Vorlesungsskript

Kryptologie

Sommersemester 2018

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

19. Juli 2018

Inhaltsverzeichnis

1 Kryptografische Hashverfahren **1**
1.1 Einführung . . . 1
1.2 Schlüssellose Hashfunktionen (MDCs) . . . 3
1.2.1 Vergleich von Sicherheitsanforderungen . . . 4
1.2.2 Das Zufallsorakelmodell (ZOM) . . . 5
1.2.3 Iterierte Hashfunktionen . . . 8
1.2.4 Die Merkle-Damgaard-Konstruktion . . . 9
1.2.5 Die MD4-Hashfunktion . . . 10
1.2.6 Die MD5-Hashfunktion . . . 11
1.2.7 Die SHA-1-Hashfunktion . . . 12
1.2.8 Die SHA-2-Familie . . . 13
1.2.9 Kryptoanalyse von Hashfunktionen . . . 14
1.2.10 Die Sponge-Konstruktion . . . 15
1.2.11 SHA-3 . . . 17
1.3 Nachrichten-Authentikationscodes (MACs) . . . 18
1.3.1 Angriffe gegen symmetrische Hashfunktionen . . . 19
1.3.2 Informationstheoretische Sicherheit von MACs . . . 19
1.3.3 CBC-MACs . . . 28
1.3.4 Kombination einer Hashfunktion mit einem MAC (HMAC) . . . 29

2 Elliptische Kurven **31**
2.1 Elliptische Kurven über den reellen Zahlen . . . 31
2.2 Elliptische Kurven über endlichen Körpern . . . 33

3 Digitale Signaturverfahren **36**
3.1 Das RSA-Signaturverfahren . . . 37
3.2 Das ElGamal-Signaturverfahren . . . 38
3.3 Das Schnorr-Signaturverfahren . . . 40
3.4 Der Digital Signature Algorithm (DSA) . . . 41
3.5 ECDSA (Elliptic Curve DSA) . . . 42
3.6 One-time Signatur (Lamport 1979) . . . 43
3.7 Full Domain Hash (FDH) Signaturen . . . 46
3.8 Verbindliche Signaturen (undeniable signatures) . . . 49
3.9 Fail-Stop-Signaturen . . . 52

4 Pseudozufallszahlen-Generatoren **57**
4.1 Kryptografische Sicherheit von Pseudozufallsgeneratoren . . . 57
4.2 Quadratische Reste . . . 60
4.3 Der BBS-Generator . . . 63
4.4 Quadratische Pseudoreste . . . 64
4.5 Sicherheit des BBS-Generators . . . 64

1 Kryptografische Hashverfahren

1.1 Einführung

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

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

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

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

Typische Schutzziele, die sich mittels Hashfunktionen realisieren lassen, sind die Nachrichten- und Teilnehmerauthentikation.

- „Nachrichtenauthentikation“ (message authentication)

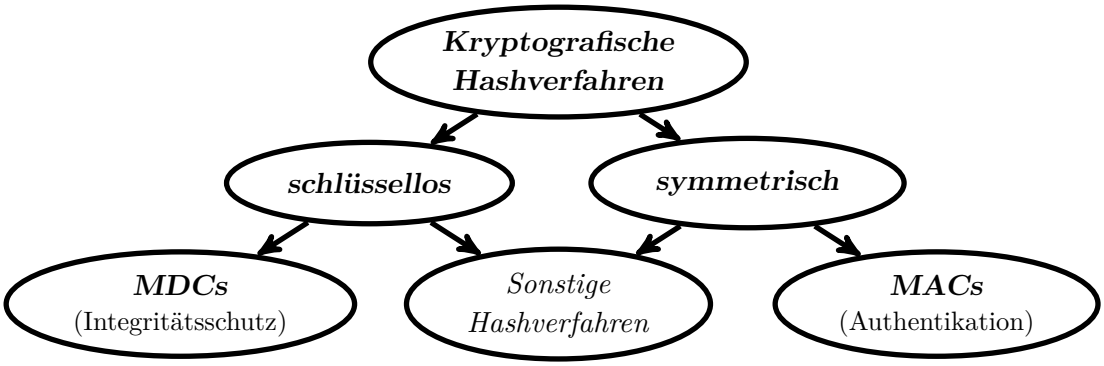


Abbildung 1.1: Eine grobe Einteilung von kryptografischen Hashverfahren.

  - Wie lässt sich sicherstellen, dass eine Nachricht (oder eine Datei) während einer (räumlichen oder auch zeitlichen) Übertragung nicht verändert wurde?
  - Wie lässt sich der Urheber (oder Absender) einer Nachricht zweifelsfrei feststellen?
- „Teilnehmerauthentikation“ (entity authentication, identification)
  - Wie kann sich eine Person (oder ein Gerät) anderen gegenüber zweifelsfrei ausweisen?

Klassifikation von Hashverfahren

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

Kryptografische Hashfunktionen, bei deren Berechnung keine Schlüssel benutzt werden, dienen vornehmlich der Erkennung von unbefugt vorgenommenen Manipulationen an Dateien oder Nachrichten. Daher werden sie auch als ***MDC*** bezeichnet (***M****anipulation* ***D****etection* ***C****ode* [englisch] = Code zur Erkennung von Manipulationen). Zuweilen wird das Kürzel *MDC* auch als eine Abkürzung für ***M****odification* ***D****etection* ***C****ode* verwendet. Seltener ist dagegen die Bezeichnung ***MIC*** (***m****essage* ***i****ntegrity* ***c****odes*). Abbildung 1.2 zeigt eine typische Anwendung von MDCs.

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

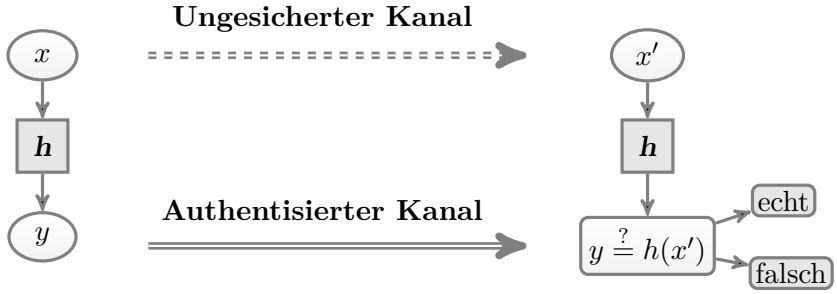


Abbildung 1.2: Einsatz eines MDC h zur Überprüfung der Integrität eines Datensatzes x.

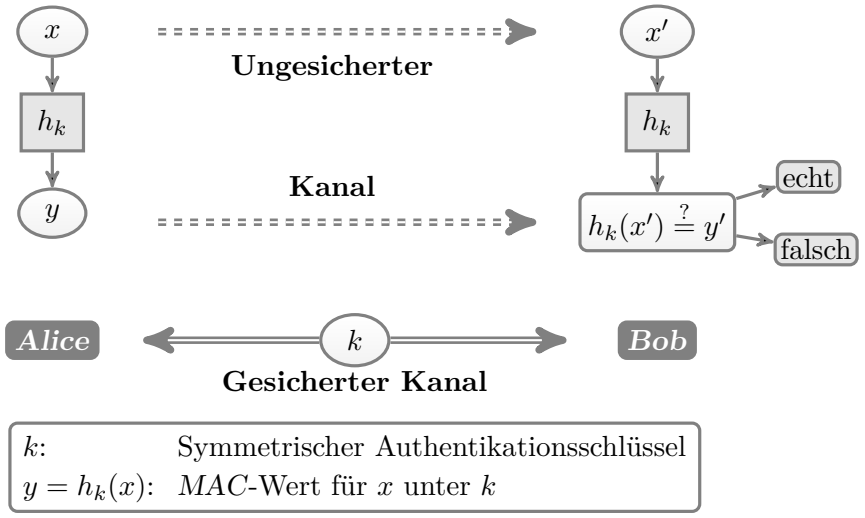


Abbildung 1.3: Verwendung eines MAC zur Nachrichtenauthentikation.

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

Kryptografische Hashverfahren mit symmetrischen Schlüsseln finden hauptsächlich bei der Authentifizierung von Nachrichten Verwendung. Diese werden daher auch als ***MAC*** (***m**essage **a**uthentication **c**ode* [englisch] = Code zur Nachrichtenauthentifizierung) oder als **Authentikationscode** bezeichnet. Daneben gibt es auch Hashverfahren mit asymmetrischen Schlüsseln. Diese werden jedoch der Rubrik der Signaturverfahren zugeordnet, da mit ihnen ausschließlich digitale Signaturen gebildet werden. Abbildung 1.3 zeigt, wie sich Nachrichten mit einem MAC authentisieren lassen. Man beachte, dass nun auch der Hashwert über den unsicheren Kanal gesendet wird.

Möchte Alice eine Nachricht x an Bob übermitteln, so berechnet sie den zugehörigen *MAC*-Wert $y = h_k(x)$ und fügt diesen der Nachricht x hinzu. Bob überprüft die Echtheit der empfangenen Nachricht (x', y'), indem er seinerseits den zu x' gehörigen Hashwert $h_k(x')$ berechnet und das Ergebnis mit y' vergleicht. Der geheime Authentikationsschlüssel k muss hierbei genau wie bei einem symmetrischen Kryptosystem über einen gesicherten Kanal vereinbart werden.

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

1.2 Schlüssellose Hashfunktionen (MDCs)

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

Sei $h\colon X \to Y$ eine Hashfunktion. Ein Paar $(x, y) \in X \times Y$ heißt **gültig** für h, falls

$h(x) = y$ ist. Ein Paar (x, x') mit $h(x) = h(x')$ heißt **Kollisionspaar** für h. Die Anzahl $\|Y\|$ der Hashwerte bezeichnen wir mit m. Ist auch der Textraum X endlich, $\|X\| = n$, so heißt h eine (n, m)-**Hashfunktion**. In diesem Fall verlangen wir meist, dass $n \geq 2m$ ist, und wir nennen h dann eine **Kompressionsfunktion** (*compression function*).

Da h öffentlich bekannt ist, ist es sehr einfach, für einen vorgegebenen Text x ein gültiges Paar (x, y) zu erzeugen. Für bestimmte kryptografische Anwendungen ist es wichtig, dass dies nicht möglich ist, falls der Hashwert y vorgegeben wird.

Problem P1: Bestimmung eines Urbilds

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

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

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

Problem P2: Bestimmung eines zweiten Urbilds

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

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

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

Problem P3: Bestimmung einer Kollision

Gegeben: Eine Hashfkt. $h\colon X \to Y$.

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

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

Obwohl die schwache Kollisionsresistenz eine gewisse Ähnlichkeit mit der Einweg-Eigenschaft aufweist, sind die beiden Eigenschaften im allgemeinen unvergleichbar. So muss eine schwach kollisionsresistente Funktion nicht notwendigerweise eine Einwegfunktion sein, da die Bestimmung eines Urbildes gerade für diejenigen Funktionswerte einfach sein kann, die nur ein einziges Urbild besitzen. Umgekehrt impliziert die Einweg-Eigenschaft auch nicht die schwache Kollisionsresistenz, da die Kenntnis eines Urbildes das Auffinden weiterer Urbilder sehr stark erleichtern kann.

1.2.1 Vergleich von Sicherheitsanforderungen

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

Satz 1. *Sei $h\colon X \to Y$ eine (n, m)-Hashfunktion. Dann ist das Problem P3, ein Kollisionspaar für h zu bestimmen, auf das Problem P2, ein zweites Urbild zu bestimmen, reduzierbar. Folglich sind stark kollisionsresistente Hashfunktionen auch schwach kollisionsresistent.*

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

Abbildung 1.4: Reduktion des Kollisionsroblems auf das Problem, ein zweites Urbild zu bestimmen

Beweis. Sei A ein Las-Vegas Algorithmus, der für ein zufällig aus X gewähltes x mit Erfolgswahrscheinlichkeit ε ein zweites Urbild x' für h liefert und andernfalls ? ausgibt. Dann ist klar, dass der in Abbildung 1.4 dargestellte Las-Vegas Algorithmus mit Wahrscheinlichkeit ε ein Kollisionspaar findet. □

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

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

Beweis. Sei A ein Invertierungsalgorithmus für h, d.h. A berechnet für jeden Hashwert y in $W(h) = \{h(x) \mid x \in X\}$ ein Urbild x mit $h(x) = y$. Betrachte den in Abbildung 1.5 dargestellten Las-Vegas Algorithmus B.
Sei $\mathcal{C} = \{h^{-1}(y) \mid y \in Y\}$. Dann hat B eine Erfolgswahrscheinlichkeit von

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

□

1.2.2 Das Zufallsorakelmodell (ZOM)

Das ZOM dient dazu, den Aufwand verschiedener Angriffe auf eine Hashfunktion $h\colon X \to Y$ nach oben abzuschätzen. Sind X und Y vorgegeben, so können wir eine Hashfunktion $h\colon X \to Y$ dadurch „konstruieren“, dass wir für jedes $x \in X$ zufällig ein $y \in Y$ wählen und $h(x)$ auf y setzen. Äquivalent hierzu ist, für h eine zufällige Funktion aus der Klasse $F(X, Y)$ aller m^n Funktionen von X nach Y zu wählen. Dieses Verfahren ist auf Grund des hohen Aufwands zwar nicht mehr praktikabel, wenn $n = \|X\|$ eine bestimmte Größe übersteigt. Es liefert uns aber ein theoretisches Modell für eine Hashfunktion mit „idealen“ kryptografischen Eigenschaften. Offensichtlich besteht für den Angreifer 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).

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

Abbildung 1.5: Reduktion des Kollisionsproblems auf das Urbildproblem

Prozedur `FindPreimage`(h, y, q)

```
1 wähle eine beliebige Menge X_0 = {x_1, ..., x_q} ⊆ X
2 for each x_i ∈ X_0 do
3   if h(x_i) = y then return(x_i)
4 return(?)
```

Abbildung 1.6: Bestimmung eines Urbilds für einen Hashwert

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

Proposition 3. *Sei $X_0 = \{x_1, \ldots, x_k\}$ eine beliebige Menge von k verschiedenen Texten aus X und seien $y_1, \ldots, y_k \in Y$. Dann gilt für eine zufällig aus $F(X, Y)$ gewählte Funktion h und für jedes Paar $(x, y) \in (X - X_0) \times Y$,*

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

Um eine obere Komplexitätsschranke für das Urbildproblem im ZOM zu erhalten, betrachten wir den in Abbildung 1.6 dargestellten Algorithmus. Hier (und bei den beiden folgenden Algorithmen) gibt der Parameter q die Anzahl der Hashwertberechnungen (also die Anzahl der gestellten Orakelfragen an das Zufallsorakel h) an. Der Zeitaufwand der Algorithmen ist dabei proportional zu q.

Satz 4. FindPreimage(h, y, q) *gibt im ZOM mit Wahrscheinlichkeit $\varepsilon = 1 - (1 - 1/m)^q$ ein Urbild von y aus (unabhängig von der Wahl der Menge X_0).*

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

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

□

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

Prozedur `FindSecondPreimage`(h, x, q)

```
1 y := h(x)
2 wähle eine beliebige Menge X_0 = {x_1, ..., x_{q-1}} ⊆ X − {x}
3 for each x_i ∈ X_0 do
4   if h(x_i) = y then return(x_i)
5 return(?)
```

Abbildung 1.7: Bestimmung eines 2. Urbilds für einen Hashwert

Prozedur `Collision`(h, q)

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

Abbildung 1.8: Bestimmung eines Kollisionspaares

Satz 5. FindSecondPreimage(h, x, q) *gibt im ZOM mit Wahrscheinlichkeit* $\varepsilon = 1 - (1 - 1/m)^{q-1}$ *ein zweites Urbild* $x_0 \neq x$ *von* $y = h(x)$ *aus.*

Ist q vergleichsweise klein, so ist bei beiden bisher betrachteten Angriffen $\varepsilon \approx q/m$. Um also auf eine Erfolgswahrscheinlichkeit von $1/2$ zu kommen, ist $q \approx m/2$ zu wählen.

