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_2 Teiler von n_1 und von $q - 1$ ist.*

Bemerkung 46. *Zyklische (Unter)-Gruppe. Wegen $\sharp E = n_1 \cdot n_2$ und da n_2 Teiler von n_1 ist, muss E im Fall, dass $\sharp E$ prim oder das Produkt von zwei verschiedenen Primzahlen ist, zyklisch sein (d.h. $n_2 = 1$). Im Fall $n_2 > 1$ hat E eine nicht-triviale zyklische Untergruppe, die zu $\mathbb{Z}_{n_1}$ isomorph ist und für kryptografische Anwendungen benutzt werden kann.*

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 additiv Inverse sehr leicht zu berechnen sind, kann mP durch 'wiederholtes Verdoppeln, Addieren und Subtrahieren' noch effizienter berechnen werden. Hierzu stellen wir m in NAF (non adjacent form) dar.

Definition 47. *$(c_{l-1}, \ldots, c_0) \in \{-1, 0, 1\}^l$ heißt SBR-Darstellung (signial binary representation) einer Zahl $c \in \mathbb{Z}$, falls*

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

ist. Ist von je zwei benachbarten c_i's mindestens eines 0, so heißt $(c_{l-1}, \ldots, c_0)$ NAF-Darstellung von c.

Beispiel 48. *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 49. *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 $(1, 0, \ldots, 0, -1)$.

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
5        Q := Q + P
6  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 um 11 Prozent effizienter als DoubleAdd.

3 Algorithmen zur Berechnung des diskreten Logarithmus

Sei $(G, *, 1)$ eine Gruppe und sei $\alpha \in G$. Weiter bezeichne $\langle\alpha\rangle = \{\alpha^i | i = 0 \cdots n-1\}$ die von α in G erzeugte Untergruppe, wobei $n = \mathrm{ord}_G(a) = \min\{e \geq 1 \mid \alpha^e = 1\}$ die Ordnung von α ist. Dann heißt die eindeutig bestimmte Zahl $e \in \{0, \ldots, n-1\}$ mit $\beta = \alpha^e$ der **diskrete Logarithmus** von β zur Basis α in G (kurz: $e = \log_{G,\alpha}(\beta)$).

Das diskrete Logarithmusproblem (DLP):

Gegeben: Gruppe G, ein Element $\alpha \in G$ und die Ordnung $n = \mathrm{ord}_G(a)$ von α sowie ein Element $\beta \in \langle\alpha\rangle$.

Gesucht: Der diskrete Logarithmus $e = \log_{G,\alpha}(\beta)$ von β zur Basis α in G.

Für viele Gruppen G ist die Funktion $e \mapsto \alpha^e$ effizient mittels wiederholtem Quadieren und Multiplizieren berechenbar. In einigen Fällen ist jedoch kein effizienter Algorithmus zur Bestimmung der Umkehrfunktion, also von $\log_\alpha(\beta)$ bekannt, d.h. $e \mapsto \alpha^e$ ist Kandidat für eine Einwegfunktion.

Beispiel 50. *Sei $G = (\mathbb{Z}_p^*, *)$, p prim, und sei α ein Erzeuger von $\mathbb{Z}_p^*$. Dann ist $\langle\alpha\rangle = \mathbb{Z}_p^*$ und α hat die Ordnung $n = p-1$. Ist p hinreichend groß und enthält $p-1$ mindestens einen großen Primfaktor, so sind keine effiziente Algorithmen zur Berechnung von $\log_{p,\alpha}(\beta)$ bekannt.*

Die Ordnung der Potenzen eines Elements $\alpha \in G$ der Ordnung n lässt sich wie folgt berechnen:

$$\mathrm{ord}_G(\alpha^i) = n / \mathrm{ggT}(n, i).$$

Ist insbesondere q ein Teiler von n, so hat $\alpha^{n/q}$ die Ordnung q.

Wir betrachten zunächst eine Reihe von naiven Algorithmen für das DLP.

Berechnung von $\log_{G,\alpha}(\beta)$

```
1  γ := 1
2  for i := 0 to n − 1 do
3     if γ = β then output(i)
4     γ := αγ
```

Dieser Algorithmus läuft in Zeit $\mathcal{O}(n)$ (wobei wir annehmen, dass elementare Gruppenoperationen in konstanter Zeit ausführbar sind) und benötigt nur logarithmischen Speicherplatz. Falls wir im Vorfeld eine Tabelle mit den Logarithmen aller möglichen Werte für β erstellen, können wir danach für jedes β den diskreten Logarithmus durch eine Binärsuche in Zeit $O(\log n)$ bestimmen. Für die Precomputation fallen jedoch Zeit $\mathcal{O}(n)$ und Platz $\mathcal{O}(n \log n)$ an.

DLP-Berechnung mittels Precomputation

```
1  Precomputation: Sortiere die Paare (α^i, i), i = 0, ..., n − 1, nach der
       ersten Komponente in eine Tabelle T
```

```
2  Computation: Ermittle in T mittels Binärsuche den Eintrag (β, i)
       und gib i aus
```

Der folgende Algorithmus von Shanks berechnet ebenfalls im Vorfeld eine Tabelle von DLP-Werten, allerdings nur für Potenzen der Form $\alpha^{jm}, j = 0, \ldots, m-1$, wobei $m = \lceil\sqrt{n}\rceil$ ist. Dadurch erhöht sich zwar die Laufzeit zur Bestimmung des diskreten Logarithmus für β von $O(\log n)$ auf $O(\sqrt{n})$, im Gegenzug geht jedoch der Speicherplatzverbrauch von $\mathcal{O}(n \log n)$ auf $\mathcal{O}(\sqrt{n} \log n)$ zurück.