Geht es lediglich darum, *irgendein* Kollisionspaar (x, x') aufzuspüren, so bietet sich ein sogenannter **Geburtstagsangriff** an. Dieser ist deutlich zeiteffizienter zu realisieren. Wie der Name schon andeutet, basiert dieser Angriff auf dem sogenannten Geburtstagsparadoxon, welches in seiner einfachsten Form folgendes besagt.

Geburtstagsparadoxon: Bereits in einer Schulklasse mit 23 Schulkindern haben mit einer Wahrscheinlichkeit größer $1/2$ mindestens zwei Kinder am gleichen Tag Geburtstag.*

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

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

mindestens eine Kugel mehr als einmal gezogen wird. Für $m = 365$ und $q = 23$ ergibt dies einen Wert von ungefähr 0,507.

Zur Kollisionsbestimmung verwenden wir den in Abbildung 1.8 dargestellten Algorithmus. Bei einer naiven Implementierung würde zwar der Zeitaufwand für die Auswertung der if-Bedingung quadratisch von q abhängen. Trägt man aber jeden Text x unter dem Suchwort $h(x)$ in eine (herkömmliche) Hashtabelle der Größe q ein, so wird der Zeitaufwand für die Bearbeitung jedes einzelnen Textes x im wesentlichen durch die Berechnung von $h(x)$ bestimmt.

Satz 6. Collision(h, q) *gibt im ZOM mit Erfolgswahrscheinlichkeit*

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

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

Beweis. Sei $X_0 = \{x_1, \ldots, x_q\}$. 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

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

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

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

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

Somit erhalten wir die Abschätzung

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

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

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

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

1.2.3 Iterierte Hashfunktionen

In diesem Abschnitt beschäftigen wir uns mit der Frage, wie sich aus einer kollisionsresistenten Kompressionsfunktion

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

eine kollisionsresistente Hashfunktion

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

konstruieren lässt. Hierzu betrachten wir folgende kanonische Konstruktionsmethode.

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

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

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

Processing: Sei $IV \in \{0,1\}^m$ ein öffentlich bekannter Initialisierungsvektor und sei $y(x) = y_1 \cdots y_r$ mit $|y_i| = t$ für $i = 1, \ldots, r$. Berechne eine Folge $z_0, \ldots, z_r$ von Strings $z_i \in \{0,1\}^m$ wie folgt:

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

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

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

Definition 7. *Eine Funktion $y\colon \{0,1\}^* \to \{0,1\}^*$ heißt **suffixfrei**, falls es keine Strings $x \neq \tilde{x}$ und z in $\{0,1\}^*$ mit $y(\tilde{x}) = zy(x)$ gibt (d.h. kein Funktionswert $y(x)$ ist Suffix eines Funktionswertes $y(\tilde{x})$ an einer Stelle $\tilde{x} \neq x$).*

Man beachte, dass jede suffixfreie Funktion insbesondere injektiv ist.

Satz 8. *Falls die Preprocessing-Funktion y suffixfrei und die Ausgabetransformation g injektiv ist, so ist mit h auch $\hat{h}$ kollisionsresistent.*

Beweis. Angenommen, es gelingt, ein Kollisionspaar $x, \tilde{x}$ für $\hat{h}$ mit $\hat{h}(x) = \hat{h}(\tilde{x})$ zu finden. Sei

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

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

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

ein Kollisionspaar für h. □

1.2.4 Die Merkle-Damgaard-Konstruktion

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

Für $x = \varepsilon$ sei $y(x) = 0^t$ und für $x \in \{0,1\}^n$ mit $n > 0$ sei $k = \lceil \frac{n}{t-1} \rceil$ und $x = x_1 x_2 \dots x_{k-1} x_k$ mit $|x_1| = |x_2| = \dots = |x_{k-1}| = t - 1$ sowie $|x_k| = t - 1 - d$, wobei $0 \leq d < t - 1$. Im Fall $k = 1$ ist dann $y(x) = 0x0^d 1 bin_{t-1}(d)$ und für $k > 1$ ist $y(x) = y_1 \cdots y_{k+1}$, wobei

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

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

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

Beweis. Seien $x \neq \tilde{x}$ zwei Texte mit $|x| \leq |\tilde{x}|$. Wir müssen zeigen, dass $y(x) = y_1 y_2 \dots y_{k+1}$ kein Suffix von $y(\tilde{x}) = \tilde{y}_1 \tilde{y}_2 \dots \tilde{y}_{l+1}$ ist. Im Fall $x = \varepsilon$ ist dies klar. Für $x \neq \varepsilon$ machen wir folgende Fallunterscheidung.

1. Fall: $|x| \not\equiv_{t-1} |\tilde{x}|$. Dann folgt $d \neq \tilde{d}$ und somit $y_{k+1} \neq \tilde{y}_{l+1}$.

2. Fall: $|x| = |\tilde{x}|$. In diesem Fall ist $k = l$. Wegen $x \neq \tilde{x}$ existiert ein Index $i \in \{1, \dots, k\}$ mit $x_i \neq \tilde{x}_i$. Dies impliziert $y_i \neq \tilde{y}_i$, also ist $y(x)$ kein Suffix von $y(\tilde{x})$.

3. Fall: $|x| \neq |\tilde{x}|$ und $|x| \equiv_{t-1} |\tilde{x}|$. In diesem Fall ist $k < l$. Da $y(x)$ mit einer Null beginnt, aber das $(l - k + 1)$-te Bit von $y(\tilde{x})$ eine Eins ist, kann $y(x)$ kein Suffix von $y(\tilde{x})$ sein. □

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

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

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

1.2.5 Die MD4-Hashfunktion

Die MD4-Hashfunktion (***M**essage **D**igest*) wurde 1990 von Rivest vorgeschlagen. Die Bitlänge von MD4 beträgt $l = 128$ Bit. Bei einer Wortlänge von 32 Bit entspricht dies 4 Wörtern. Die im Folgenden vorgestellten Hashfunktionen benutzen u.a. folgende Operationen auf Wörtern.

Operatoren auf $\{0,1\}^{32}$	
$X \wedge Y$	bitweises „Und“ von X und Y
$X \vee Y$	bitweises „Oder“ von X und Y
$X \oplus Y$	bitweises „exklusives Oder“ von X und Y
$\neg X$	bitweises Komplement von X
$X + Y$	Ganzzahl-Addition modulo 2^{32}
$X \rightarrow s$	Rechtsshift um s Stellen
$X \hookleftarrow s$	zirkulärer Linksshift um s Stellen

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

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

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

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

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

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

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

MD4(x)

input $x \in \{0,1\}^*, |x| = n$
$y := x10^k$**bin**$_{64}(n)$, $k \in \{0, 1, \ldots, 511\}$ **mit** $n + 1 + k + 64 \equiv 0 \pmod{512}$
$(H_1, H_2, H_3, H_4) := (67452301, efcdab89, 98badcfe, 10325476)$
sei $y = M_1 \cdots M_r$, $r = (n + 1 + k + 64)/512$
for $i := 1$ **to** r **do**
  sei $M_i = X[0] \cdots X[15]$
  $(A, B, C, D) := (H_1, H_2, H_3, H_4)$
  for $j := 0$ **to** 47 **do**
    $(A, B, C, D) := (D, (A + f_j(B, C, D) + X[z_j] + y_j) \hookleftarrow s_j, B, C)$
  $(H_1, H_2, H_3, H_4) := (H_1 + A, H_2 + B, H_3 + C, H_4 + D)$
output $H_1H_2H_3H_4$

1.2.6 Die MD5-Hashfunktion

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

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

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

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

MD5(x)

1 **input** $x \in \{0,1\}^*, |x| = n$
2 $y := x10^k$ **bin**$_{64}(n)$, $k \in \{0, 1, \ldots, 511\}$ mit $n + 1 + k + 64 \equiv 0 \pmod{512}$
3 $(H_1, H_2, H_3, H_4) := (67452301, efcdab89, 98badcfe, 10325476)$
4 sei $y = M_1 \cdots M_r$, $r = (n + 1 + k + 64)/512$
5 **for** $i := 1$ **to** r **do**
6 sei $M_i = X[0] \cdots X[15]$
7 $(A, B, C, D) := (H_1, H_2, H_3, H_4)$
8 **for** $j := 0$ **to** 63 **do**
9 $(A, B, C, D) := (D, B + (A + f_j(B, C, D) + X[z_j] + y_j) \hookleftarrow s_j, B, C)$
10 $(H_1, H_2, H_3, H_4) := (H_1 + A, H_2 + B, H_3 + C, H_4 + D)$
11 **output** $H_1H_2H_3H_4$

1.2.7 Die SHA-1-Hashfunktion

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

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

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

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

SHA-1(x)

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

1.2.8 Die SHA-2-Familie

Im Jahr 2001 veröffentlichte die US-Behörde NIST drei weitere Hashfunktionen der SHA-Familie: SHA-256, SHA-384, and SHA-512. Diese Funktionen werden auch als SHA-2 Hashfunktionen bezeichnet. In 2004 kam noch SHA-224 als vierte Variante hinzu.

SHA-256 und SHA-512 haben denselben Aufbau, unterscheiden sich aber in erster Linie in der benutzten Wortlänge: 32 Bit bei SHA-256 und 64 Bit bei SHA-512. Zudem werden unterschiedliche Shift- und Summationskonstanten verwendet und auch die Rundenzahlen differieren. SHA-224 und SHA-384 sind reduzierte Varianten von SHA-256 und SHA-512. Der SHA-256-Algorithmus benutzt die folgenden Konstanten K_j, $j = 0, \ldots, 63$ (in Hexadezimaldarstellung).

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

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

SHA-256(x)

```
1  input x ∈ {0,1}*, |x| = n
2  y := x10^k bin_64(n), k ∈ {0,1,...,511} mit n+1+k+64 ≡ 0 (mod 512)
3  (H_0,H_1,H_2,H_3,H_4,H_5,H_6,H_7) := (6a09e667, bb67ae85, 3c6ef372, a54ff53a,
4    510e527f, 9b05688c, 1f83d9ab, 5be0cd19)
5  sei y = M_1 ··· M_r, r = (n+1+k+64)/512
6  for i := 1 to r do
7    sei M_i = X[0] ··· X[15]
8    for t := 16 to 63 do
9      s0 := (X[t-15] ↪ 7) ⊕ (X[t-15] ↪ 18) ⊕ (X[t-15] → 3)
10     s1 := (X[t-2] ↪ 17) ⊕ (X[t-2] ↪ 19) ⊕ (X[t-2] → 10)
11     X[t] := X[t-16] + s0 + X[t-7] + s1
12     (A,B,C,D,E,F,G,H) := (H_0,H_1,H_2,H_3,H_4,H_5,H_6,H_7)
13     for j := 0 to 63 do
14       s0 := (A ↪ 2) ⊕ (A ↪ 13) ⊕ (A ↪ 22)
15       maj := (A ∧ B) ⊕ (A ∧ C) ⊕ (B ∧ C)
16       t2 := s0 + maj
17       s1 := (E ↪ 6) ⊕ (E ↪ 11) ⊕ (E ↪ 25)
18       ch := (E ∧ F) ⊕ (¬E ∧ G)
19       t1 := H + s1 + ch + K_j + X[j]
20       (A,B,C,D,E,F,G,H) := (t1+t2, A, B, C, D+t1, E, F, G)
21     (H_0,H_1,H_2,H_3,H_4,H_5,H_6,H_7)
22       := (H_0+A, H_1+B, H_2+C, H_3+D, H_4+E, H_5+F, H_6+G, H_7+H)
23 output H_0H_1H_2H_3H_4H_5H_6H_7
```

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

1.2.9 Kryptoanalyse von Hashfunktionen

Bereits 1991 wurden von Den Boer und Bosselaers Schwächen im MD4 aufgedeckt. Im August 2004 erschien ein Bericht [1] mit einer Anleitung, wie sich Kollisionen für MD4 mittels "hand calculation" finden lassen.

In 1993, fanden den Boer und Bosselaers einen Weg, so genannte "Pseudo-Kollisionen" für die MD5 Kompressionsfunktion zu generieren. In 1996, fand Dobbertin ein Kollisionspaar für die MD5 Kompressionsfunktion.

Im August 2004 wurden schließlich Kollisionen für MD5 von Xiaoyun Wang, Dengguo Feng, Xuejia Lai und Hongbo Yu berechnet. Der benötigte Aufwand wurde mit ca. 1 Stunde auf einem IBM p690 Cluster abgeschätzt.

Im März 2005 veröffentlichten Arjen Lenstra, Xiaoyun Wang und Benne de Weger zwei X.509 Zertifikate mit unterschiedlichen Public-keys, die auf denselben MD5-Hashwert führten. Nur wenige Tage später beschrieb Vlastimil Klima eine Möglichkeit, Kollisionen für MD5 innerhalb weniger Stunden auf einem Notebook zu berechnen. Mittels der so genannten Tunneling-Methode wurde die Rechenzeit vom gleichen Autor im März 2006 auf eine Minute verkürzt.

Auf der CRYPTO 98 stellten Chabaud und Joux einen Angriff auf SHA-0 vor, der ein Kollisionspaar mit nur 2^{61} Hashwertberechnungen (anstelle von 2^{80} bei einem Geburtstagsangriff) aufspürt.

In 2004 fanden Biham und Chen Beinahe-Kollisionen für den SHA-0, bei denen sich die Hashwerte nur an 18 von den 160 Bitpositionen unterschieden. Zudem legten sie volle Kollisionen für den auf 62 Runden reduzierten SHA-0 Algorithmus vor.

Schließlich wurde im August 2004 die Berechnung einer Kollision für den vollen 80-Runden SHA-0 Algorithmus von Joux, Carribault, Lemuet und Jalby bekannt gegeben. Hierzu wurden lediglich 2^{51} Hashwerte berechnet, die ca. 80 000 Stunden CPU-Rechenzeit auf einem 2-Prozessor 256-Itanium Supercomputer benötigten.

Ebenfalls im August 2004 wurde von Wang, Feng, Lai und Yu auf der CRYPTO 2004 eine Angriffsmethode für MD5, SHA-0 und andere Hashfunktionen vorgestellt, mit der sich die Anzahl der Hashwertberechnungen auf 2^{40} senken lässt. Dies wurde im Februar 2005 von Xiaoyun Wang, Yiqun Lisa Yin und Hongbo Yu geringfügig auf 2^{39} Hashwertberechnungen verbessert.

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

Im Jahr 2005 veröffentlichten Rijmen und Oswald einen Angriff, der mit weniger als 2^{80} Hashwertberechnungen ein Kollisionspaar für den auf 53 Runden reduzierten `SHA-1` Algorithmus findet. Nur wenig später kündigten Xiaoyun Wang, Yiqun Lisa Yin und Hongbo Yu einen Angriff auf den vollen 80-Runden `SHA-1` mit 2^{69} Hashwertberechnungen an. Im August 2005 erfuhr der benötigte Aufwand von Xiaoyun Wang, Andrew Yao und Frances Yao auf der CRYPTO 2005 eine weitere Reduktion auf 2^{63} Berechnungen. In 2008 wurde von Stephane Manuel ein Kollisionsangriff mit einem geschätzten Aufwand von 2^{51} bis 2^{57} Berechnungen veröffentlicht.

Die besten bekannten Angriffe gegen SHA-2 brechen die von 64 auf 41 Runden reduzierte Variante von SHA-256 und die von 80 auf 46 Runden reduzierte Variante von SHA-512.

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

1.2.10 Die Sponge-Konstruktion

Die Konstruktionsidee hinter dem SHA-3-Gewinner Keccak wird von den Autoren als *Sponge* (Schwamm) bezeichnet. Auf der Basis dieser Entwurfsmethode lassen sich außer Hashfunktionen bspw. auch Pseudozufallsgeneratoren gewinnen. Der Aufbau eines Sponges ähnelt oberflächlich betrachtet der in 1.2.3 vorgestellten Konstruktion von iterierten Hashfunktionen, weist aber einige Unterschiede auf. So basiert ein Sponge statt auf einer Kompressionsfunktion h auf einer Permutation (oder allgemeiner Transformation) $f : \{0,1\}^b \to \{0,1\}^b$, die wie h iteriert angewendet wird. Dabei wird der aktuelle b-Bitblock in zwei Teilblöcke der Länge r und c unterteilt, die als äußerer bzw. innerer Zustand bezeichnet werden. Wie der Name schon sagt, verbleiben die Bits des inneren Zustands im Sponge, d.h. sie dienen nur zur Berechnung des nächsten Zustands und werden im Gegensatz zu den Bits des äußeren Zustands nicht unmittelbar für die Gewinnung der Ausgabe genutzt. Die Anzahl c der Bits des inneren Zustands wird als **Kapazität**

des Sponges bezeichnet und ist sein wichtigster Sicherheitsparameter. Die Anzahl r der Bits des äußeren Zustands heißt **Bitrate** , wobei $r + c = b$ gelten muss.

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

Definition 10. *Sei $r \geq 1$. Eine Funktion $y\colon \{0,1\}^* \to \bigcup_{k\geq 1}\{0,1\}^{kr}$ heißt **sponge-konforme Paddingfunktion** für Bitrate r, falls gilt:*

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

Es ist leicht zu sehen, dass die Funktion $\texttt{pad10*1}_r\colon \{0,1\}^* \to \{0,1\}^*$ definiert durch

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

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

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

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

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

$$\texttt{Sponge}_{f,y,r}(l, x) = z_1 \dots z_m z'_{m+1} \quad .$$

Für die Analyse definieren wir

$$\texttt{Absorb}_{f,y,r}(x) = s_k \textit{ und } \texttt{Squeeze}_{f,r}(l, s_k) = z_1 \dots z_m z'_{m+1} \quad .$$

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

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

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

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

Abbildung 1.9: Bestimmung eines inneren Kollisionspaares

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

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

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

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

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

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

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

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

1.2.11 SHA-3

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

Jede Funktion $f_w : \{0,1\}^{5\times5\times w} \to \{0,1\}^{5\times5\times w}$ bildet ein zweidimensionales Feld A aus w-Bit-Wörtern auf ein ebensolches Feld $f_w(A)$ ab. Dabei wird $(12 + \log_2 w)$-mal eine

Rundenfunktion $f'_w : \{0,1\}^{5\times5\times w} \times \{0,1\}^w \to \{0,1\}^{5\times5\times w}$ aufgerufen, die A und eine Rundenkonstante RC_i auf A' abbildet.

Es gilt

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

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

Für die Werte $l \in \{224, 256, 384, 512\}$ definiert der Standard FIPS 202:

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

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

1.3 Nachrichten-Authentikationscodes (MACs)

Definition 13. *Eine **Hashfamilie** $\mathcal{H} = (X, Y, K, H)$ wird durch folgende Komponenten beschrieben:*

- *X, eine endliche oder unendliche Menge von Texten,*
- *Y, endliche Menge aller möglichen **Hashwerte**, $\|Y\| \leq \|X\|$,*
- *K, endlicher **Schlüsselraum** (key space), wobei jeder Schlüssel $k \in K$ eine Hashfunktion $h_k \colon X \to Y$ in H spezifiziert, d.h. $H = \{h_k \mid k \in K\}$.*

Im folgenden werden wir die Größe $\|X\|$ des Textraumes mit n, die des Hashwertbereiches Y mit m und die des Schlüsselraumes K mit l bezeichnen. Wir nennen dann $\mathcal{H}$ auch eine **$(\mathbf{n}, \mathbf{m}, \mathbf{l})$-Hashfamilie** oder einen **$(\mathbf{n}, \mathbf{m}, \mathbf{l})$-MAC**.

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

Berechnungsresistenz: Auch wenn eine Reihe von unter einem Schlüssel k generierten Text-Hashwert-Paaren $(x_1, h_k(x_1)), \ldots, (x_n, h_k(x_n))$ 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 Angreifer nicht möglich, an Bob eine Nachricht x zu schicken, die Bob als von Alice stammend anerkennt.

Verwendung eines *MAC* zur Versiegelung von Software

Mithilfe einer berechnungsresistenten Hashfunktion kann der Integritätsschutz für mehrere Datensätze auf die Geheimhaltung eines Schlüssels k zurückgeführt werden.

Um die Datensätze $x_1, \ldots, x_n$ gegen unbefugt vorgenommene Veränderungen zu schützen, legt man sie zusammen mit ihren MAC-Werten $y_1 =$

$h_k(x_1), \ldots, y_n = h_k(x_n)$ 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 MAC-Werten zum Ziel. Das heißt, der Angreifer versucht, MAC-Werte $h_k(x)$ ohne Kenntnis des geheimen Schlüssels k zu berechnen. Entsprechend der Art des zur Verfügung stehenden Textmaterials lassen sich die Angriffe gegen einen MAC wie folgt klassifizieren.

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

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

Angriff bei bekanntem Text (*known-text attack*)
: Der Angreifer kennt für eine Reihe von Texten $x_1, \ldots, x_r$ (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 Angreifer kann die Texte x_i selbst wählen.

Angriff bei adaptiv wählbarem Text (*adaptive chosen-text attack*)
: Der Angreifer kann die Wahl des Textes x_i von den zuvor erhaltenen *MAC*-Werten $h_k(x_j)$, $j < i$, abhängig machen.

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

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 Angreifer bekannt ist. Wir nehmen o.B.d.A. an, dass $p(x) > 0$ und $p(k) > 0$ für alle $x \in X$ und $k \in K$ gilt.

Erfolgswahrscheinlichkeit für Impersonation

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

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

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

wobei $K(x,y) = \{k \in K \mid h_k(x) = y\}$ alle Schlüssel enthält, die x auf y abbilden. D.h. $p(x \mapsto y)$ ist die Wahrscheinlichkeit, dass Bob das Paar (x,y) als echt akzeptiert. Somit gibt $p(x \mapsto y)$ die Wahrscheinlichkeit an, mit der einem Angreifer bei Wahl des Paares (x,y) eine Impersonation gelingt, weshalb wir diese Wahrscheinlichkeit auch mit $\alpha(x,y)$ bezeichnen. Schließlich ist $\alpha(x) = \max\{\alpha(x,y) \mid y \in Y\}$ die Wahrscheinlichkeit, mit der einem Angreifer bei optimaler Strategie eine Impersonation mit dem Text x gelingt, und es gilt $\alpha = \max\{\alpha(x) \mid x \in X\}$.

Beispiel 14. *Sei $K = \{1,2,3\}$, $X = \{a,b,c,d\}$ und $Y = \{0,1\}$. Wir beschreiben H durch die zugehörige **Authentikationsmatrix**. Die Zeilen und Spalten dieser Matrix werden mit den Schlüsseln $k \in K$ und den Texten $x \in X$ indiziert und ihr Eintrag in Zeile k und Spalte x ist der Wert $h_k(x)$.*

		0,1	0,2	0,3	0,4
		a	b	c	d
0,25	1	0	0	0	1
0,30	2	1	1	0	1
0,45	3	0	1	1	0

Die umrahmten Zahlen geben die Wahrscheinlichkeiten $p(x)$ bzw. $p(k)$ an. Dann hat der Angreifer folgende Erfolgsaussichten $\alpha(x)$, falls er an Bob den Text x senden möchte.

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

Folglich ist $\alpha = 0{,}75$. ◁

Satz 15. *Für alle $x \in X$ ist $\alpha(x) \geq \frac{1}{m}$ und daher gilt $\alpha \geq \frac{1}{m}$.*

Beweis. Sei $x \in X$ beliebig. Dann gilt

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

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

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

□

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

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

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

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

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

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

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

□

Satz 18. *Für jeden MAC (X, Y, K, H) gilt:*

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

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

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

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

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

Wegen

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

und

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

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

Beispiel 19 (Fortsetzung von Beispiel 14)**.** *Es gilt*

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

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

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

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

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

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

Erfolgswahrscheinlichkeit für Substitution

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

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

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

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

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

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

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

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

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

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

Beispiel 20 (Fortsetzung von Beispiel 14)**.**

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

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

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

◁

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

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

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

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

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

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

Folglich ist

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

□

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

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

*Die zugehörige **Authentikationsmatrix** ist*

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

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

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

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

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

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

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

dann wäre auch

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

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

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

was im Widerspruch zur Voraussetzung des Satzes steht. Ist andererseits

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

muss wegen

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

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

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

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

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

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

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

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

$$p(x' \mapsto y') = \sum_{y} p(x \mapsto y)p(x' \mapsto y' \,|\, x \mapsto y) = 1/m$$

und daher

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 $p + 1$ (und somit der Quotient n/m von 1 auf $\frac{p+1}{p}$) vergrößern, ohne K und Y zu verändern (siehe Übungen). Wie der nächste Satz zeigt, lässt sich eine stärkere Kompression mit dem Parameterwert $\lambda = 1$ jedoch nicht realisieren.

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

$$n \leq m + 1$$

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

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

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

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

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

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

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

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

$$Ax = y$$

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

Beweis. Siehe Übungen. □

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

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

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

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

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

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

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

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

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

Im Beweis des nächsten Satzes benötigen wir folgendes Lemma.

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

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

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

□

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

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

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

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

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

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

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

Mit obigem Lemma ergibt sich

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

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

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

Somit erhalten wir

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

□

1.3.3 CBC-MACs

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

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

Berechnung von $h_k(x)$:

---

1 $y := y(x) = y_1 \dots y_n$, $n \geq 1$, $y_i \in \{0,1\}^t$
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 MAC-Wertlänge beträgt also t Bit. Wird auf den Preprocessing-Schritt verzichtet, so lässt sich leicht ein Angriff mit 2 adaptiven Fragen ausführen. Kennt der Angreifer 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 nt für die Texte vorschreibt, wodurch die Anwendbarkeit des CBC-MACs allerdings einschränkt wird. Der folgende Geburtstagsangriff ist auch bei fester Textlänge möglich.

Geburtstagsangriff auf einen CBC-MAC

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

Wegen $x_1^i \neq x_1^j$ für $i \neq j$ sind auch die Texte $x^1, \dots, x^q$ paarweise verschieden. Seien $z_1^1, \dots, z_1^q$ die nach der ersten Iteration des CBC-MACs berechneten Kryptotexte $z_1^i = E_k(IV \oplus x_1^i)$. Da die Blöcke x_2^i zufällig gewählt werden, sind auch die Eingangsblöcke $z_1^i \oplus x_2^i$ für die 2. Iteration zufällig, d.h. es gilt

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

Da die Gleichheit der Eingangsblöcke $z_1^i \oplus x_2^i$ und $z_1^j \oplus x_2^j$ für die 2. Iteration mit der Gleichheit der Ausgangsblöcke z_n^i und z_n^j der n-ten Iteration und damit mit der Gleichheit der zugehörigen MAC-Werte z^i und z^j äquivalent ist, kann der Angreifer 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 Angreifer für einen beliebigen Bitblock $u \in \{0,1\}^t - \{0^t\}$ den MAC-Wert $\bar{z}_i = h_k(\bar{x}^i)$ für den Text $\bar{x}^i = x_1^i(x_2^i \oplus u)x_3 \cdots x_n$, welcher zugleich MAC-Wert des Textes $\bar{x}^j = x_1^j(x_2^j \oplus u)x_3 \cdots x_n$ ist, den er zuvor nicht erfragt hat.

Definition 35. *Sei $0 \leq \varepsilon \leq 1$ und sei $q \in \mathbb{N}$. Ein (ε, q)-Fälscher für einen MAC $\mathcal{H}$ 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 MAC-Werten der Texte $x_1, \ldots, x_{i-1}$ abhängen) und nicht-adaptiven Fragen. Zudem unterscheiden wir zwischen selektiven Fälschungen (d.h. der Angreifer kann den MAC-Wert für einen Text seiner Wahl generieren) und existientiellen Fälschungen (d.h. der Angreifer kann den MAC-Wert für irgendeinen Text $x \notin \{x_1, \ldots, x_q\}$ generieren, auf dessen Wahl er keinen Einfluss hat).

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

Eine Variante dieses Angriffs ist auch bei Verwendung einer Preprocessing-Funktion möglich. Meist wird hierzu die Funktion $y : x \mapsto y(x) = y_0 \ldots y_n$ mit $y_0 = bin_t(|x|)$ und $y_1 \ldots y_n = x0^{nt-|x|}$ verwendet, wobei $n = \lceil |x|/t \rceil$ ist. Der erste Block $y_0 = bin_t(|x|)$ kodiert also die Länge von x als Binärzahl, die mit führenden Nullen auf die Länge t erweitert wird, und der letzte Block wird ebenfalls mit Nullen auf die Länge t aufgefüllt.

1.3.4 Kombination einer Hashfunktion mit einem MAC (HMAC)

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

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

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

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

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

512 Bit Konstanten. Dann berechnet sich HMAC wie folgt:

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

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

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

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

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

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

$$\Pr[\underbrace{x \notin \{x_1, \ldots, x_q\}}_{B} \wedge \underbrace{g_{k_2}(f_{k_1}(x)) = z}_{C}] < \varepsilon_1 + \varepsilon_2$$

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

Behauptung 41. $\Pr[\underbrace{x \notin \{x_1, \ldots, x_q\}}_{B} \wedge \underbrace{f_{k_1}(x) \in \{f_{k_1}(x_1), \ldots, f_{k_1}(x_q)\}}_{D}] < \varepsilon_1$.

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

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

Da A' genau dann Erfolg hat, wenn das Ereignis $B \cap D$ eintritt, folgt Behauptung 88.

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

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

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

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

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

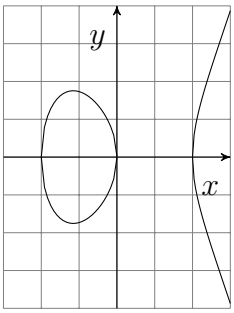
□

2 Elliptische Kurven

2.1 Elliptische Kurven über den reellen Zahlen

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

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



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

Auf den nicht-singulären Punkten von E lässt sich eine additive Gruppenoperation $+$ definieren. Die Idee dabei ist, dass die Addition von 3 beliebigen Punkten von E, die auf einer Geraden liegen, das neutrale Element $\mathcal{O}$ ergeben soll. Hierbei werden Tangentialpunkte doppelt und Wendepunkte dreifach gezählt und den parallel zur y-Achse verlaufenden Geraden wird zusätzlich noch der Punkt $\mathcal{O}$ hinzugerechnet (d.h. alle Geraden, die parallel zur y-Achse verlaufen, schneiden sich im Punkt $\mathcal{O}$ und es werden nur solche Geraden g betrachtet, auf denen bei dieser Zählweise 3 Punkte von E liegen).

Um nun die Summe $R = P + Q$ von zwei gegebenen Punkten $P = \{x_1, y_1\}$ und $Q = \{x_2, y_2\}$ zu berechnen, bestimmen wir zuerst die Gerade g, auf denen P und Q liegen, wobei g im Fall $P = Q$ die Tangente an E im Punkt P ist. Falls g parallel zur y-Achse verläuft, ist $x_1 = x_2$ und $y_1 = -y_2$ (also $Q = (x_1, -y_1)$). Da in diesem Fall zudem der Punkt $\mathcal{O}$ auf g liegt, erhalten wir die Gleichung $P + Q(+\mathcal{O}) = \mathcal{O}$ bzw. $-P = Q = (x_1, -y_1)$.