Algorithmus `Shanks`(G, n, α, β)

```
1  Precomputation:
2    m := ⌈√n⌉
3    sortiere die Paare (α^{jm}, j), j = 0, ..., m − 1, nach der ersten
       Komponente in eine Tabelle T1
4  Computation:
5    sortiere die Paare (βα^{−i}, i) nach der ersten Komponente in eine
       Tabelle T2
6    ermittle durch parallele sequentielle Suche Paare (γ, j) in T1
       und (γ, i) in T2 mit derselben ersten Komponente
7    output(mj + i)
```

3.1 Die Rho-Algorithmen von Pollard

Von Pollard wurde eine heuristische Strategie entwickelt, die sich sowohl zur Lösung des DLP als auch des Faktorisierungsproblems eignet. Die Idee dabei ist, mit wenig Speicherplatz eine Kollision $a_i = a_j$ mit $i \neq j$ für eine Folge (a_n) der Form $a_{n+1} = f(a_n)$ zu finden. Zahlenfolgen dieser Bauart haben die Eigenschaft, dass $a_i = a_j$ die Gleichheit $a_{i+k} = a_{j+k}$ für alle $k \geq i$ impliziert.

Der Rho-Faktorisierungsalgorithmus

Sei n eine Zahl mit mindestens 2 verschiedenen Primteilern $p < q$ (falls n nur einen Primteiler hat, also eine Primzahlpotenz ist, lässt sich n leicht durch Berechnung der k-ten Wurzeln für $k = 2, \ldots, \log_2(n)$ faktorisieren).

Angenommen, wir wählen zufällig eine Menge $X \subseteq \mathbb{Z}_n$ der Größe $\sqrt{p}$, so enthält X mit großer Wahrscheinlichkeit 2 Elemente $x \neq x'$ mit $x \equiv_p x'$, die auf den nichttrivialen Faktor $d = \mathrm{ggT}(x - x', n)$ von n führen.

Wählen wir nun anstelle von X eine pseudozufällige Menge der Form $X = \{x_1, x_2 = f(x_1), \ldots, x_j = f(x_{j-1})\}$, wobei x_1 ein zufällig gewählter Startwert ist, so tritt bei geeigneter Wahl von $f : \mathbb{Z}_n \to \mathbb{Z}_n$ für $j \approx \sqrt{p}$ mit großer Wahrscheinlichkeit eine Kollision auf. Eine gute Wahl für f ist beispielsweise $f(x) = x^2 \pm 1 \bmod n$.

Werden zur Berechnung von f nur die Ringoperationen von $\mathbb{Z}_n$ verwendet, so impliziert $x_i \equiv_p x_j$ die Kongruenz $f(x_i) \equiv_p f(x_j)$, was wiederum für $l = j - i$ die Kongruenz $x_k \equiv_p x_{k+dl}$ für alle $k \geq i$ und $d \geq 1$ impliziert. Insbesondere folgt also $x_k \equiv_p x_{2k}$ für alle $k \geq i$ mit $k \equiv_l 0$. Daher können wir in X ein Kollisionspaar (x_i, x_j) mit $x_i \equiv_p x_j$ wie folgt bestimmen (ohne p zu kennen).

Algorithmus `Pollard-Rho-Factorize`(n)

```
1  wähle zufällig x ∈ Z_n
2  y := x^2 + 1 mod n
3  while ggT(x − y, n) = 1 do
4     x := f(x)
5     y := f(f(y))
6  if d = ggT(x − y, n) < n then output(d)
7  else output(?)
```

Der Rho-DLP-Algorithmus

Dieser Algorithmus berechnet eine pseudozufällige Folge von Paaren $(c_i, d_i) \in \mathbb{Z}_n \times \mathbb{Z}_n$. Ziel ist es, zwei Paare verschiedene (c_i, d_i) und (c_j, d_j) mit $\alpha^{c_i}\beta^{d_i} = \alpha^{c_j}\beta^{d_j}$ zu finden. Im Fall $\mathrm{ggT}(d_j - d_i, n) = 1$ lässt sich hieraus wegen

$$\alpha^{c_i + ad_i} = \alpha^{c_i}\beta^{d_i} = \alpha^{c_j}\beta^{d_j} = \alpha^{c_j + ad_j}$$

der diskrete Logarithmus $\log_{G,\alpha}(\beta) = (c_i - c_j)(d_j - d_i)^{-1} \bmod n$ leicht bestimmen. Andernfalls erhalten wir $g = \mathrm{ggT}(f - d, n)$ Kandidaten $a_1, \ldots, a_g$, unter denen der richtige leicht zu ermitteln ist. Zur Bildung der Pseudozufallsfolge kann bspw. die Funktion f in folgendem Algorithmus benutzt werden. Aus Effizienzgründen berechnet sie auch gleich die Werte $x_i = \alpha^{c_i}\beta^{d_i}$. Die Mengen S_1, S_2, S_3 bilden eine Partition von G in drei etwa gleich große Mengen, wobei das neutrale Element 1 von G nicht in S_2 enthalten sein sollte.

Algorithmus `Pollard-Rho-DLP`(G, n, α, β)

```
1  function f(x, c, d)
2    case
3      x ∈ S_1: return(βx, c, d + 1 mod n)
4      x ∈ S_2: return(x^2, 2c mod n, 2d mod n)
5      x ∈ S_3: return(αx, c + 1 mod n, d)
6
7  wähle zufällig c, d ∈ Z_n
8  x := α^c β^d
9  (y, e, f) := f(x, c, d)
10 while x ≠ y do
11    (x, c, d) := f(x, c, d)
12    (y, e, f) := f(f(y, e, f))
13 g := ggT(f − d, n)
14 bestimme alle Lösungen a_1, ..., a_g von (f − d)a ≡_n (c − e)
15 output a_i mit α^{a_i} = β
```

Ähnlich wie beim Rho-Faktorisierungsalgorithmus lässt sich argumentieren, dass die while-Schleife nach ca. $\sqrt{n}$ Iterationen abbricht.

3.2 Der Pohlig-Hellman-Algorithmus

Sei $n = \prod_{i=1}^{k} p_i^{e_i}$ die Primfaktorzerlegung von n und sei $a = \log_{G,\alpha}(\beta)$ der diskrete Logarithmus von β zur Basis α in G. Falls wir für $i = 1, \ldots, k$ die Werte $x_i = a \bmod p_i^{e_i}$

kennen, so lässt sich daraus a leicht mit dem Chinesischen Restsatz berechnen. Schreiben wir x_i als Zahl zur Basis p_i, so erhalten wir Ziffern $a_0, \ldots, a_{c_i-1}$ mit $x_i = \sum_{j=0}^{c_i-1} a_j p_i^j$. Weiter ex. eine Zahl s_i mit $a = x_i + s_i p_i^{e_i}$.

Um nun die Ziffern $a_0, \ldots, a_{c_i-1}$ zu berechnen, betrachten wir für $j = 0, \ldots, e_i - 1$ und $\beta_j = \beta\alpha^{-a_0 - a_1 p_i - a_2 p_i^2 \cdots - a_{j-1} p_i^{j-1}}$ die Gleichung

$$\beta_j^{n/p_i^{j+1}} = \alpha^{a_j n/p_i},$$

die sich leicht verifizieren lässt:

$$\begin{aligned}
\beta_j^{n/p_i^{j+1}} &= (\alpha^{a - a_0 - a_1 p_i - a_2 p_i^2 \cdots - a_{j-1} p_i^{j-1}})^{n/p_i^{j+1}} \\
&= (\alpha^{a_j p_i^j + a_{j+1} p_i^{j+1} + \cdots + a_{c_i-1} p_i^{c_i-1} + s_i p_i^{c_i}})^{n/p_i^{j+1}} \\
&= (\alpha^{a_j p_i^j + t p_i^{j+1}})^{n/p_i^{j+1}} \text{ für eine Zahl } t \\
&= \alpha^{a_j n/p_i} \alpha^{tn} \\
&= \alpha^{a_j n/p_i}
\end{aligned}$$

Der folgende Algorithmus berechnet sukzessive die Zahlen β_j und dazu die Ziffern $a_j = \log_{G,\alpha^{n/p_i}}(\beta^{n/p_i^{j+1}})$, die sich wegen $\mathrm{ord}_G(\alpha^{n/p_i}) = p_i$ in Zeit $\mathcal{O}(\sqrt{p_i})$ (etwa mit dem Algorithmus von Shanks) ermitteln lassen. Insgesamt erhalten wir somit eine Laufzeit von $\mathcal{O}(c_i\sqrt{p_i})$ zur Bestimmung von x_i.

Algorithmus `Pohlig-Hellman-DLP`$(G, n, \alpha, \beta, p_i, e_i)$

```
1  for j := 0 to e_i − 1 do
2    a_j := log_{G,α^{n/p_i}}(β^{n/p_i^{j+1}})
3    β := βα^{−a_j p_i^j}
4  output(a_0 ··· a_{c_i−1})
```

3.3 Die Index-Calculus-Methode

Hierbei handelt es sich nicht um einen generischen DLP-Algorithmus, da er nur im Fall $G = \mathbb{Z}_p^*$, p prim, und $\mathrm{ord}(\alpha) = p - 1$ anwendbar ist. Der Algorithmus benutzt eine Faktorbasis $B = \{p_1, \ldots, p_b\}$, wobei wir annehmen, dass B die ersten b Primzahlen enthält.

Algorithmus `Index-Calculus`(p, α, β)

```
1  Precomputation:
2    bestimme l_i = log_α p_i für i = 1, ..., b
3  Computation:
4    wähle zufällig eine Zahl s ∈ {0, ..., p − 2}
5    γ := βα^s mod p
6    if (γ ist über B faktorisierbar) then
7      berechne Exponenten c_1, ..., c_b mit γ = p_1^{c_1} ··· p_b^{c_b}
8      output (c_1 l_1 + ··· + c_b l_b mod p − 1)
```

Zur Bestimmung der Zahlen l_i kann man wie folgt vorgehen. Wähle c etwas größer als b (z.B. $c = b + 10$) und generiere c Kongruenzen der Form

$$\alpha^{x_j} \equiv_p p_1^{a_{1j}} \cdots p_b^{a_{bj}}, j = 1, \ldots, c.$$

Hierzu kann man x_j zufällig wählen und testen, ob $y_j = \alpha^{x_j} \bmod p$ über B faktorisierbar ist. Die Wahrscheinlichkeit hierfür hängt natürlich von der Größe von B ab. Aus den Kongruenzen lässt sich ein lineares Kongruenzgleichungssystem der Form

$$\underbrace{\begin{pmatrix} a_{11} & \cdots & a_{b1} \\ & \ddots & \\ a_{1c} & \cdots & a_{bc} \end{pmatrix}}_{A} \begin{pmatrix} l_1 \\ \vdots \\ l_b \end{pmatrix} \equiv_{p-1} \begin{pmatrix} x_1 \\ \vdots \\ x_c \end{pmatrix}$$

für die Unbekannten $l_1, \ldots, l_b$ gewinnen, das die gewünschten Werte liefert, falls A durch Streichen von $c - b$ Zeilen in eine $b \times b$-Matrix A' mit $\det A' \not\equiv_{p-1} 0$ transformiert werden kann.

Durch eine heuristische Komplexitätsanalyse lässt sich zeigen, dass die Precomputation-Phase in Zeit $\mathcal{O}(e^{(1+o(1))\sqrt{\ln p \ln\ln p}})$ und die Computation-Phase in Zeit $\mathcal{O}(e^{(1/2+o(1))\sqrt{\ln p \ln\ln p}})$ ausführbar ist.