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

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

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

Für alle $(x, y) \in E \cap g$ gilt

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

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

p lässt sich in $\mathbb{C}$ vollständig in Linearfaktoren zerlegen,

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

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

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

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

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

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

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

Die Steigung λ von g erhalten wir durch implizites Differenzieren:

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

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

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

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

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

und

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

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

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

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

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

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

2.2 Elliptische Kurven über endlichen Körpern

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

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

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

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

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

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

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

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

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

Beispiel 48. *$p = 11$, 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 $\pm z^{\frac{p+1}{4}} \bmod p$ bestimmen.*

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

Da die Gruppe $(E, +, \mathcal{O})$ die Größe $\|E\| = 13$ hat und 13 prim ist, haben alle Elemente entweder die Ordnung 1 oder 13. Da nur das neutrale Element $\mathcal{O}$ die Ordnung 1 hat, haben alle anderen Elemente $P \in E - \{\mathcal{O}\}$ die Ordnung 13, sind also Erzeuger der Gruppe. Folglich ist $(E, +, \mathcal{O})$ zyklisch und somit isomorph zu $\mathbb{Z}_{13}$: $(E, +, \mathcal{O}) \cong (\mathbb{Z}_{13}, +, 0)$.
Berechnung von $2g = (2, 7) + (2, 7) = (5, 2)$:

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

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

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

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

◁

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

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

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

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

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

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

Kompakte Darstellung von Punkten auf E

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

Folgende Funktion liefert dann eine kompakte Darstellung.

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

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

Prozedur `PointDeCompress`(x, b)

```
1  z := p(x) mod p
2  y := z^((p+1)/4) mod p
3  if y^2 ≡_p z then
4     if y ≢_2 b then y := p − y
5     output(x, y)
6  else output(''error'')
```

Effiziente Berechnung von Vielfachen von Punkten auf E

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Prozedur `DoubleAddSub`$(P, c_s, \ldots, c_0)$

```
1  Q := O
2  for i := s downto 0 do
3     Q := 2Q + c_i P
4  output(Q)
```

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

3 Digitale Signaturverfahren

Handschriftliche Signaturen

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

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

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

Definition 57. *Ein digitales Signaturverfahren besteht aus:*

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

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

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

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

Klassifikation von Angriffen gegen Signaturverfahren

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

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

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

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

Erfolgskriterien für die Fälschung digitaler Signaturen

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

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

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

3.1 Das RSA-Signaturverfahren

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

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

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

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

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

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

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

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

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

Bei der Signaturerstellung benötigte Eigenschaften einer Hashfunktion h

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

3.2 Das ElGamal-Signaturverfahren

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

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

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

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

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

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

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

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

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

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

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

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

◁

Zur Sicherheit des ElGamal-Systems

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

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

1. *Die Zufallszahl z muss geheim gehalten werden.*

2. Zufallszahlen dürfen nicht mehrfach verwendet werden.

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

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

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

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

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

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

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

3.3 Das Schnorr-Signaturverfahren

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

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

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

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

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

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

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

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

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

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

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

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

und verifiziert die Gleichheit $h(xbin(2518)) = 96$. ◁

3.4 Der Digital Signature Algorithm (DSA)

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

1. δ als Lösung von $z\delta - a\gamma \equiv_{p-1} x$ (d.h. $\delta = (x + a\gamma)z^{-1}$) $\rightsquigarrow$ 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 hat der Signierschlüssel also die Form $\hat{k} = (p, q, \alpha, a)$, wobei $a \in \mathbb{Z}_q^*$ ist, und der zugehörige Verifikationsschlüssel ist $k = (p, q, \alpha, \beta)$ mit $\beta = \alpha^a \bmod p$. Zudem gilt $X = \mathbb{Z}_q$ und $Y = \mathbb{Z}_q \times \mathbb{Z}_q^*$.

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

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

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

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

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

Korrektheit: 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 63. *$q = 101$, $p = 78q + 1 = 7879$, $g = 3$ $(ord_p(3) = p - 1)$*

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

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

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

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

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

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

3.5 ECDSA (Elliptic Curve DSA)

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

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

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

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

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

wobei

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

Hierbei wird u als eine Zahl in $\{0,\ldots,p^n-1\}$ interpretiert. Falls $\gamma=0$ oder $\delta=0$ ist, muss eine neue Zufallszahl z gewählt und der Vorgang wiederholt werden.
Verifikation: $ver(k,x,\gamma,\delta)=1$, falls $u \bmod q=\gamma$ ist, wobei

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

Korrektheit der Verifikation beim ECDSA:

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

Beispiel 64. *Sei E über $\mathbb{Z}_{11}$ definiert durch $\gamma^2=x^3+x+6$. Wir wählen $A=(2,7)$, $m=7\rightarrow p=11,q=13,B=7A=(7,2)$.*
Um einen Text x mit dem Hashwert $h(x)=4$ unter Verwendung des Signierschlüssels $\hat{k}=(E,q,A,m)$ und der Zufallszahl $r=3$ signieren, berechnet Alice

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

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

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

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

3.6 One-time Signatur (Lamport 1979)

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

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

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

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

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

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

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

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

Beispiel 65. *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 $\ell = 3$.

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

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

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

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

◁

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

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

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

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

Prozedur `Lamport-Urbild`(v)

```
1 wähle zufällig ein Indexpaar (j,a) und setze v_{j,a} := v
2 for (i,b) ≠ (j,a) do
3   wähle zufällig u_{i,b} ∈_R U und setze v_{i,b} := f(u_{i,b})
4 k := (v_{1,0}...v_{ℓ,0} / v_{1,1}...v_{ℓ,1})
5 (x_1...x_ℓ, (u_1,...,u_ℓ)) =: Lamport-Fälschung(k)
6 if f(u_j) = v then output(u_j) else output(?)
```

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

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

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

Da zudem K (und damit auch (X, Y)) unabhängig von (J, A) und auch J und A unabhängig voneinander sind, ist A von (J, K, X, Y) und damit auch von X_J unabhängig. Sei p die Erfolgswk von Lamport-Urbild bei Eingabe V. Wegen

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

folgt nun

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

□

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

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

Für den Beweis betrachten wir folgenden probabilistischen Algorithmus

Prozedur `Lamport-Urbild'`(v)

```
1 wähle zufällig ein Indexpaar (j,a) und setze v_{j,a} := v
2 for (i,b) ≠ (j,a) do
3   wähle zufällig u_{i,b} ∈_R U und setze v_{i,b} := f(u_{i,b})
4 k := (v_{1,0}...v_{ℓ,0} / v_{1,1}...v_{ℓ,1})
5 simuliere Lamport-Fälschung'(k) und beantworte die Frage x' mit
    u_{1,x'_1}, ..., u_{ℓ,x'_ℓ} (falls x'_j = a ist, brich ab und gib ? aus); sei
    (x, y) = (x_1...x_ℓ, (u_1,...,u_ℓ)) die erzeugte Ausgabe
6 if f(u_j) = v then output(u_j) else output(?)
```

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

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

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

Prozedur `Lamport-Urbild`*

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

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

$$\begin{aligned} p* &\geq \Pr[ver(K, X, Y) = 1 \wedge X'_J \neq X_J = A] \\ &= \Pr[ver(K, X, Y) = 1] \underbrace{\Pr[X'_J \neq X_J = A \mid ver(K, X, Y) = 1]}_{\geq 1/2\ell} \geq \varepsilon/2\ell. \end{aligned}$$

□

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

3.7 Full Domain Hash (FDH) Signaturen

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

- $f_{\hat{k}}(f_k(u)) = u$ für alle $u \in U$.

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

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

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

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

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

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

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

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

Sicherheitsanalyse der FDH-Signatur im ZOM

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

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

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

Betrachte folgenden Invertierungsalgorithmus für f_k.

Prozedur `FDH-Invert`(k, v)

```
1 wähle zufällig j ∈_R {1, ..., q}
```

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

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

Beweis. Seien J, K, U, V, X, $X_1, \ldots, X_q$ Zufallsvariablen, die die Wahl von j, k, u, v, x, $x_1, \ldots, x_q$ beschreiben. Da die Eingabe v gleichverteilt ist, erhält `FDH-Fälschung` auf die Fragen $x_1, \ldots, x_q$ an G zufällig gewählte Strings $v_1, \ldots, v_q$ als Antwort, was dem ZOM entspricht. Daher liefert die Simulation von `FDH-Fälschung`(k) für einen zufällig generierten Schlüssel k mit Wahrscheinlichkeit ε eine Fälschung (x, y) mit $f_k(y) = G(x)$:

$$\Pr[f_K(Y) = G(X)] = \varepsilon.$$

Wir wollen zeigen, dass $\Pr[f_K(Y) = V] \geq \varepsilon/q$ ist. Da $x \in \{x_1, \ldots, x_q\}$ enthalten ist, existiert ein i mit $x = x_i$ und die Gleichheit $f_k(y) = G(x)$ impliziert $f_k(y) = G(x_i) = v_i$, was im Fall $i = j$ wiederum $f_k(y) = v_j = v$ impliziert:

$$ver(k, x, y) = 1 \wedge x_j = x \implies f_k(y) = v.$$

Daher folgt

$$\Pr[f_K(Y) = V] \geq \Pr[f_K(Y) = G(X) \wedge X_J = X].$$

Da zudem $j \in \{1, \ldots, q\}$ zufällig gewählt und die Fragen $x_1, \ldots, x_q$ unabhängig voneinander durch zufällige $v_1, \ldots, v_q$ beantwortet werden (nach Voraussetzung trifft dies auch auf $v_j = v$ zu), erhält `FDH-Fälschung` weder durch k noch durch die Antworten $v_1, \ldots, v_q$ irgendeine Information über j. Daher ist die Zufallsvariable J stochastisch unabhängig von K, $X_1, \ldots, X_q$, X sowie Y und somit auch von der Zufallsvariablen I, die den Index $i \in \{1, \ldots, q\}$ mit $x = x_i$ bestimmt. Daher folgt

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

□

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

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

Sei `FDH-Fälschung'` ein probabilistischer Algorithmus, der für einen zufällig generierten Verifikationsschlüssel k mit Wahrscheinlichkeit ε eine existentielle Fälschung (x, y) mit $f_k(y) = G(x)$ ausgibt und insgesamt für q Texte $x_1, \ldots, x_q$ den Wert $G(x_i)$ oder die Signatur $sig(\hat{k}, x_i)$ erfragt. Dabei können wir o.B.d.A. annehmen, dass `FDH-Fälschung'` zwar den G-Wert aber nicht die Signatur von x erfragt und vor jeder Signaturfrage den G-Wert des betreffenden Textes erfragt.

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

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

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

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

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

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

3.8 Verbindliche Signaturen (undeniable signatures)

In manchen Fällen ist es für den Unterzeichner eines Textes nicht wünschenswert, dass jeder dazu in der Lage ist, die Gültigkeit einer vorgelegten Signatur zu verifizieren.

Zum Beispiel könnte eine Softwarefirma (Alice) ihre Produkte mit einer Signatur versehen, die u.a. Virenfreiheit garantiert.

Problem: Neben den legalen Erwerbern der Software (Bob) können sich auch Kaufinteressenten auf dem Schwarzmarkt von der Gültigkeit einer Signatur (und damit von der Virenfreiheit des signierten Produkts) überzeugen.

Lösung: Die Gültigkeit einer Signatur lässt sich nur unter Mitwirkung von Alice verifizieren.

Neues Problem: Alice könnte versuchen, eine von ihr erzeugte gültige Signatur abzuleugnen, indem sie ihre Verifikation sabotiert.

Lösung: Es gibt zusätzlich ein *Ableugnungsprotokoll (disavowal protocol)*, mit dem Alice die Ungültigkeit von Signaturen nachweisen kann. Falls Alice die Gültigkeit einer Signatur bestreitet und sich dennoch weigert, die Ungültigkeit mithilfe des Ableugnungsprotokolls zu beweisen, kann man davon ausgehen, dass die Signatur gültig ist.

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 Text- 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 $\beta = \alpha^a \bmod p$.

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

$$sig(\hat{k}, x) = x^a \bmod p.$$

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

Verifikationsprotokoll:

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

Es ist leicht zu sehen, dass Bob eine gültige Signatur $y = x^a \bmod p$ mit Wk 1 als gültig akzeptiert, falls sich beide an das Verifikationsprotokoll halten:

$$x^e \alpha^f \equiv_p \underbrace{(x^{ae} \alpha^{af})}_{y^e \beta^f \equiv_p c}^{a^{-1} \bmod q} \equiv_p c^{a^{-1} \bmod q} \equiv_p d.$$

Beispiel 71. *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 $e, f \in \mathbb{Z}_q$ ($e = 38, f = 164$) und sendet $c = y^e \beta^f \bmod p = 129^{38} 449^{164} \bmod 467 = 13$ an Alice.*
2. *Alice sendet $d = c^{a^{-1} \bmod q} \bmod p = 9$ an Bob zurück.*
3. *Bob akzeptiert, da $d = x^e \alpha^f = 119^{38} 4^{164} \bmod 467 = 9$ ist.* ◁

Bemerkung 72. *Die Wahl von p der Form $p = 2q+1$ mit q prim dient folgenden Zielen:*

- *Die Ordnung q der Untergruppe G von $\mathbb{Z}_p^*$ ist prim (dies erlaubt die Berechnung von $a^{-1} \bmod q$ in Schritt 2 des Verifikationsprotokolls).*
- *G ist eine möglichst große Untergruppe von $\mathbb{Z}_p^*$ mit primer Ordnung.*

Behauptung 73. *Bob akzeptiert eine ungültige Signatur $y \not\equiv_p x^a$ nur mit Wahrscheinlichkeit $1/q$ (auch wenn sich Alice nicht an das Verifikationsprotokoll hält).*

Beweis. Alice steht in Zeile 2 des Verifikationsprotokolls vor der Aufgabe, eine Zahl $d \in G$ zu finden, so dass Bob in Zeile 3 akzeptiert. Das wäre für Alice problemlos möglich, wenn sie e und f kennen würde. Alice hat aber nur partielles Wissen über das Paar (e, f), nämlich dass es die Kongruenz

$$c \equiv_p y^e \beta^f \tag{3.1}$$

erfüllt. Da es für jedes $e \in \mathbb{Z}_q$ genau ein $f \in \mathbb{Z}_q$ gibt, so dass das Paar (e, f) die Kongruenz (3.1) erfüllt, gibt es genau q solche Paare in $\mathbb{Z}_q \times \mathbb{Z}_q$. Da Alice nur c kennt, sind aus ihrer Sicht diese q Paare alle gleichwahrscheinlich. Wir zeigen nun, dass unabhängig davon, welches $d \in G$ Alice an Bob sendet, genau eines dieser q Paare zusätzlich die Kongruenz

$$d \equiv_p x^e \alpha^f \tag{3.2}$$

erfüllt. Folglich akzeptiert Bob mit der Wahrscheinlichkeit $1/q$.

Seien $c', d', x', y' \in \mathbb{Z}_q$ die zu $c, d, x, y \in G$ gehörigen Exponenten, d.h. $c \equiv_p \alpha^{c'}, \ldots, y \equiv_p \alpha^{y'}$. Dann erfüllt ein Paar (e, f) genau dann die beiden Kongruenzen (3.1) und (3.2), wenn Folgendes gilt:

$$\begin{array}{l} c \equiv_p y^e \beta^f \\ d \equiv_p x^e \alpha^f \end{array} \Leftrightarrow \begin{array}{l} \alpha^{c'} \equiv_p \alpha^{y'e} \cdot \alpha^{af} \\ \alpha^{d'} \equiv_p \alpha^{x'e} \cdot \alpha^{f} \end{array} \Leftrightarrow \begin{array}{l} c' \equiv_q y'e + af \\ d' \equiv_q x'e + f \end{array} \Leftrightarrow \underbrace{\begin{pmatrix} y' & a \\ x' & 1 \end{pmatrix}}_{A} \begin{pmatrix} e \\ f \end{pmatrix} \equiv_q \begin{pmatrix} c' \\ d' \end{pmatrix}.$$

Wegen $\alpha^{y'} \equiv_p y \not\equiv_p x^a \equiv_p \alpha^{x'a}$ folgt $y' \not\equiv_q x'a$ und daher ist $detA \not\equiv_q 0$. □

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

Ableugnungsprotokoll

1 **Bob wählt zufällig** $e_1, f_1 \in \mathbb{Z}_q$ **und sendet** $c_1 = y^{e_1}\beta^{f_1} \bmod p$ **an Alice.**
2 **Alice sendet** $d_1 = c_1^{a^{-1} \bmod q} \bmod p$ **zurück.**
3 **Bob testet, ob** $d_1 \not\equiv_p x^{e_1}\alpha^{f_1}$ **ist.**
4 **Bob wählt zufällig** $e_2, f_2 \in \mathbb{Z}_q$ **und sendet** $c_2 = y^{e_2}\beta^{f_2} \bmod p$ **an Alice.**
5 **Alice sendet** $d_2 = c_2^{a^{-1} \bmod q} \bmod p$ **zurück.**
6 **Bob testet, ob** $d_2 \not\equiv_p x^{e_2}\alpha^{f_2}$ **ist.**
7 **Bob erkennt** y **als ungültig an, falls mindestens einer der Tests in Schritt 3 oder 6 erfolgreich war und** $(d_1\alpha^{-f_1})^{e_2} \equiv_p (d_2\alpha^{-f_2})^{e_1}$ **gilt.**

Bei den Schritten 1-3 und 4-6 handelt es sich jeweils um eine fehlgeschlagene Verifikation der Signatur 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_1 und d_2 gemäß dem Protokoll gewählt hat.

Beispiel 74. *Sei $p = 467, q = 233, \alpha = 4, a = 101, \beta = 449$. Wir nehmen an, dass der Text $x = 286$ mit der Alice zugeschriebenen Signatur $y = 81$ unterschrieben ist und Alice Bob davon überzeugen möchte, dass y ungültig ist.*

1. *Bob wählt $e_1 = 45, f_1 = 237$ und sendet $c_1 = 305$ an Alice.*
2. *Alice antwortet mit $d_1 = c_1^{a^{-1}} = 109$*
3. *Bob verifiziert, dass $286^{45}4^{237} \equiv_p 149 \not\equiv_p 109$ gilt.*
4. *Bob wählt $e_2 = 125, f_2 = 9$ und sendet $c_2 = 72$ an Alice.*
5. *Alice antwortet mit $d_2 = c_2^{a^{-1}} = 68$*
6. *Bob verifiziert, dass $286^{125}4^{9} \equiv_p 25 \not\equiv_p 109$ gilt.*
7. *Bob erkennt y als ungültig an, da $(109 \cdot 4^{-237})^{125} \equiv_p 188 \equiv_p (68 \cdot 4^{-9})^{45}$ ist und somit die Konsistenzbedingung erfüllt ist.*

◁

Es bleibt zu zeigen, dass sich Bob von der Ungültigkeit einer Signatur y im Fall $y \not\equiv_p x^a$ mit sehr hoher und im Fall $y \equiv_p x^a$ nur mit sehr kleiner Wahrscheinlichkeit überzeugen lässt (auch wenn sich im zweiten Fall Alice nicht an das Ableugnungsprotokoll hält).

Behauptung 75. *Im Fall $y \not\equiv_p x^a$ erkennt Bob y mit Wahrscheinlichkeit $1 - \frac{1}{q^2}$ als ungültig an, falls sich beide an das Ableugnungsprotokoll halten.*

Beweis. Nach Behauptung 73 beträgt die Wahrscheinlichkeit, dass beide Tests in Schritt 3 und 6 fehlschlagen genau $\frac{1}{q^2}$. Wegen $\beta \equiv_p \alpha^a$, $c_i \equiv_p y^{e_i}\beta^{f_i}$ und $d_i \equiv_p c_i^{a^{-1} \bmod q}$ für $i \in \{1,2\}$ folgt

$$d_i\alpha^{-f_i} \equiv_p (y^{e_i}\beta^{f_i})^{a^{-1}}\alpha^{-f_i} \equiv_p y^{e_i a^{-1}} \underbrace{\beta^{f_i a^{-1}}}_{\alpha^{f_i}} \alpha^{-f_i} \equiv_p y^{e_i a^{-1}}$$

und somit

$$(d_1\alpha^{-f_1})^{e_2} \equiv_p y^{e_1 a^{-1} e_2} \equiv_p y^{e_2 a^{-1} e_1} \equiv_p (d_2\alpha^{-f_2})^{e_1},$$

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

Behauptung 76. *Im Fall $y \equiv_p x^a$ erkennt Bob y nur mit einer Wahrscheinlichkeit $\leq \frac{1}{q}$ als ungültig an, auch wenn sich Alice nicht an das Ableugnungsprotokoll hält.*

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

$$(d_1 \not\equiv_p x^{e_1}\alpha^{f_1} \text{ oder } d_2 \not\equiv_p x^{e_2}\alpha^{f_2}) \text{ und } (d_1\alpha^{-f_1})^{e_2} \equiv_p (d_2\alpha^{-f_2})^{e_1}$$

gilt. Da die beiden Fälle $d_1 \not\equiv_p x^{e_1}\alpha^{f_1}$ und $d_2 \not\equiv_p x^{e_2}\alpha^{f_2}$ symmetrisch sind, reicht es, einen davon zu betrachten.

Wir nehmen also an, dass Alice eine Zahl $d_1 \not\equiv_p x^{e_1}\alpha^{f_1}$ an Bob sendet. Nachdem Alice die Zahl c_2 in Zeile 4 von Bob erhalten hat, weiß sie nur, dass das von Bob gewählte Paar (e_2, f_2) die Kongruenz $c_2 \equiv_p y^{e_2}\beta^{f_2}$ erfüllt. Wie wir bereits im Beweis zu Behauptung 73 gesehen haben, trifft dies auf genau q Paare zu. Wir zeigen nun, dass für jedes $d_2 \in G$ genau eines dieser q Paare die Konsistenzbedingung

$$(d_1\alpha^{-f_1})^{e_2} \equiv_p (d_2\alpha^{-f_2})^{e_1}$$

erfüllt. Dies beweist, dass unabhängig davon, welches d_2 Alice an Bob sendet, Bob y nur mit Wahrscheinlichkeit $1/q$ als ungültig akzeptiert.

Sei $u = d_1\alpha^{-f_1} \bmod p$ und seien $c_2', d_2', x', u' \in \mathbb{Z}_q$ die zu c_2, d_2, x, u gehörigen Exponenten. Dann gilt

$$\begin{array}{l} c_2 \equiv_p y^{e_2}\beta^{f_2} \\ (\underbrace{d_1\alpha^{-f_1}}_{u})^{e_2} \equiv_p (d_2\alpha^{-f_2})^{e_1} \end{array} \Leftrightarrow \begin{array}{l} c_2' \equiv_q x'ae_2 + af_2 \\ u'e_2 \equiv_q d_2'e_1 - e_1f_2 \end{array} \Leftrightarrow \underbrace{\begin{pmatrix} x'a & a \\ u' & e_1 \end{pmatrix}}_{A} \begin{pmatrix} e_2 \\ f_2 \end{pmatrix} \equiv_q \begin{pmatrix} c_2' \\ d_2'e_1 \end{pmatrix}.$$

Wegen

$$x^{e_1} \equiv_p \underbrace{x^{e_1}\alpha^{f_1}}_{\not\equiv_p d_1}\alpha^{-f_1} \not\equiv_p d_1\alpha^{-f_1} \equiv_p u$$

folgt $x'e_1 \not\equiv_q u'$ und somit ist $detA = x'ae_1 - u'a = a(x'e_1 - u') \not\equiv_q 0$. □

3.9 Fail-Stop-Signaturen

Ein Nachteil aller bisher betrachteten Signaturverfahren ist, dass Alice eine vorgelegte Fälschung (x, y) nicht als solche nachweisen kann. Dies liegt daran, dass Alice einen Dritten nicht davon überzeugen kann, dass sie die Signatur y nicht selbst erzeugt hat. Bei sog. Fail-Stop-Signaturen ist genau dies möglich: Sollte es einem Angreifer gelingen, das Signaturverfahren zu brechen ("fail") und eine Fälschung (x, y) zu generieren, so kann Alice dies mit hoher Wahrscheinlichkeit beweisen und somit ihre Signatur widerrufen ("stop").

Das van Heyst-Pedersen Signaturverfahren

Definition 77. *Sei $p = 2q + 1$ prim, p, q prim und sei $\alpha \in \mathbb{Z}_p^*$ ein Element der Ordnung q. Weiter sei $G = \{\alpha^a | a \in \mathbb{Z}_q\}$ die von α in $\mathbb{Z}_p^*$ erzeugte Untergruppe und $\beta = \alpha^a \bmod p$ für ein $a \in \mathbb{Z}_q^*$.*

Die Zahlen p, q, α, β werden von einer vertrauenswürdigen Instanz generiert und bekannt gegeben, a wird jedoch vor allen Teilnehmern geheim gehalten.

Der Textraum ist $X = \mathbb{Z}_q$ und der Signaturenraum ist $Y = \mathbb{Z}_q \times \mathbb{Z}_q$.

Um einen Signierschlüssel zu generieren, wird zufällig ein 4-Tupel $\hat{k} = (a_1, b_1, a_2, b_2) \in_R \mathbb{Z}_q^4$ gewählt. Der zugehörige Verifikationsschlüssel ist $k = (\gamma_1, \gamma_2) = (\alpha^{a_1}\beta^{b_1}, \alpha^{a_2}\beta^{b_2}) \in G^2$.

Signaturerstellung: *Die Signatur für einen Text $x \in \mathbb{Z}_q$ unter $\hat{k} = (a_1, b_1, a_2, b_2) \in \mathbb{Z}_q^4$ ist*

$$sig(\hat{k}, x) = (y_1, y_2) = (a_1 + xa_2 \bmod q, b_1 + xb_2 \bmod q).$$

Verifikation: *Für einen Verifikationsschlüssel $k = (\gamma_1, \gamma_2)$, eine Signatur $y = (y_1, y_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$ und einen Text $x \in \mathbb{Z}_q$ gilt*

$$ver(k, x, y) = \begin{cases} 1, & \gamma_1\gamma_2^x \equiv_p \alpha^{y_1}\beta^{y_2}, \\ 0, & sonst. \end{cases}$$

Sei

$$S = \{(\hat{k}, k) \mid \hat{k} = (a_1, b_1, a_2, b_2) \in \mathbb{Z}_q^4, k = (\alpha^{a_1}\beta^{b_1}, \alpha^{a_2}\beta^{b_2}) \in G \times G\}$$

die Menge aller möglichen Schlüsselpaare. Für einen Verifikationsschlüssel $k \in G \times G$ sei

$$S(k) = \{\hat{k} \in \mathbb{Z}_q^4 \mid (\hat{k}, k) \in S\}$$

die Menge aller Signierschlüssel, die zu k passen, und für einen Text x und eine Signatur $y = (y_1, y_2)$ sei

$$S(k, x, y) = \{\hat{k} \in S(k) \mid sig(\hat{k}, x) = y\}$$

die Menge aller Signierschlüssel in $S(k)$, die für x die Signatur y berechnen.

Lemma 78. *Für jeden Signierschlüssel $\hat{k} \in S(k)$ und jedes Paar (x, y) mit $sig(\hat{k}, x) = y$ ist die Verifikationsbedingung $ver(k, x, y) = 1$ erfüllt.*

Beweis. Sei $\hat{k} = (a_1, b_1, a_2, b_2)$ und $sig(\hat{k}, x) = y = (y_1, y_2)$. Wegen $\hat{k} \in S(k)$ folgt $k = (\gamma_1, \gamma_2) = (\alpha^{a_1}\beta^{b_1}, \alpha^{a_2}\beta^{b_2})$ und daher gilt

$$\begin{aligned} \gamma_1\gamma_2^x &\equiv_p \alpha^{a_1}\beta^{b_1}(\alpha^{a_2}\beta^{b_2})^x \\ &\equiv_p \alpha^{a_1 + xa_2}\beta^{b_1 + xb_2} \\ &\equiv_p \alpha^{y_1}\beta^{y_2} \end{aligned}$$

□

Anders gesagt gibt es im Fall $ver(k, x, y) = 0$ keinen Signierschlüssel $\hat{k} \in S(k)$ mit $sig(\hat{k}, x) = y$, d.h. $S(k, x, y) = \emptyset$. Das nächste Lemma zeigt, dass $S(k, x, y)$ im Fall $ver(k, x, y) = 1$ genau q Signierschlüssel enthält.

Lemma 79. *Zu jedem Paar (x, y) mit $ver(k, x, y) = 1$ gibt es genau q Signierschlüssel $\hat{k} \in S(k)$ mit $sig(\hat{k}, x) = y$.*

Beweis. Wir zeigen zuerst, dass $S(k)$ für jeden Verifikationsschlüssel $k = (\gamma_1, \gamma_2)$ genau q^2 Signierschlüssel enthält. Ein Signierschlüssel $\hat{k} = (a_1, b_1, a_2, b_2)$ ist genau dann in $S(k)$, wenn er die beiden Kongruenzen

$$\begin{aligned} \alpha^{a_1}\beta^{b_1} &\equiv_p \gamma_1 \\ \alpha^{a_2}\beta^{b_2} &\equiv_p \gamma_2 \end{aligned}$$

erfüllt. Seien $c_1, c_2 \in \mathbb{Z}_q$ eindeutig bestimmte Exponenten mit $\gamma_1 \equiv_p \alpha^{c_1}$ und $\gamma_2 \equiv_p \alpha^{c_2}$. Dann sind diese Kongruenzen äquivalent zu

$$\begin{aligned} a_1 + ab_1 &\equiv_q c_1 \\ a_2 + ab_2 &\equiv_q c_2 \end{aligned} \quad \text{bzw.} \quad \underbrace{\begin{pmatrix} 1 & a & 0 & 0 \\ 0 & 0 & 1 & a \end{pmatrix}}_{A} \begin{pmatrix} a_1 \\ b_1 \\ a_2 \\ b_2 \end{pmatrix} \equiv_q \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \quad (*)$$

Da A den Rang 2 hat, folgt $\|S(k)\| = q^2$. Sei nun (x, y) ein Paar mit $x \in \mathbb{Z}_q$ und $y = (y_1, y_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$. Dann ist ein Signierschlüssel $\hat{k} = (a_1, b_1, a_2, b_2)$ genau dann in $S(k, x, y)$, wenn er die Kongruenzen

$$\begin{aligned} a_1 + ab_1 &\equiv_q c_1 \\ a_2 + ab_2 &\equiv_q c_2 \\ a_1 + xa_2 &\equiv_q y_1 \\ b_1 + xb_2 &\equiv_q y_2 \end{aligned} \quad \text{bzw.} \quad \underbrace{\begin{pmatrix} 1 & a & 0 & 0 \\ 0 & 0 & 1 & a \\ 1 & 0 & x & 0 \\ 0 & 1 & 0 & x \end{pmatrix}}_{A'} \begin{pmatrix} a_1 \\ b_1 \\ a_2 \\ b_2 \end{pmatrix} \equiv_q \underbrace{\begin{pmatrix} c_1 \\ c_2 \\ y_1 \\ y_2 \end{pmatrix}}_{s} \quad (**)$$

erfüllt. Wir zeigen, dass A' den Rang $rang(A') = 3$ hat. Seien $r_1, \ldots, r_4$ die Zeilen von A'. Dann gilt $rang(A') \geq 3$, da die Zeilen r_2, r_3, r_4 linear unabhängig sind, und $rang(A') \leq 3$, da $r_1 = r_3 + ar_4 - xr_2$ ist. Damit hat $(**)$ im Falle der Lösbarkeit genau $q^{4-3} = q$ Lösungen. Zum Nachweis der Lösbarkeit von (**) zeigen wir, dass die in A' bestehende Zeilenabhängigkeit $r_1 = r_3 + ar_4 - xr_2$ im Fall $ver(k, x, y) = 1$ auch für den Spaltenvektor s auf der rechten Seite von (**) gilt:

$$\gamma_1\gamma_2^x \equiv_p \alpha^{y_1}\beta^{y_2} \;\Rightarrow\; c_1 + xc_2 \equiv_q y_1 + ay_2 \;\Rightarrow\; c_1 \equiv_q y_1 + ay_2 - xc_2.$$

Da somit die Erweiterung der Matrix A' um den Spaltenvektor s deren Rang im Fall $ver(k, x, y) = 1$ nicht erhöht, ist (**) in diesem Fall lösbar. □

Lemma 80. *Für alle $x, x' \in \mathbb{Z}_q$ und $y = (y_1, y_2), y' = (y_1', y_2') \in \mathbb{Z}_q^2$ mit $x' \neq x$ gilt*

$$\|S(k, x, y) \cap S(k, x', y')\| \leq 1.$$

Im Fall $ver(k, x, y) = ver(k, x', y') = 1$ gilt sogar Gleichheit.