3.4 Eine untere Komplexitätsschranke für generische DLP-Algorithmen

In diesem Abschnitt gehen wir der Frage nach, wie effizient der diskrete Logarithmus $\log_{\alpha,G} \beta$ berechenbar ist, wenn über die Gruppe G nichts bekannt ist, außer dass $\alpha \in G$ die Ordnung n hat und $\beta \in \langle\alpha\rangle$ ist. Ein Algorithmus, der das DLP unter dieser Voraussetzung löst, heißt **generisch**.

Dabei nehmen wir an, dass sich die Gruppenoperation und auch die Potenzierung von Elementen in $\langle\alpha\rangle$ effizient ausführen lassen. Um zu verhindern, dass der DLP-Algorithmus spezielle Eigenschaften von G ausnützen kann (bspw. lässt sich das DLP in der additiven Gruppe $(\mathbb{Z}_n, +)$ sehr effizient lösen), gehen wir davon aus, dass die Elemente von $\langle\alpha\rangle$ durch beliebige Binärstrings kodiert sind. Formal verwenden wir hierzu eine injektive Kodierungsfunktion $\sigma : \mathbb{Z}_n \to \{0,1\}^l$, die jedem Exponenten $i \in \mathbb{Z}_n$ eine (zufällig gewählte) Kodierung $\sigma(i) \in \{0,1\}^l$ des Elements α^i zuweist.

Da ein generischer DLP-Algorithmus A zu Beginn der Rechnung nur die (Kodierungen der) beiden Elemente α und β kennt, kann er durch wiederholte Ausführung von Gruppen- und Potenzoperationen nur Elemente der Form $\alpha^c\beta^d$ berechnen. Der Einfachheit halber nehmen wir an, dass A die Möglichkeit hat, für beliebige Paare $(c,d) \in \mathbb{Z}_n \times \mathbb{Z}_n$ die Kodierung von $\alpha^c\beta^d$ in einem Rechenschritt zu erfragen. Seien also $\mathcal{C} = \{(c_1,d_1), \ldots, (c_m,d_m)$ die während seiner Rechnung erfragten Paare.

Offensichtlich kann A den gesuchten Wert $a = \log_{\alpha,G} \beta$ genau dann berechnen, wenn er für zwei verschiedene Paare (c_i,d_i) und (c_j,d_j) die gleiche Antwort erhält. Wegen $\alpha^{c_i}\beta^{d_i} = \alpha^{c_j}\beta^{d_j}$ gilt dann nämlich $\alpha^{c_i-c_j} = \beta^{d_j-d_i}$ und daher

$$a = (c_i - c_j)(d_j - d_i)^{-1} \bmod n.$$

Genauer folgt aus der Gleichheit $\alpha^{c_i-c_j} = \beta^{d_j-d_i}$, dass $a = (c_i - c_j)(d_j - d_i)^{-1} \bmod n$, und aus $\alpha^{c_i-c_j} \neq \beta^{d_j-d_i}$, dass $a \not\equiv_n (c_i - c_j)(d_j - d_i)^{-1}$ ist. Falls also A auf $n-1$ Fragen lauter verschiedene Antworten erhalten hat, lässt sich daraus auch auf den Wert von a schließen.

Wir betrachten zuerst den einfacheren Fall, dass A nur nichtadaptive Fragen stellt (bspw. ist der Shanks-Algorithmus ein nichtadaptiver generischer DLP-Algorithmus). Weiterhin nehmen wir an, dass β (und damit der Wert $a = \log_{\alpha,G}$) zufällig gewählt wird.

Sei $\mathit{Good}(\mathcal{C}) = \{(c_i - c_j)(d_j - d_i)^{-1} \bmod n \mid 1 \le i < j \le m\}$. Dann kann A den Wert a im Fall $a \in \mathit{Good}(\mathcal{C})$ mit Sicherheit bestimmen. Dagegen gelingt dies A im Fall $a \notin \mathit{Good}(\mathcal{C})$ nur mit Wk $(n-m)^{-1}$, und zwar unabhängig davon, nach welcher Strategie A den Ausgabewert aus der Menge $\mathbb{Z}_n - \mathit{Good}(\mathcal{C})$ auswählt. Somit gilt für die Erfolgswk γ von A,

$$\begin{aligned}\gamma &\le \Pr[A \text{ gibt } a \text{ aus} \mid a \in \mathit{Good}(\mathcal{C})] \cdot \delta + \Pr[A \text{ gibt } a \text{ aus} \mid a \notin \mathit{Good}(\mathcal{C})] \cdot (1-\delta) \\ &\le \delta + (n-m)^{-1}(1-\delta) = m/n + 1/n \le \frac{\binom{m}{2} + 1}{n},\end{aligned}$$

wobei $\delta = \Pr[a \in \mathit{Good}(\mathcal{C})] = m/n$ ist. Um also eine Erfolgswk $\gamma = \Omega(1)$ zu erreichen, muss A mindestens $m = \Omega(\sqrt{n})$ Fragen stellen.

Abschließend betrachten wir den Fall, dass A adaptive Fragen stellt. Da die Antworten zufällig gewählte Binärstrings sind, können sie offensichtlich nicht bei der Suche nach nachfolgenden Fragen von Nutzen sein. Der einzige Vorteil, den ein adaptiver generischer DLP-Algorithmus A hat, besteht darin, dass er sofort die Rechnung beenden kann, sobald er auf zwei verschiedene Fragen die gleiche Antwort erhält. Legen wir aber von vornherein eine Obergrenze für die Anzahl der Fragen fest, so ergibt sich genau die gleiche Erfolgswk wie im nichtadaptiven Fall.

4 Digitale Signaturverfahren

Handschriftliche Signaturen

- Die durch die Unterschrift gekennzeichnete Person hat überprüfbar die Unterschrift geleistet.
- Die Unterschrift ist nicht auf ein anderes Dokument übertragbar.
- 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 das elektronische (digitale) Dokument gebunden.

Definition 51. *Ein digitales Signaturverfahren besteht aus:*

- *endlicher Menge X von Dokumenten,*
- *endlicher Menge Y von Unterschriften,*
- *endlicher 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): d.h. 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): d.h. der Gegner kann für Dokumente seiner Wahl die zugehörigen Signaturen bestimmen (eventuell mit Hilfe des legalen Unterzeichners).

nichtselektives (existentielles) Fälschungsvermögen: d.h. 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 $sig(d,n,x) := x^d \bmod n$, $x \in X = \mathbb{Z}_n$ und die Verifikationsbedingung ist

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

Satz 52. *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. □

Es ist nicht schwer, eine nichtselektive Fälschung beibekanntem Verifikationsschlüssel durchzuführen. Hierzu wählt der Gegner zu einer beliebigen Signatur $y \in Y$ das Dokument $x = y^e \bmod n$. Diesen Angriff kann man vereiteln, indem man das Dokument x mit Redundanz versieht (z.B. anstelle von x den Text xx signiert). Effizienter ist es jedoch, nicht das gesamte Dokument x, sondern nur den Hashwert $h(x)$ zu signieren.

Von h benötigte Eigenschaften

- h sollte eine Einweg-Hashfunktion sein, da sonst der Gegner zu $h(x)$ ein passendes x 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, da sonst der Gegner ein Kollisonspaar (x,x') für h berechnen kann (und sich zu x die zugehörige Signatur beschaffen). Dann gilt $ver(k,x',y) = 1$.

Wird keine Hashfunktion benutzt, sind noch weitere Angriffe möglich.

- Falls der Gegner zwei signierte Dokumente $(x_1,y_1),(x_2,y_2)$ mit $ver(k,x_i,y_i) = 1$ kennt, so ist eine existentielle Fälschung bei bekannten Signaturen möglich: Wegen $y_i^e \equiv_n x_i$ für $i = 1,2$ folgt nämlich $(y_1y_2)^e \equiv_n y_1^e y_2^e \equiv_n x_1x_2$ und somit $ver(k, x_1x_2 \bmod n, y_1y_2 \bmod n) = 1$.
- Weiterhin ist eine selektive Fälschung bei frei wählbaren Dokumenten denkbar. Kennt der Gegner bereits die Signatur für ein beliebiges Dokument $x' \in \mathbb{Z}^*_n$ und kann er sich die Signatur für das Dokument $x'' = x \cdot x'^{-1} \bmod n$ beschaffen, so kann er daraus wie oben die Signatur für das Dokument x berechnen.

4.1 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 El Gamal. 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 $a \in \mathbb{Z}_p^* = \{1, \ldots, p-1\}$ 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 $x \in \mathbb{Z}_{p-1}$ zu signieren, wählt der Signierer zufällig eine Zahl $z \in \mathbb{Z}_{p-1}^*$ und berechnet $sig(\hat{k}, x, z) = (\gamma, \delta)$ 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 53. *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 $\beta^\gamma \gamma^\delta \equiv_p \alpha^{a\gamma} \alpha^{z\delta} \equiv_p \alpha^{a\gamma + z\delta} \equiv_p^{(*)} \alpha^x$. Da α ein Erzeuger von $\mathbb{Z}_p^*$ ist, gilt die Kongruenz $(*)$ genau dann, wenn $a\gamma + z\delta \equiv_{p-1} x$ ist. □

Zur Sicherheit des El Gamal-Systems

1. Falls Oskar 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) Oskar 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 Instanz des dikreten Logarithmus.
   b) Oscar 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) Oscar wählt γ und δ gleichzeitig. Auch hierfür ist kein effizientes Verfahren bekannt.
3. Versucht Oskar 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. Wir können jedoch eine existentielle Fälschung wie folgt durchführen. Wähle beliebige Zahlen $i \in \mathbb{Z}_{p-1}$, $j \in \mathbb{Z}_{p-1}^*$ und setze $\gamma = \alpha^i \beta^j \bmod p$. Dann ist $(x, (\gamma, \delta))$ genau dann eine gültige Signatur, wenn $\alpha^x \equiv_p \beta^\gamma (\alpha^i \beta^j)^\delta$ ist. Dies gilt wiederum genau dann, wenn $\alpha^{x - i\delta} \equiv_p \beta^{\gamma + j\delta}$ ist. Diese Bedingung lässt sich erfüllen, indem wir $\delta = -\gamma j^{-1} \bmod p-1$ und $x = i\delta \bmod p-1$ setzen.

Bemerkung 54. *Bei der Benutzung des El Gamal-Signaturverfahrens sind folgende Punkte zu beachten.*

1. *Die Zufallszahl z muss geheim gehalten werden.*
2. *Zufallszahlen dürfen nicht mehrfach verwendet werden.*

Kennt nämlich Oskar zu einer Signatur $(x,(\gamma,\delta))$ die Zufallszahl z, so kann er wegen $\delta \equiv_{p-1} (x-a\gamma)z^{-1}$ die geheime Zahl $a = (-z\delta + x)\gamma^{-1} \bmod p-1$ berechnen.
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.

4.2 Das Schnorr-Signaturverfahren

Da die Primzahl p beim El Gamal-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: $k = (p,\alpha,\beta)$, $\beta = \alpha^a \bmod p$.

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

$$\gamma = h(x \text{ bin}(\alpha^z \bmod p)) \quad \delta = (z + a\gamma) \bmod q \quad ver(k,\gamma,\delta) = \begin{cases} 1, & h(x \text{ bin}(\alpha^\delta\beta^{-\gamma} \bmod p)) = \gamma \\ 0, & sonst \end{cases}$$

Der Signaturraum ist also $Y := \mathbb{Z}_q \times \mathbb{Z}_q$.