Beweis. Die Bedingung $\hat{k} = (a_1, b_1, a_2, b_2) \in S(k, x, y) \cap S(k, x', y')$ ist äquivalent zu

$$\underbrace{\begin{pmatrix} 1 & a & 0 & 0 \\ 0 & 0 & 1 & a \\ 1 & 0 & x & 0 \\ 0 & 1 & 0 & x \\ 1 & 0 & x' & 0 \\ 0 & 1 & 0 & x' \end{pmatrix}}_{A''} \begin{pmatrix} a_1 \\ b_1 \\ a_2 \\ b_2 \end{pmatrix} = \underbrace{\begin{pmatrix} c_1 \\ c_2 \\ y_1 \\ y_2 \\ y_1' \\ y_2' \end{pmatrix}}_{s'} \quad (***)$$

wobei wieder $\gamma_1 \equiv_p \alpha^{c_1}, \gamma_2 \equiv_p \alpha^{c_2}$ ist. Wir zeigen, dass die Zeilen r_1, r_2, r_4, r_6 von A'' linear unabhängig sind und somit A'' den Rang $rang(A) = 4$ hat. Daraus folgt, dass (***) höchstens eine Lösung hat.

Aus $l_1r_1+l_2r_2+l_4r_4+l_6r_6 = \vec{0}$ folgt nämlich $l_1 = l_2 = 0$ und $l_4+l_6 = 0$ sowie $l_4x+l_6x' = 0$, was $l_6 = -l_4$ sowie $l_4(x - x') = 0$ und somit wegen $x - x' \neq 0$ auch $l_4 = l_6 = 0$ impliziert.

Da auch die Zeilen $r_3, \ldots, r_6$ von A'' linear unabhängig sind, lässt sich $\hat{k}$ bei Kenntnis zweier Signaturen $y = sig(\hat{k}, x)$ und $y' = sig(\hat{k}, x')$ für zwei Texte $x \neq x'$ leicht bestimmen, d.h. es handelt sich um ein *One-time-Signaturverfahren*.

Um die Lösbarkeit von (***) im Fall $ver(k, x, y) = ver(k, x', y') = 1$ nachzuweisen, zeigen wir, dass die in A'' bestehenden Zeilenabhängigkeiten $r_3 = r_1 + xr_2 - ar_4$ und $r_5 = r_1 + x'r_2 - ar_6$ auch für den Spaltenvektor s' auf der rechten Seite von (***) gelten: Aus $ver(k, x, y) = 1$ folgt

$$\gamma_1\gamma_2^x \equiv_p \alpha^{y_1}\beta^{y_2} \;\Rightarrow\; c_1 + xc_2 \equiv_q y_1 + ay_2 \;\Rightarrow\; y_1 \equiv_q c_1 + xc_2 - ay_2$$

und analog folgt aus $ver(k, x', y') = 1$ die Kongruenz $y'_1 \equiv_q c_1 + x'c_2 - ay'_2$. □

Im nächsten Satz zeigen wir, dass ein Angreifer, der den Verifikationsschlüssel k und eine von Alice für einen Text x erzeugte Signatur $y = sig(\hat{k}, x)$ kennt, nur mit Wahrscheinlichkeit höchstens $1/q$ ein Paar (x', y') mit $x' \neq x$ und $y' = sig(\hat{k}, x')$ finden kann. Dies gilt auch, wenn der Angreifer über unbeschränkte Rechenressourcen verfügt.

Lemma 81. *Für alle $x, x' \in \mathbb{Z}_q$, $y, y' \in \mathbb{Z}_q^2$ und $k = (\gamma_1, \gamma_2) \in G^2$ mit $x' \neq x$ und $ver(k, x, y) = 1$ gilt*

$$Prob_{\hat{k}\in_R\mathbb{Z}_q^4}[\underbrace{sig(\hat{k}, x') = y'}_{A} \mid \underbrace{\hat{k} \in S(k, x, y)}_{B}] \leq \frac{1}{q}$$

Beweis. Es gilt

$$\Pr[A|B] = \frac{Prob[A \cap B]}{Prob[B]} = \frac{\|S(k, x', y') \cap S(k, x, y)\|}{\|S(k, x, y)\|} \leq \frac{1}{q}.$$

□

Frage: Wie funktioniert der *Fail-Stop-Mechanismus*? D.h. wie kann Alice bei Vorlage eines Paares (x', y') mit $ver(k, x', y') = 1$ und $y' \neq sig(\hat{k}, x')$ beweisen, dass die gültige Signatur y' nicht von ihr erzeugt wurde?

Antwort: Sie berechnet $y'' = sig(\hat{k}, x')$ und benutzt das Tripel (x', y', y''), um a zu berechnen. Wegen

$$ver(k, x', y''_1, y''_2) = 1 = ver(k, x', y'_1, y'_2)$$

folgt

$$\alpha^{y'_1}\beta^{y'_2} \equiv_p \gamma_1\gamma_2^{x'} \equiv_p \alpha^{y''_1}\beta^{y''_2} \text{ und somit } y'_1 + ay'_2 \equiv_q y''_1 + ay''_2,$$

weshalb

$$a = (y''_1 - y'_1)(y'_2 - y''_2)^{-1} \bmod q$$

ist. Dabei ist zu beachten, dass die Gleichheit von y'_2 und y''_2 wegen $\alpha^{y'_1}\beta^{y'_2} \equiv_p \alpha^{y''_1}\beta^{y''_2}$ auch die Gleichheit von y'_1 und y''_1 (also $y' = y''$) implizieren würde, was wir ausgeschlossen haben.

Beispiel 82. *Die vertrauenswürdigen Instanz (TTP, trusted third party) generiert Primzahlen p und q mit $p = 3467 = 2 \cdot \underbrace{1733}_{=q} + 1$, sowie ein Element $\alpha = 4 \in \mathbb{Z}_p^*$ mit $ord_p(\alpha) = q$ und eine geheime Zahl $a = 1567 \in \mathbb{Z}_q^*$ und gibt die Zahlen p, q, α und $\beta = \alpha^a \bmod p = 4^{1567} \bmod p = 514$ bekannt, hält aber a geheim.*
Angenommen Alice wählt

$$\hat{k} = (\underbrace{888}_{a_1}, \underbrace{1024}_{b_1}, \underbrace{786}_{a_2}, \underbrace{999}_{b_2}),$$

so berechnet sich k zu

$$k = (\gamma_1, \gamma_2), \text{ wobei}$$
$$\gamma_1 = \alpha^{a_1}\beta^{b_1} = 4^{888}514^{1024} = 3405$$

und

$$\gamma_2 = \alpha^{a_2}\beta^{b_2} = 4^{786}514^{999} = 2281$$

ist. Wird nun Alice mit dem Paar $(x', y') = (x', (y'_1, y'_2)) = (3383, (822, 55))$ konfrontiert, das wegen

$$\begin{aligned} \gamma_1\gamma_2^{x'} &= 3405 \cdot 2281^{3383} \equiv_p 2282 \\ \text{und } \alpha^{y''_1}\beta^{y''_2} &= 4^{822}515^{55} \equiv_p 2282 \end{aligned}$$

die Verifikationsbedingung $ver(k, x', y') = 1$ erfüllt, so berechnet Alice zunächst

$$y'' = sig(k, x') = (y''_1, y''_2)$$

mit

$$\begin{aligned} y''_1 &= a_1 + x'a_2 \bmod q = 888 + 3383 \cdot 786 \bmod \text{ und } q = 1504 \\ y''_2 &= b_1 + x'b_2 \bmod q = 1024 + 3383 \cdot 999 \bmod q = 1291, \end{aligned}$$

um sich zu vergewissern, dass $y' \neq y''$ ist. Hieraus erhält sie dann a zu

$$a = \frac{y''_1 - y'_1}{y'_2 - y''_2} \bmod q = \frac{1504 - 822}{55 - 1291} \bmod q = 1567.$$

◁

4 Pseudozufallszahlen-Generatoren

Pseudozufallszahlen-Generatoren (kurz PZG) f werden mit einem Startwert x – dem sogenannten *Keim* (engl. seed) – für die Erzeugung einer „zufälligen“ Bitfolge $f(x)$ gestartet. Dabei wird die Eingabe x zufällig unter Gleichverteilung gewählt und die Ausgabe $f(x)$ sollte länger sein als x und möglichst zufällig aussehen. Zudem sollte f von einem deterministischen Algorithmus effizient berechenbar sein.

Linear-Kongruenz-Generator

Der Keim x_0 wird zufällig aus der Menge $\mathbb{Z}_n = \{0, 1, \ldots n-1\}$gewählt. Die Parameter a und b sind ebenfalls aus $\mathbb{Z}_n$.

Algorithmus $\texttt{LinGen}_{n,l,a,b}(x_0)$

```
1 for i := 1 to l do
2     x_i := a · x_{i-1} + b mod n
3     b_i := x_i mod 2
4 ouput(b_1, ... b_l)
```

Power-Generator

Der Keim x_0 wird zufällig aus der Menge $\mathbb{Z}_n^*$ gewählt.

Algorithmus $\texttt{PowerGen}_{n,l,e}(x_0)$

```
1 for i := 1 to l do
2     x_i := x_{i-1}^e mod n
3     b_i := x_i mod 2
4 ouput(b_1, ... b_l)
```

Es gibt zwei interessante Spezialfälle des Powergenerators:

- *RSA-Generator* (RsaGen) mit $n = p \cdot q$ wobei $p, q \in \mathsf{IP}$ und $\mathrm{ggT}(e, \varphi(n)) = 1$
- *Quadratischer-Reste-Generator* (BBS) mit $e = 2$ (siehe folgenden Abschnitt).

4.1 Kryptografische Sicherheit von Pseudozufallsgeneratoren

Wir betrachten hier nur den Fall, dass sowohl x als auch $f(x)$ Bitfolgen sind und die Länge der Ausgabe nur von der Länge der Eingabe abhängt.

Sei $l = l(k) \geq k+1$ eine Funktion. Ein $l(k)$-*Generator* ist eine Funktion f auf $\{0,1\}^*$, die Strings der Länge k auf Strings der Länge $l(k)$ abbildet und in Polynomialzeit berechenbar ist.

Seien (X_k) und (Y_k) Familien von Zufallsvariablen mit Wertebereich $W(X_k), W(Y_k) \subseteq \{0,1\}^{l(k)}$ und sei $\varepsilon : \mathbb{N} \to \mathbb{R}^+$ eine Funktion. Ein ε-*Unterscheider* zwischen (X_k) und

(Y_k) ist ein in Polynomialzeit berechenbarer probabilistischer Algorithmus D, so dass für unendlich viele Werte von k gilt:

$$|\Pr[D(X_k) = 1] - \Pr[D(Y_k) = 1]| \geq \varepsilon(l(k)).$$

Hierbei ist $\Pr[D(X_k) = 1]$ die Wahrscheinlichkeit, dass D bei einer zufällig gemäß X_k gewählten Eingabe akzeptiert (bzw. 1 ausgibt). In diesem Fall heißen die beiden Familien (X_k) und (Y_k) *ε-unterscheidbar*.

Ein $l(k)$-Generator f heißt *ε-unterscheidbar*, falls die beiden Familien $(f(U_k))$ und $(U_{l(k)})$ von Zufallsvariablen ε-unterscheidbar sind, wobei U_n eine auf $\{0,1\}^n$ gleichverteilte Zufallsvariable ist. f heißt *(kryptografisch) sicher*, falls f für kein Polynom p $1/p$-unterscheidbar ist. f ist also genau dann sicher, wenn f nur für vernachlässigbare Funktionen ε-unterscheidbar ist.

Beispiel 83. *Betrachte folgenden Unterscheider D für den $(k+1)$-Generator f mit $f(x) = 1x$ für alle $x \in \{0,1\}^*$.*

```
1 input y = y_1 ··· y_{k+1} ∈ {0,1}^{k+1}
2 output(y_1)
```

Dann gilt $\Pr[D(f(U_k)) = 1] = 1$ *und* $\Pr[D(U_{k+1}) = 1] = 1/2$ *und somit*

$$|\Pr[D(f(U_k)) = 1] - \Pr[D(U_{k+1}) = 1]| = 1/2$$

für alle k. Folglich ist f $(1/2)$-unterscheidbar. ◁

Es ist nicht bekannt, ob kryptografisch sichere PZGen existieren. Eine notwendige Bedingung hierfür ist $\mathsf{P} \neq \mathsf{NP}$, da $\mathsf{P} = \mathsf{NP}$ die Existenz eines effizienten Unterscheiders impliziert, welcher genau die Strings im Bild von f akzeptiert. Ob diese Bedingung auch hinreichend ist, ist ebenfalls nicht bekannt. Man kann jedoch zeigen, dass die Existenz von kryptografisch sicheren PZGen äquivalent zur Existenz von Einwegfunktionen ist.

Bei manchen Anwendungen ist es wichtig, dass kein effizienter Algorithmus das nächste Bit der Pseudozufallsfolge korrekt vorhersagen kann. Es ist nicht schwer zu sehen, dass ein sicherer PZG diese Bedingung erfüllt.

Ein probabilistischer Algorithmus N heißt *ε-next bit predictor (ε-NBP)* für f, falls für unendlich viele k

$$\Pr[N(f_{[I-1]}(U_k), 1^{l(k)}) = f_I(U_k)] \geq 1/2 + \varepsilon(l(k))$$

ist, wobei die Zufallsvariable I auf der Menge $\{1, \ldots, l(k)\}$ gleichverteilt ist. Hierbei bezeichnet $f_i(x)$ das i-te Bit und $f_{[i]}(x)$ die Folge der ersten i Bits von $f(x)$.

Beispiel 84. *Betrachte folgenden NBP N für den $(k+1)$-Generator f mit $f(x) = 1x$ für alle $x \in \{0,1\}^*$.*

```
1 input (y, 1^l) mit y = y_1 ··· y_{i−1} ∈ {0,1}^{i−1} für ein i ∈ {1, ..., l}
2 output(1)
```

Dann gilt

$$\Pr[N(f_{[i-1]}(U_k)) = f_i(U_k)] = \begin{cases} 1, & i = 1 \\ 1/2, & i = 2, \ldots k+1 \end{cases}$$

und somit

$$\Pr[N(f_{[I-1]}(U_k)) = f_I(U_k)] = (1/(k+1)) \sum_{i=1}^{k+1} \Pr[N(f_{[i-1]}(U_k)) = f_i(U_k)] = 1/2 + 1/\underbrace{(2k+2)}_{2l(k)}.$$

Folglich ist N ein $(1/2l)$-NBP für f. ◁

Satz 85. *Falls es einen ε-NBP N für f gibt, so ex. auch ein ε-Unterscheider für f.*

Beweis. Sei N ein ε-NBP für f und betrachte folgenden Unterscheider D.