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

4.3 Der Digital Signature Algorithm (DSA)

(Standard in USA seit 1994)

Hierbei wird das El Gamal-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 55. *$q = 101$, $p = 78q + 1 = 7879$, $g = 3$ $(ord_p(3) = p - 1)$*

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

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:

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

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

4.4 ECDSA (Elliptic Curve DSA)

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

Definition 56. *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.*

$X = \{0,1\}^*$, $Y = \mathbb{Z}_q^* \times \mathbb{Z}_q^*$. 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

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

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

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

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

Annahme: Wollen Dokument x mit SHA-1$(x) = 4$ 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 $k = (p, q, E, A, B)$:

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

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

4.5 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 58. *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

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

ist.

Zum Nachweis der Sicherheit des Signaturverfahrens nehmen wir an, dass $f : U \to V$ eine Bijektion ist und dass ein Algorithmus LAMPORT-FÄLSCHUNG(k) existiert, der bei Eingabe eines Verifikationsschlüssel 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 59. *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:

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

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 $k \in \mathcal{S}_v$, bei deren Wahl LAMPORT-URBILD(v) Erfolg hat. 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 $T_{v_{i,b}}$ 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 zu

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

□

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 Signierschlüssel (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.

4.6 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 $f_{\hat{k}}(f_k(x)) = x$ für alle $x \in \{0,1\}^n$.

Weiter sei $G : \{0,1\}^* \to \{0,1\}^n$ eine Zufallsfunktion, d.h. die ZVn $X_x = G(x)$ 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 für 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 $y \in \{0,1\}^n$ 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$(k) = (x, y) \wedge f_k(y) = z_0$ **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 60. *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, so nimmt $G(x)$ den Wert $f_k(y)$ mit Wahrscheinlichkeit 2^{-n} an, d.h.

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

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

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

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

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

□

Falls sich also f_k nur mit einer sehr kleinen Wk ε' effizient invertieren lässt, so gelingt einem ähnlich effizienten Gegner, der nicht mehr als q Hashwertberechnungen durchführt im ZOM höchstens mit Wk $q\varepsilon' + 2^{-n}$ 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.

4.7 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.

Beispiel 61. *Eine Softwarefirma möchte sicherstellen, dass nur rechtmässige Käufer (keine SW-Piraten) ihre Signatur, die u.a. Virusfreiheit garantiert, verifizieren können.*

Lösung: Zur Verifikation wird die Kooperation des Unterzeichners Alice benötigt.

Wie kann man verhindern, dass sich Alice absichtlich unkooperativ verhält, nur damit eine von ihr geleistete Unterschrift als falsch eingestuft wird?

Lösung: Es gibt ein *Ableugnungsprotokoll (disavowal protocol)* mit dem Alice gefälschte Unterschriften als solche entlarven kann. Scheitert auch dieses Protokoll so liegt der Verdacht nahe, dass die fragliche Unterschrift doch von Alice stammt.

Das Signaturverfahren von Chaum und van Antwerpen

Bei diesem Signaturverfahren wird eine Primzahl $p = 2q + 1$ benutzt, wobei auch q prim ist, so dass das Diskrete Logarithmus Problem in $\mathbb{Z}_p^*$ hart ist. Sei $\alpha \in \mathbb{Z}_p^*$ ein Element der Ordnung q und sei $G = \{\alpha^a | a \in \mathbb{Z}_q\}$, die von α in $\mathbb{Z}_p^*$ erzeugte Untergruppe.

Der Dokumenten- und Signaturraum ist $X = Y = G$. Der Signierschlüssel hat die Form $\hat{k} = (p, \alpha, a)$, $a \in \mathbb{Z}_q^*$ und der zugehörige Verifikationsschlüssel ist $k = (p, \alpha, \beta)$ mit $\beta = \alpha^a \bmod p$. Der Signieralgorithmus berechnet $sig(\hat{k}, x) = x^a \bmod p$.

Will Bob eine von Alice geleistete Unterschrift $y \in G$ für ein Dokument $x \in G$ verifizieren, so führt er zusammen mit Alice folgendes Protokoll aus.

Verifikationsprotokoll:

1. Bob wählt zufällig $e_1, e_2 \in \mathbb{Z}_q$ und und sendet $c = y^{e_1}\beta^{e_2} \bmod p$ an Alice.
2. Alice sendet $d = c^{a^{-1} \bmod q} \bmod p$ zurück an Bob.
3. Bob akzeptiert y als echt, falls $d \equiv_p x^{e_1}\alpha^{e_2}$ ist.

Bemerkung 62. *Die Wahl von p der Form $p = 2q+1$ mit q prim dient folgenden Zielen:*

- *$\|G\|$ ist prim (erlaubt die Berechnung von $a^{-1} \bmod \|G\|$).*
- *$\|G\|$ ist für vorgegebenes p möglichst groß.*

Es ist leicht zu sehen, dass Bob eine echte Signatur y akzeptiert, falls Alice kooperiert. Wegen

$$\beta \equiv_p \alpha^a$$

folgt

$$\beta^{a^{-1}} \equiv_p \alpha^{a \cdot a^{-1}} \equiv_p \alpha$$

und wegen

$$y \equiv_p x^a$$

folgt

$$y^{a^{-1}} \equiv_p x^{a \cdot a^{-1}} \equiv_p x.$$

Somit ist

$$d = c^{a^{-1}} = (y^{e_1}\beta^{e_2})^{a^{-1}} = y^{a^{-1}e_1}\beta^{a^{-1}e_2} = x^{e_1}\alpha^{e_2}.$$