---

1 **input** $y = y_1 \cdots y_l$
2 `wähle` $i \in_R \{1, \ldots, l\}$
3 **output**$(N(y_1 \cdots y_{i-1}, 1^l) \oplus y_i \oplus 1)$

D gibt also bei Eingabe $y = y_1 \cdots y_l$ genau dann 1 aus, wenn der Prediktor N bei Eingabe $y_1 \cdots y_{i-1}$ das i-te Bit von y richtig rät, wobei i zufällig gewählt wird. Daher ist

$$\Pr[D(f(U_k)) = 1] = \Pr[N(f_{(I-1)}(U_k), 1^l) = f_I(U_k)] \geq 1/2 + \varepsilon.$$

Andererseits ist klar, dass jeder NBP das i-te Bit y_i einer rein zufälligen Eingabe y genau mit Wahrscheinlichkeit $1/2$ richtig rät, und somit $\Pr[D(U_{l(k)}) = 1] = 1/2$ ist. □

Ein probabilistischer Algorithmus P heißt *ε-previous bit predictor (ε-PBP)* für f, falls für unendlich viele k gilt:

$$\Pr[P(f_{I+1}(U_k) \cdots f_{l(k)}(U_k), 1^{l(k)}) = f_I(U_k)] \geq 1/2 + \varepsilon(l(k)).$$

Vollkommen analog zu obigem Satz lässt sich der folgende Satz beweisen.

Satz 86. *Falls es einen ε-PBP N für f gibt, so ex. auch ein ε-Unterscheider für f.*

Interessanterweise lässt sich aus einem Unterscheider auch ein NBP bzw. PBP gewinnen. Um also die Sicherheit eines PZG f zu beweisen, genügt der Nachweis, dass es für kein Polynom p einen $(1/p)$-NBP für f gibt.

Satz 87. *Falls es einen ε-Unterscheider D für f gibt, so ex. auch ein (ε/l)-NBP für f.*

Beweis. Wir können o.B.d.A. annehmen, dass

$$\Pr[D(U_{l(k)}) = 1] - \Pr[D(f(U_k)) = 1] \geq \varepsilon(l(k))$$

für unendlich viele k gilt, da wir andernfalls D invertieren können. Die Ausgabe $D(y) = 1$ deutet also darauf hin, dass y tendenziell ein echter Zufallsstring ist, während die Ausgabe $D(y) = 0$ darauf hindeutet, dass y ein Pseudozufallsstring ist. Wir zeigen, dass folgender probabilistische Algorithmus N ein (ε/l)-NBP für f ist.

---

1 **input** $(y_1 \cdots y_{i-1}, 1^l)$ `mit` $1 \leq i \leq l$
2 `rate zufällig` $b_i, \ldots, b_l \in_R \{0, 1\}$
3 **output**$(D(y_1 \cdots y_{i-1} b_i \cdots b_l) \oplus b_i)$

N sagt also das i-te Bit y_i mit b_i vorher, falls D den String $y_1 \cdots y_{i-1} b_i \cdots b_l$ für pseudozufällig hält (also $D(y_1 \cdots y_{i-1} b_i \cdots b_l) = 0$ ist), und sonst mit $b_i \oplus 1$. Betrachte für $i = 1, \ldots, l(k)$ die Zufallsvariablen

$$H_i = f_{[i-1]}(U_k) B_i \cdots B_{l(k)},$$

wobei U_k, $B_i, \ldots, B_{l(k)}$ unabhängig und gleichverteilt auf $\{0,1\}^k$ bzw. $\{0,1\}$ sind. Insbesondere ist also $H_1 = B_1 \cdots B_{l(k)} = U_{l(k)}$ gleichverteilt auf $\{0,1\}^{l(k)}$ und $H_{l(k)+1} = f(U_k)$ pseudozufällig verteilt auf $\{0,1\}^{l(k)}$.

Behauptung 88. *Es gilt*

$$\Pr[N(f_{[i-1]}(U_k), 1^{l(k)}) = f_i(U_k)] \geq 1/2 + \Pr[D(H_i) = 1] - \Pr[D(H_{i+1}) = 1].$$

Beweis. Wegen

$$N(f_{[i-1]}(U_k), 1^{l(k)}) = D(\underbrace{f_{[i-1]}(U_k) B_i \cdots B_{l(k)}}_{H_i}) \oplus B_i = D(H_i) \oplus B_i$$

folgt

$$\begin{aligned}
\Pr[N(f_{[i-1]}(U_k), 1^{l(k)}) = f_i(U_k)] \;&=\; \Pr[D(H_i) \oplus B_i = f_i(U_k)] \\
&= \underbrace{\Pr[D(H_i) = 0 \wedge B_i = f_i(U_k)]}_{\Pr[B_i = f_i(U_k)] - \Pr[B_i = f_i(U_k) \wedge D(H_i) = 1]} + \underbrace{\Pr[D(H_i) = 1 \wedge B_i \neq f_i(U_k)]}_{\Pr[D(H_i)=1] - \Pr[D(H_i)=1 \wedge B_i = f_i(U_k)]} \\
&= \underbrace{\Pr[B_i = f_i(U_k)]}_{1/2} + \Pr[D(H_i) = 1] - 2 \underbrace{\Pr[D(H_i) = 1 \wedge B_i = f_i(U_k)]}_{\substack{\Pr[D(H_{i+1})=1 \wedge B_i = f_i(U_k)] \\ = \Pr[D(H_{i+1})=1] \underbrace{\Pr[B_i = f_i(U_k)]}_{1/2}}} \\
&= 1/2 + \Pr[D(H_i) = 1] - \Pr[D(H_{i+1}) = 1].
\end{aligned}$$

□

Sei die Zufallsvariable I auf $\{1, \ldots, l(k)\}$ gleichverteilt. Dann folgt

$$\begin{aligned}
\Pr[N(f_{[I-1]}(U_k), 1^{l(k)}) = f_I(U_k)] &= 1/2 + \Pr[D(H_I) = 1] - \Pr[D(H_{I+1}) = 1] \\
&= 1/2 + \sum_{i=1}^{l(k)} \underbrace{\Pr[I = i]}_{1/l(k)} (\Pr[D(H_i) = 1] - \Pr[D(H_{i+1}) = 1]) \\
&= 1/2 + (\Pr[D(\underbrace{H_1}_{U_{l(k)}}) = 1] - \Pr[D(\underbrace{H_{l(k)+1}}_{f(U_k)}) = 1])/l(k)
\end{aligned}$$

Somit gilt $\Pr[N(f_{[I-1]}(U_k), 1^{l(k)}) = f_I(U_k)] \geq 1/2 + \varepsilon(l(k))/l(k)$ für unendlich viele k. □

Ganz ähnlich wie der obige Satz lässt sich auch folgendes Resultat beweisen.

Satz 89. *Falls es einen ε-Unterscheider D für f gibt, so ex. auch ein (ε/l)-PBP für f.*

4.2 Quadratische Reste

In diesem Abschnitt beschäftigen wir uns mit dem Problem, Lösungen für eine quadratische Kongruenzgleichung

$$x^2 \equiv_m a \tag{4.1}$$

zu bestimmen. Zunächst gehen wir der Frage nach, wie sich feststellen lässt, ob überhaupt Lösungen existieren.

Definition 90. *Ein Element $a \in \mathbb{Z}_m^*$ heißt* ***quadratischer Rest modulo m*** *(kurz:* $\boldsymbol{a \in \mathsf{QR}_m}$*), falls ein $x \in \mathbb{Z}_m^*$ existiert mit $x^2 \equiv_m a$.* $\mathbf{QNR_m} := \mathbb{Z}_m^* \setminus \mathsf{QR}_m$ *ist die Menge der* ***quadratischen Nichtreste modulo m****.*

Falls $m = p$ prim ist, lassen sich quadratische Reste effizient von quadratischen Nichtresten modulo p unterscheiden. Sei $p > 2$ eine Primzahl und $a \in \mathbb{Z}$. Dann heißt

$$\mathcal{L}(a,p) = \left(\frac{a}{p}\right) = \begin{cases} 1, & a \bmod p \in \mathsf{QR}_p \\ -1, & a \bmod p \in \mathsf{QNR}_p \\ 0, & \textit{sonst} \end{cases}$$

das ***Legendre-Symbol von a modulo p****.*

Die Kongruenzgleichung (4.1) besitzt also für ein $a \in \mathbb{Z}_m^*$ genau dann eine Lösung, wenn $a \in \mathsf{QR}_m$ ist. Da mit $a, b \in \mathsf{QR}_m$ auch $ab \in \mathsf{QR}_m$ ist, bildet QR_m eine Untergruppe von $\mathbb{Z}_m^*$. Wie das folgende Lemma zeigt, kann die Lösbarkeit von (4.1) für primes m effizient entschieden werden.

Lemma 91. *Sei $a \in \mathbb{Z}_p^*$, $p > 2$ prim, und sei g ein beliebiger Erzeuger von $\mathbb{Z}_p^*$. Dann sind die folgenden drei Bedingungen äquivalent:*

1. $a \in \mathsf{QR}_p$.
2. $a^{(p-1)/2} \equiv_p 1$,
3. $\log_{p,g}(a)$ *ist gerade,*

Beweis.

$1 \Rightarrow 2$: Sei $a \in \mathsf{QR}_p$, d. h. $b^2 \equiv_p a$ für ein $b \in \mathbb{Z}_p^*$. Dann folgt mit dem Satz von Fermat,

$$a^{(p-1)/2} \equiv_p b^{p-1} \equiv_p 1.$$

$2 \Rightarrow 3$: Angenommen, $a \equiv_p g^k$ für ein ungerades $k = 2 \cdot j + 1$. Dann ist

$$a^{(p-1)/2} \equiv_p \underbrace{g^{j\cdot(p-1)}}_{\equiv_p 1} g^{(p-1)/2} \equiv_p g^{(p-1)/2} \not\equiv_p 1.$$

$3 \Rightarrow 1$: Ist $a \equiv_p g^k$ für $k = 2j$ gerade, so folgt $a \equiv_p (g^j)^2$, also $a \in \mathsf{QR}_p$.

□

Somit zerfällt $\mathbb{Z}_p$ in die drei Teilmengen QR_p, QNR_p und $\mathbb{Z}_p \setminus \mathbb{Z}_p^* = \{0\}$, wobei die ersten beiden jeweils $(p-1)/2$ Elemente enthalten. Zudem ist das Produkt ab von $a, b \in \mathbb{Z}_p^*$ genau dann in QR_p, wenn $a, b \in \mathsf{QR}_p$ oder $a, b \in \mathsf{QNR}_p$ sind. Als weitere Folgerung erhalten wir folgende Formel zur effizienten Berechnung des Legendre-Symbols.

Satz 92 (Eulers Kriterium)**.** *Für alle $a \in \mathbb{Z}$ und $p > 2$ prim gilt*

$$a^{(p-1)/2} \equiv_p \left(\frac{a}{p}\right).$$

Beweis. Es ist klar, dass diese Kongruenz im Fall $a \equiv_p 0$ gilt. Nach obigem Lemma gilt dies auch im Fall $a \bmod p \in \mathsf{QR}_p$, da dann $a^{(p-1)/2} \equiv_p 1$ ist.

Es bleibt also der Fall, dass $a \bmod p \in \mathsf{QNR}_p$ ist. Da die Kongruenz $x^2 \equiv_p 1$ nach dem Satz von Lagrange modulo p nur die beiden Lösungen ± 1 hat und $a^{(p-1)/2}$ nach dem Satz von Fermat eine Lösung dieser Kongruenz ist, muss $a^{(p-1)/2} \equiv_p -1$ sein. Andernfalls wäre $a^{(p-1)/2} \equiv_p 1$ und somit $a \bmod p \in \mathsf{QR}_p$ (d.h. $a \bmod p \notin \mathsf{QNR}_p$). □

Korollar 93. *Für alle $a, b \in \mathbb{Z}$ und $p > 2$ prim gilt*

1. $\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2} = \begin{cases} 1, & p \equiv_4 1, \\ -1, & p \equiv_4 3, \end{cases}$
2. $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{b}{p}\right)$.

Als weiteres Korollar aus Eulers Kriterium erhalten wir eine Methode, quadratische Kongruenzgleichungen im Fall $p \equiv_4 3$ effizient zu lösen. Im Fall $p \equiv_4 1$ ist dagegen kein effizienter deterministischer Lösungsalgorithmus bekannt. Allerdings gibt es hierfür einen effizienten probabilistischen Algorithmus von Adleman, Manders und Miller (1977).

Korollar 94. *Sei $p > 2$ prim, dann besitzt die quadratische Kongruenzgleichung $x^2 \equiv_p a$ für jedes $a \in \mathsf{QR}_p$ genau zwei Lösungen. Im Fall $p \equiv_4 3$ sind dies $\pm a^k \bmod p$ (für $k = (p+1)/4$), wovon nur $a^k \bmod p$ ein quadratischer Rest ist.*

Beweis. Sei $a \in \mathsf{QR}_p$, d. h. es existiert ein $b \in \mathbb{Z}_p^*$ mit $b^2 \equiv_p a$. Mit b ist auch $-b$ eine Lösung von $x^2 \equiv_p a$, die von b verschieden ist (p ist ungerade). Nach Lagrange existieren keine weiteren Lösungen.
Sei nun $p \equiv_4 3$. Dann gilt

$$\left(\frac{-b}{p}\right) = \left(\frac{-1}{p}\right) \cdot \left(\frac{b}{p}\right) = -\left(\frac{b}{p}\right)$$

nach Korollar 93. Demnach ist genau eine der beiden Lösungen $\pm b$ ein quadratischer Rest. Schließlich liefert Eulers Kriterium die Kongruenz $a^{(p-1)/2} \equiv_p 1$. Daher folgt für $k = (p+1)/4$ die Kongruenz

$$(a^k)^2 = a^{(p+1)/2} = a^{(p-1)/2} \cdot a \equiv_p a.$$

Da mit a auch $a^k \bmod p$ ein quadratischer Rest ist, ist $-a^k \bmod p$ ein quadratischer Nichtrest. □

Satz 95. *Sei $n = pq$ für Primzahlen p, q mit $p \equiv_4 q \equiv 3$. Dann besitzt die quadratische Kongruenz $x^2 \equiv_n a$ für jedes $a \in \mathsf{QR}_n$ genau vier Lösungen, wovon genau eine ein quadratischer Rest ist.*

Beweis. Mit $x^2 \equiv_n a$ besitzen wegen $n = pq$ auch die beiden quadratischen Kongruenzen $x^2 \equiv_p a$ und $x^2 \equiv_q a$ Lösungen, und zwar jeweils genau zwei (siehe Korollar 94): $y_1 = a^{(p+1)/4} \bmod p \in \mathsf{QR}_p$, $y_2 = -a^{(p+1)/4} \bmod p \in \mathsf{QNR}_p$, $z_1 = a^{(q+1)/4} \bmod q \in \mathsf{QR}_q$ und $z_2 = -a^{(q+1)/4} \bmod q \in \mathsf{QNR}_q$. Mit dem Chinesischen Restsatz können wir vier verschiedene Lösungen $x_{i,j}$, $1 \le i, j \le 2$ mit

$$\begin{aligned} x &\equiv_p y_i \\ x &\equiv_q z_j \end{aligned}$$

bestimmen. Es können aber auch nicht mehr als diese vier Lösungen existieren, da sich daraus für mindestens eine der beiden Kongruenzen $x^2 \equiv_p a$ und $x^2 \equiv_q a$ mehr als zwei Lösungen ableiten ließen.
Wegen

$$\begin{aligned} x \in \mathsf{QR}_n &\Rightarrow \exists u : u^2 \equiv_n x \\ &\Rightarrow \exists u : u^2 \equiv_p x \equiv_q u^2 \\ &\Rightarrow x \bmod p \in \mathsf{QR}_p \wedge x \bmod q \in \mathsf{QR}_q \end{aligned}$$

können $x_{1,2}, x_{2,1}, x_{2,2}$ keine quadratischen Reste modulo n sein.