Beispiel 63. *Sei $p = 467 = 2 \cdot 233 + 1$ mit $q = 233$. Da $g = 2$ ein Erzeuger von $\mathbb{Z}_p^*$ ist, hat $\alpha = g^2 = 4$ die gewünschte Ordnung $q = \frac{p-1}{2}$. Da α die Untergruppe QR_p der quadratischen Reste erzeugt, ist $G = QR_p$. Wählen wir den Signierschlüssel $\hat{k} = (p, \alpha, a) = (467, 4, 101)$, so erhalten wir $k = (p, \alpha, \beta) = (467, 4, 449)$ als zugehörigen Verifikationsschlüssel. Die Signatur für $x = 119 \in G$ berechnet sich wie folgt:*

$sig(\hat{k}, x) = x^a \bmod p = 119^{101} \bmod 467 = 129 = y$

Verifikation von $y = 129$ *für* $x = 119$ *unter* k*:*

1. *Bob wählt* $e_1, e_2 \in \mathbb{Z}_q$ *(*$e_1 = 38, e_2 = 397 = 164$*) und sendet* $c = y^{e_1}\beta^{e_2} \bmod p = 129^{38}449^{164} \bmod 467 = 13$ *an Alice.*
2. *Alice sendet* $d = c^{a^{-1} \bmod q} \bmod p = 9$ *an Bob zurück.*
3. *Bob akzeptiert, da* $d = x^{e_1}\alpha^{e_2} = 119^{38}4^{164} \bmod 467 = 9$ *gilt.*

Behauptung 1. *Im Fall* $y \not\equiv_p x^a$ *akzeptiert Bob* y *mit Wahrscheinlichkeit* $1/q$*.*

Beweis. Da zu $y, \beta, c \in G$ und zu $e_1 \in \mathbb{Z}_q$ genau ein $e_2 \in \mathbb{Z}_q$ mit

$$c \equiv_p y^{e_1}\beta^{e_2} \tag{4.1}$$

existiert, führen je q Paare $(e_1, e_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$ auf dasselbe c. Aus der Sicht von Alice, die nur c kennt, sind diese q Paare alle gleichwahrscheinlich. Wir zeigen nun, dass für jedes $d \in G$ genau eines dieser q Paare die Kongruenz

$$d \equiv_p x^{e_1}\alpha^{e_2} \tag{4.2}$$

erfüllt, weshalb Bob mit Wahrscheinlichkeit $1/q$ akzeptiert.
Seien $i, j, k, l \in \mathbb{Z}_q$ die zu $c, d, x, y \in G$ gehörigen Exponenten, d.h. $c \equiv_p \alpha^i, \ldots, y \equiv_p \alpha^l$. Dann sind die Kongruenzen (4.1) und (4.2) äquivalent zu

$$\begin{array}{l} c \equiv_p y^{e_1}\beta^{e_2} \\ d \equiv_p x^{e_1}\alpha^{e_2} \end{array} \Leftrightarrow \begin{array}{l} \alpha^i \equiv_p \alpha^{le_1} \cdot \alpha^{ae_2} \\ \alpha^j \equiv_p \alpha^{ke_1} \cdot \alpha^{e_2} \end{array} \Leftrightarrow \begin{array}{l} i \equiv_q le_1 + ae_2 \\ j \equiv_q ke_1 + e_2 \end{array} \Leftrightarrow \underbrace{\begin{pmatrix} l & a \\ k & 1 \end{pmatrix}}_{A} \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} \equiv_q \begin{pmatrix} i \\ j \end{pmatrix}.$$

Wegen $\alpha^l \equiv_p y \not\equiv_p x^a \equiv_p \alpha^{ka}$ folgt $l \not\equiv_q ka$ und daher ist $\det A \not\equiv_q 0$. □

Möchte nun Alice Bob gegenüber nachweisen, dass eine Signatur y gefälscht ist, so führen beide folgendes Protokoll aus.

Ableugnungsprotokoll

1 **Bob wählt zufällig** $e_1, e_2 \in \mathbb{Z}_q$ **und sendet** $c = y^{e_1}\beta^{e_2} \bmod p$ **an Alice.**
2 **Alice sendet** $d = c^{a^{-1}} \bmod p$ **zurück.**
3 **Bob testet, ob** $d \not\equiv_p x^{e_1}\alpha^{e_2}$ **ist.**
4 **Bob wählt zufällig** $f_1, f_2 \in \mathbb{Z}_q$ **und sendet** $C = y^{f_1}\beta^{f_2} \bmod p$ **an Alice.**
5 **Alice sendet** $D = C^{a^{-1}} \bmod p$ **zurück.**
6 **Bob testet, ob** $D \not\equiv_p x^{f_1}\alpha^{f_2}$ **ist.**
7 **Bob erkennt** y **als gefälscht an, falls mindestens einer der Tests in Schritt 3) oder 6) erfolgreich war und** $(d\alpha^{-e_2})^{f_1} \equiv_p (D\alpha^{-f_2})^{e_1}$ **gilt.**

Bei den Schritten 1.-3. und 4.-6. handelt es sich jeweils um eine fehlgeschlagene Verifikation der Unterschrift y (sofern der Test von Bob in Zeile 3 bzw. 6 positiv ausfällt). In Schritt 7 führt Bob zusätzlich einen Konsistenztest aus, um sich davon zu überzeugen, dass Alice die Zahlen d und D gemäß dem Protokoll gewählt hat.

Beispiel 64. *Sei* $p = 467, q = 233, \alpha = 4, a = 101, \beta = 449$*. Wir nehmen an, dass das Dokument* $x = 286$ *mit der Alice zugeschriebenen Signatur* $y = 81$ *unterschrieben ist und Alice Bob davon überzeugen möchte, dass* y *gefälscht ist.*

1. *Bob wählt* $e_1 = 45, e_2 = 237$ *und sendet* $c = 305$ *an Alice.*
2. *Alice antwortet mit* $d = c^{a^{-1}} = 109$
3. *Bob verifiziert, dass* $286^{45}4^{237} \equiv_p 149 \not\equiv_p 109$ *gilt.*
4. *Bob wählt* $f_1 = 125, f_2 = 9$ *und sendet* $C = 72$ *an Alice.*
5. *Alice antwortet mit* $D = C^{a^{-1}} = 68$
6. *Bob verifiziert, dass* $286^{125}4^9 \equiv_p 25 \not\equiv_p 109$ *gilt.*
7. *Bon erkennt* y *also gefälscht an, da* $(109 \cdot 4^{-237})^{125} \equiv_p 188 \equiv_p (68 \cdot 4^{-9})^{45}$ *ist, also die Konsistenzbedingung erfüllt ist.*

Es bleibt zu zeigen, dass Alice zwar Bob mit hoher Wahrscheinlichkeit von der Falschheit einer Signatur $y \not\equiv_p x^a$ überzeugen kann, es ihr aber nicht gelingt, Bob von der Falschheit einer echten Signatur $y \equiv_p x^a$ zu überzeugen.

Behauptung 2. *Im Fall* $y \not\equiv_p x^a$ *erkennt Bob* y *mit Wahrscheinlichkeit* $1 - \frac{1}{q^2}$ *als gefälscht an, falls sich beide an das Ableugnungsprotokoll halten.*

Beweis. Nach vorigem Satz beträgt die Wahrscheinlichkeit, dass beide Tests in Schritt 3. und 6. fehlschlagen genau $\frac{1}{q^2}$. Wegen

$$d \equiv_p c^{a^{-1}}, c \equiv_p y^{e_1}\beta^{e_2}, \beta \equiv_p \alpha^a$$

folgt

$$\begin{aligned}(d\alpha^{-e_2})^{f_1} &\equiv_p ((y^{e_1}\beta^{e_2})^{a^{-1}}\alpha^{-e_2})^{f_1} \\ &\equiv_p y^{e_1a^{-1}f_1}\beta^{e_2a^{-1}f_1}\alpha^{-e_2f_1} \\ &\equiv_p y^{e_1a^{-1}f_1}\alpha^{e_2f_1}\alpha^{-e_2f_1} \\ &\equiv_p y^{e_1a^{-1}f_1}\end{aligned}$$

Analog ergibt sich aus

$$D \equiv_p C^{a^{-1}}, C \equiv_p y^{f_1}\beta^{f_2}, \beta \equiv_p \alpha^a$$

$$\begin{aligned}(D\alpha^{-f_2})^{e_1} &\equiv_p ((y^{f_1}\beta^{f_2})^{a^{-1}}\alpha^{-f_2})^{e_1} \\ &\equiv_p y^{f_1a^{-1}e_1} \\ &\equiv_p (d\alpha^{-e_2})^{f_1}\end{aligned}$$

d.h. die Konsistenzbedingung wird mit Wahrscheinlichkeit 1 erfüllt. □

Behauptung 3. *Im Fall* $y \equiv_p x^a$ *erkennt Bob* y *mit Wahrscheinlichkeit* $\leq \frac{1}{q}$ *als gefälscht an, auch wenn sich Alice nicht an das Ableugnungsprotokoll hält.*

Beweis. Bob erkennt y nur dann als gefälscht an, wenn

$$(d \not\equiv_p x^{e_1}\alpha^{e_2} \text{ oder } D \not\equiv_p x^{f_1}\alpha^{f_2}) \text{ und } (d\alpha^{-e_2})^{f_1} \equiv_p (D\alpha^{-f_2})^{e_1}$$

gilt. Da die beiden Fälle $d \not\equiv_p x^{e_1}\alpha^{e_2}$ und $D \not\equiv_p x^{f_1}\alpha^{f_2}$) symmetrisch sind, reicht es einen davon zu betrachten.

Wir nehmen also an, dass Alice eine Zahl d an Bob sendet mit $d \not\equiv_p x^{e_1}\alpha^{e_2}$. Nachdem Alice die Zahl C in Zeile 4 von Bob erhalten hat, weiß sie nur, dass das von Bob gewählte Paar

(f_1, f_2) die Kongruenz $C \equiv_p y^{f_1}\beta^{f_2}$ erfüllt. Wie wir bereits im Beweis zu Behauptung 1 gesehen haben, trifft dies auf genau q Paare zu. Wir zeigen nun, dass für jedes $D \in G$ genau eines dieser q Paare die Konsistenzbedingung

$$(d\alpha^{-e_2})^{f_1} \equiv_p (D\alpha^{-f_2})^{e_1}$$

erfüllt. Dies beweist, dass Bob y mit Wahrscheinlichkeit höchstens $1/q$ als gefälscht akzeptiert.

Sei $u = d\alpha^{-e2}$ und seien $i, j, k, l \in \mathbb{Z}_q$ die zu C, D, x, u gehörigen Exponenten, d.h. $C \equiv_p \alpha^i, \ldots, u \equiv_p \alpha^l$. Dann gilt

$$\begin{array}{c} C \equiv_p y^{f_1}\beta^{f_2} \\ (d\alpha^{-e_2})^{f_1} \equiv_p (D\alpha^{-f_2})^{e_1} \end{array} \Leftrightarrow \begin{array}{r} i \equiv_q kaf_1 + af_2 \\ lf_1 \equiv_q je_1 - e_1f_2 \end{array} \Leftrightarrow \underbrace{\begin{pmatrix} ka & a \\ l & e_1 \end{pmatrix}}_{A} \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} \equiv_q \begin{pmatrix} i \\ je_1 \end{pmatrix}.$$

Wegen $d \not\equiv_p x^{e_1}\alpha^{e_2}$ folgt $l \not\equiv_q e_1 k$ und daher ist $\det A = kae_1 - al = a(ke_1 - l) \not\equiv_q 0$. □