Weiterhin gibt es Zahlen $l \in \mathbb{Z}_p^*$, $k \in \mathbb{Z}_q^*$ mit $l^2 \equiv_p y_1$ und $k^2 \equiv_q z_1$. Sei $m \in \mathbb{Z}_n^*$ eine Lösung für das System

$$x \equiv_p l$$
$$x \equiv_q k$$

Dann folgt

$$x_{1,1} \equiv_p y_1 \equiv_p l^2 \equiv_p m^2 \quad \text{und} \quad x_{1,1} \equiv_q z_1 \equiv_q k^2 \equiv_q m^2$$

und daher $x_{1,1} \equiv_n m^2$. Also ist $x_{1,1}$ ein quadratischer Rest modulo n. □

Als weitere zahlentheoretische Funktion mit für die Kryptografie wichtigen Eigenschaften erhalten wir somit die Quadratfunktion $x^2 : \mathsf{QR}_n \to \mathsf{QR}_n$, die nach vorigem Satz bijektiv ist (falls $n = pq$ für Primzahlen p, q mit $p \equiv_4 q \equiv 3$). Ihre Umkehrfunktion $x \mapsto \sqrt{x}$ heißt **diskrete Wurzelfunktion**, und kann (nur) bei Kenntnis der Primfaktoren p und q von n effizient berechnet werden. Es ist bekannt, dass die Berechnung dieser Funktion äquivalent zur Faktorisierung von n ist. Ohne Kenntnis der Faktoren von n ist dagegen nicht einmal ein effizientes Verfahren bekannt, mit dem man für ein gegebenes $a \in \mathbb{Z}_n^*$ entscheiden kann, ob $a \in \mathsf{QR}_n$ ist oder nicht.

4.3 Der BBS-Generator

Der BBS-Pseudozufallsgenerator von Blum, Blum und Shub 1986 verwendet die Quadratfunktion

$$x^2 : \mathsf{QR}_n \mapsto \mathsf{QR}_n,$$

mit $n = p \cdot q$ für p, q prim und $p \equiv_4 q \equiv_4 3$. Seine Sicherheit lässt sich unter der Voraussetzung beweisen, dass ohne Kenntnis der Faktoren p, q für fast alle $y \in \mathsf{QR}_n$ das niederwertigste Bit von $\sqrt{y}$ nur mit vernachlässigbarem Vorteil richtig geraten werden kann. Dies wiederum ist äquivalent dazu, dass sich n nicht effizient faktorisieren lässt.

Als Keim wird eine zufällig aus $\mathbb{Z}_n^*$ gewählte Zahl x_0 verwendet. Daraus werden der Reihe nach Zahlen $x_i \in \mathsf{QR}_n$ durch Quadrieren berechnet, deren Paritäten die Bits der Ausgabefolge bilden.

Algorithmus $\mathsf{BBS}_{n,l}(x_0)$

```
1 for i := 1 to l do
2     x_i := x_{i-1}^2 mod n
3     b_i := x_i mod 2
4 ouput(b_1, ... b_l)
```

Beispiel 96. *Wählen wir z. B. die Primzahlen $p = 11$, $q = 19$, also $n = 209$, und als Keim $x_0 = 20$, so erhalten wir die Pseudo-Zufallsbitfolge* $\mathrm{BBS}_{209}(20) = 11001100\ldots$

i	0	1	2	3	4	5	6	7	8	...
x_i	20	191	115	58	20	191	115	58	20	...
b_i	0	1	1	0	0	1	1	0	0	...

Man kann zeigen, dass sich aus jedem effizienten Unterscheider D für den BBS-Generator $\mathsf{BBS}_{n,l}$ ein effizienter probabilistischer Algorithmus zur Faktorisierung von n gewinnen

lässt. Da die Laufzeit von D in Abhängigkeit von l gemessen wird, sollte l polynomiell in der Länge von n beschränkt sein, um einen effizienten probabilistischen Faktorisierungsalgorithmus zu erhalten. In einem ersten Schritt wird D benutzt, um für eine Zahl $a \in \mathbb{Z}_n^*$ zu entscheiden, ob sie ein quadratischer Rest ist oder nicht.

4.4 Quadratische Pseudoreste

Wir erweitern nun das Legendre-Symbol zum *Jacobi-Symbol*.

Definition 97 (Jacobi-Symbol)**.** *Das* Jacobi-Symbol *ist für alle a und alle ungeraden $m > 3$ durch*

$$\mathcal{J}(a,m) = \left(\frac{a}{m}\right) = \left(\frac{a}{p_1}\right)^{e_1} \cdots \left(\frac{a}{p_r}\right)^{e_r}$$

*definiert, wobei $p_1^{e_1} \cdots p_r^{e_r}$ die Primfaktorzerlegung von m ist. Ist zwar $\left(\frac{a}{m}\right) = 1$, aber $a \in \mathsf{QNR}_m$ kein quadratischer Rest modulo m, so heißt a **quadratischer Pseudorest** modulo m (kurz: $a \in \widetilde{\mathsf{QR}}_m$).*

Man beachte, dass im Gegensatz zum Legendre-Symbol die Eigenschaft $\left(\frac{a}{m}\right) = 1$ für ein $a \in \mathbb{Z}_m^*$ nicht unbedingt mit $a \in \mathsf{QR}_m$ gleichbedeutend ist. Zum Beispiel gibt es in $\mathbb{Z}_n^*$ ($n = p \cdot q$ für Primzahlen p und q mit $p \equiv_4 q \equiv_4 3$) wie wir gesehen haben, genau $\varphi(n)/4$ quadratische Reste und $3\varphi(n)/4$ quadratische Nichtreste, wogegen nur für die Hälfte aller $a \in \mathbb{Z}_n^*$ die Gleichung $\left(\frac{a}{m}\right) = -1$ gilt. Folglich gibt es in diesem Fall genau so viele quadratische Reste wie quadratische Pseudoreste.

Allerdings überträgt sich die in Teil 2 von Korollar 93 festgehaltene Eigenschaft des Legendre-Symbols auf das Jacobi-Symbol. Interessanterweise ist das Jacobi-Symbol auch ohne Kenntnis der Primfaktorzerlegung des Moduls effizient berechenbar.

Sei $n = pq$ das Produkt zweier Primzahlen p, q mit der Eigenschaft $p \equiv_4 q \equiv_4 3$. Es ist bekannt, dass die Berechnung der Umkehrfunktion $\sqrt{x} : \mathsf{QR}_n \to \mathsf{QR}_n$ der Quadratfunktion $x^2 : \mathsf{QR}_n \to \mathsf{QR}_n$ äquivalent zur Faktorisierung von n ist. Folglich ist die diskrete Wurzelfunktion $\sqrt{x}$ schwer zu berechnen, falls die Faktorisierung von n schwer ist. Man nimmt sogar an, dass bereits das Entscheidungsproblem, ob eine gegebene Zahl $x \in \mathbb{Z}_n^*$ ein quadratischer Rest ist, schwierig ist. Da dieses Problem für Eingaben x mit Jacobisymbol $\left(\frac{x}{n}\right) = -1$ trivial ist, schließt man sie üblicherweise von der Betrachtung aus.

Quadratische-Reste-Problem (QR-Problem):

> *Gegeben:* Zahlen n und $x \in \mathbb{Z}_n^*$ mit Jacobisymbol $\left(\frac{x}{n}\right) = 1$ (d.h. $x \in \mathsf{QR}_n \cup \widetilde{\mathsf{QR}}_n$), wobei n das Produkt zweier unbekannter Primzahlen ist.
>
> *Gefragt:* Ist $x \in \mathsf{QR}_n$?

Beim QR-Problem geht es also darum, quadratische Reste von quadratischen Pseudoresten zu unterscheiden.

4.5 Sicherheit des BBS-Generators

Wir zeigen nun, dass sich aus jedem effizienten Unterscheider für den BBS-Generator ein effizienter probabilistischer Algorithmus für das QR-Problem gewinnen lässt. Im Umkehrschluss bedeutet dies, dass der BBS-Generator sicher ist, falls das QR-Problem hart ist.

Sei also D ein effizienter ε-Unterscheider für den Generator $\mathsf{BBS}_{n,l}$. Dann ex. ein effizienter (ε/l)-PBP P für $\mathsf{BBS}_{n,l}$. Der folgende Satz zeigt, wie sich aus einem δ-PBP P für $\mathsf{BBS}_{n,l}$ ein Entscheidungsalgorithmus für das QR-Problem gewinnen lässt, der für eine zufällige Eingabe $x \in \mathsf{QR}_n \cup \widetilde{\mathsf{QR}}_n$ eine Korrektheitswahrscheinlichkeit $\geq 1/2 + \delta$ hat.

Satz 98. *Falls es einen effizienten δ-PBP P für $\mathsf{BBS}_{n,l}$ gibt, so lässt sich für eine zufällig aus $\mathsf{QR}_n \cup \widetilde{\mathsf{QR}}_n$ gewählte Eingabe x mit Wahrscheinlichkeit $\geq 1/2 + \delta$ entscheiden, ob $x \in \mathsf{QR}_n$ ist.*

Beweis. Betrachte folgenden Entscheidungsalgorithmus.

Algorithmus $\mathtt{QR\text{-}Test}(x, n)$

```
1  wähle i ∈_R {1, ..., l}
2  z_i := x
3  for j := i + 1 to l do
4     z_j := z_{j-1}^2 mod n
5     b_j := z_j mod 2
6  b_i := P(b_{i+1} ··· b_l, 1^l)
7  if x ≡_2 b_i then ouput(1) else ouput(0)
```

Die Aussage des Satzes folgt unmittelbar aus folgender Behauptung.

Behauptung. $\Pr_{x \in_R \mathsf{QR}_n \cup \widetilde{\mathsf{QR}}_n}[\mathtt{QR\text{-}Test}(x, n) = 1 \Leftrightarrow x \in \mathsf{QR}_n] \geq 1/2 + \delta$.

Wird x zufällig aus $\mathsf{QR}_n \cup \widetilde{\mathsf{QR}}_n$ gewählt, so ist $x_{i+1} = x^2 \bmod n$ ein zufälliger quadratischer Rest in QR_n, d.h. die Eingabe für den PBP P sind $l - i$ konsekutive Bits einer mit $\mathsf{BBS}_{n,l}$ generierten Pseudozufallsfolge (man überlegt sich leicht, dass die Verteilung dieser Bitfolge nicht vom Index des Startbits abhängt, da alle x_i, $i \geq 1$, auf QR_n gleichverteilt sind). Daher gibt P mit Wahrscheinlichkeit $1/2+\delta$ die Parität der diskreten Wurzel $\sqrt{x_{i+1}} \bmod n$ aus. Da $x \in \mathsf{QR}_n \cup \widetilde{\mathsf{QR}}_n$ ist, gilt $\sqrt{x_{i+1}} \bmod n \in \{x, n - x\}$. Zudem hat $\sqrt{x_{i+1}} \bmod n$ wegen $x \not\equiv_2 n - x$ genau dann die gleiche Parität wie x, wenn $x = \sqrt{x_{i+1}} \bmod n$ ist. Da dies wiederum mit $x \in \mathsf{QR}_n$ äquivalent ist, folgt die Behauptung. □

Als nächstes zeigen wir, wie sich `QR-Test` in einen Algorithmus verwandeln lässt, der jede Eingabe $x \in \mathsf{QR}_n \cup \widetilde{\mathsf{QR}}_n$ mit Wahrscheinlichkeit $\geq 1/2 + \delta$ korrekt entscheidet.

Satz 99. *Falls es einen effizienten Algorithmus A gibt, der für eine zufällig aus $\mathsf{QR}_n \cup \widetilde{\mathsf{QR}}_n$ gewählte Eingabe x mit Wahrscheinlichkeit $\geq 1/2 + \delta$ entscheidet, ob $x \in \mathsf{QR}_n$ ist, so ex. auch ein effizienter Algorithmus A', der für jede Eingabe $x \in \mathsf{QR}_n \cup \widetilde{\mathsf{QR}}_n$ die Zugehörigkeit von x zu QR_n mit Wahrscheinlichkeit $\geq 1/2 + \delta$ korrekt entscheidet.*

Beweis. Betrachte folgenden Entscheidungsalgorithmus.

Algorithmus $A'(x, n)$

```
1  wähle zufällig eine Zahl z ∈ Z_n^*
2  wähle zufällig ein Bit b ∈ {0, 1}
3  x' := (-1)^b z^2 x mod n
4  ouput A(x', n) ⊕ b
```

Für jede Eingabe $x \in \mathsf{QR}_n \cup \widetilde{\mathsf{QR}}_n$ ist x' eine Zufallszahl in $\mathsf{QR}_n \cup \widetilde{\mathsf{QR}}_n$. Da $-1 \in \widetilde{\mathsf{QR}}_n$ ist, ist die Funktion $x \mapsto -x \bmod n$ eine Bijektion zwischen QR_n und $\widetilde{\mathsf{QR}}_n$, d.h. die Ausgabe von $A(x, n)$ ist genau dann korrekt, wenn die Ausgabe von $A(x', n)$ korrekt ist. □

Schließlich zeigen wir noch, wie sich die Fehlerwahrscheinlichkeit von A' exponentiell klein machen lässt. Hierzu benötigen wir das folgende Lemma.

Lemma 100. *Sei E ein Ereignis, das mit Wahrscheinlichkeit $1/2 - \delta$, $\delta > 0$, auftritt. Dann ist die Wahrscheinlichkeit, dass sich E bei $m = 2t+1$ unabhängigen Wiederholungen mehr als t-mal ereignet, höchstens $1/2(1 - 4\delta^2)^t$.*

Beweis. Für $i = 1, \ldots, m$ sei X_i die Indikatorvariable

$$X_i = \begin{cases} 1, & \text{Ereignis } E \text{ tritt beim } i\text{-ten Versuch ein,} \\ 0, & \text{sonst} \end{cases}$$

und X sei die Zufallsvariable $X = \sum_{i=1}^{m} X_i$. Dann ist X binomial verteilt mit Parametern m und $p = 1/2 - \delta$. Folglich gilt für $i > m/2$,

$$\begin{aligned} \Pr[X = i] &= \binom{m}{i} (1/2 - \delta)^i (1/2 + \delta)^{m-i} \\ &= \binom{m}{i} (1/2 - \delta)^{m/2} (1/2 + \delta)^{m/2} \left(\frac{1/2 - \delta}{1/2 + \delta} \right)^{i - m/2} \\ &\leq \binom{m}{i} \underbrace{(1/2 - \delta)^{m/2} (1/2 + \delta)^{m/2}}_{\left(1/4 - \delta^2\right)^{m/2}}. \end{aligned}$$

Wegen

$$\sum_{i=t+1}^{m} \binom{m}{i} = 2^{m-1} = \frac{4^{m/2}}{2}$$

erhalten wir somit

$$\begin{aligned} \sum_{i=t+1}^{m} \Pr[X = i] &\leq \left(1/4 - \delta^2\right)^{m/2} \sum_{i=t+1}^{m} \binom{m}{i} = \frac{(1 - 4\delta^2)^{m/2}}{2} \\ &\leq \frac{(1 - 4\delta^2)^t}{2}. \end{aligned}$$

□

Falls wir also A' $m = 2t + 1$-mal ausführen und einen Mehrheitsentscheid treffen, so reduziert sich die Fehlerwahrscheinlichkeit von $1/2 - \delta$ auf $1/2(1 - 4\delta^2)^t < e^{-4\delta^2 t}$. Wählen wir beispielsweise $t = s/4\delta^2$, so wird diese kleiner als 2^{-s}.