Inhaltsverzeichnis

1 Grundlegende Konzepte und mathematisches Wissen

XOR-Operation

Für die XOR-Operation, auch ⊕-Operation genannt, werden zwei n lange Binärzahlen $b1 = \|_{i=0}^{n} b_i$ und $b2 = \|_{i=0}^{n} b_i$ verrechnet, indem die einzelnen Bits nach folgender Wahrheitstafel verrechnet werden:

$b1_i$	$b2_i$	$b1_i \oplus b2_i$
0	0	0
0	1	1
1	0	1
1	1	0

Beispiel 1.1. (XOR-Operator)

$b1$	=	10001000
$b2$	=	11110000
result $b_1 \oplus b_2$	=	01111000

Wahrscheinlichkeitsraum

Ein Wahrscheinlichkeitsraum besteht aus drei Komponenten:
- Ω ist der Ergebnisraum (sample space)
- Σ ist die Menge an (Zwischen-)Ereignissen (event set)
- $P : \Sigma \to [0, 1]$ ist die Wahrscheinlichkeitsfunktion

Bemerkung 1.2 (Begrifflichkeiten).
- Die Wahrscheinlichkeitsfunktion kann auch Pr[...] genannt werden kann (Grund: Im Kontext von P und NP ist die Doppelbelegung von P doof). Es gibt keinen Unterschied zwischen beiden Notationen
- **Zufallsvariable:** Abstrakt: Eine ZV filtert die Elemente aus Ω, welchen schließlich ein Wert $\neq 0$ zugewiesen werden kann. Verständlich: Eine ZV ist eine Teilmenge von Ω. Σ bildet die Menge aller möglichen Zufallsvariablen
- **Ereignis:** Ein Element aus Ω

Beispiel 1.3. (Experiment mit einem d4-DnD-Würfel)

$\Omega = \{1, 2, 3, 4\}$

$\Sigma = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}, \Omega\}$

$P(\{1\}) = P(\{2\}) = P(\{3\}) = P(\{4\}) = 0.25$

$P(\{1, 2\}) = P(\{1, 3\}) = P(\{1, 4\}) = ... = P(\{3, 4\}) = 0.5$

$P(\{1, 2\} \wedge \{1, 4\}) = P(\{1, 2\} \wedge \{1, 3\}) = 0.25$

$P(\{1, 3\} \mid \{1, 2, 4\}) = P(\{2, 3\} \mid \{1, 3, 4\}) = 0.25/0.75 = 0.33...$

Anschaulich stellt die Zufallesvariable $\mathcal{X} = \{2, 4\}$ die Frage nach allen gerade Ergebnissen. Die Wahrscheinlichkeit für ein Event aus $x \in \mathcal{X}$ wird als $P(x = 2) = 0.25$ geschrieben.

Formelsammlung Stocharschtik

Für zwei **unabhängige** Events gilt:

$$P(a \wedge b) = P(a \wedge b) = P(a) \cdot P(b)$$

$$P(a \mid b) = P(a)$$

Generell gilt:

$$P(a \vee b) = P(a) + P(b) + P(a \wedge b)$$

Bayes Theorem:

$$P(a \mid b) = \frac{P(a) \cdot P(b \mid a)}{P(b)}$$

Union Bound:

$$P(E_1 \vee ... \vee E_n) \leq P(E_1) + ... + P(E_n)$$

Für zwei **abhängige** Events gilt:

$$P(a \mid b) = \frac{P(a \wedge b)}{P(b)}$$

Bayesian Rules für **abhängige** Events:
- **Product Rule:**

$$P(a \wedge b) = P(a \mid b) \cdot P(b)$$

- **Chain Rule:**

$$P(x_1, ..., x_n) = P(x_n \mid x_{n-1}, ..., x_1) \cdot P(x_{n-1} \mid x_{n-2}, ..., x_1) \cdot ...$$

- **Marginalization:**

$$P(x) = \sum_{y \in Y} P(x, y)$$

$$P(a) = P(a \mid b) \cdot P(b) + P(a \mid \neg b) \cdot P(\neg b)$$

Teil I
Symmetrische Verschlüsselung

2 Grundlegende Begrifflichkeiten und Definitionen

Definition 2.1 (Grundschemata einer Private Key Encryption)

Ein **Private Key Encryption** Schemata $\Pi = (Gen, Enc, Dec)$ wird über einen Klartextraum $\mathcal{M} = \{0,1\}^*$ anhand von drei Algorithmen definiert:

- *Gen*: **Schlüsselgenerierung** (Key-Generation) erzeugt als Output einen Schlüssel k anhand einer gewissen Wahrscheinlichkeitsverteilung
- $Enc_k(m)$: **Verschlüsselung** (Encryption) nimmt als Input den Schlüssel k und die Klartextnachricht $m \in \mathcal{M}$ und erzeugt daraus als Output den Chiffretext $c \in C$
- $Dec_k(c)$: **Entschlüsselung** (Decryption) nimmt als Input den Schlüssel k und den Chiffretext c und erzeugt daraus als Output den ursprünglichen Klartext m

Bemerkung 2.2.

- Für diese Schemata gilt immer die **perfekte Korrektheit** $Dec_k(Enc_k(m)) = m$
- $P(k)$ gibt die Wahrscheinlichkeit an, dass Gen den Schlüssel k erzeugt
- $P(m)$ gibt die Wahrscheinlichkeit an, dass die Nachricht m verwendet wird
- $P(c)$ gibt die Wahrscheinlichkeit an, dass der erzeugte Ciphertext c ist

Definition 2.3 (Angriffsszenarien)

- **Ciphertext-Only Attack:** Der Angreifer sieht nur das Chiphrat
- **Known-Plaintext Attack:** Dem Angreifer sind gewisse Klartext Ciphertext Paare bekannt
- **Chosen-Plaintext Attack:** Der Angreifer kann jeden Klartext entschlüsseln lassen
- **Chosen-Ciphertext Attack:** Der Angreifer kann gewisse Chiphretexte entschlüsseln lassen

Definition 2.4 (Principles of Modern Cryptography)

Principle 1: Formal Definition

- Definition des Systems, dessen Sicherheit, Ziele, der möglichen Angriffsszenarien und der Möglichkeiten aller Parteien, die das System nutzen

Principle 2: Precise Assumptions

- Die Annahmen zur Sicherheit eines Systems müssen präzise und nachvollziehbar sein. Beispielsweise eignen sich mathematische Probleme wie der diskrete Logarithmus oder Faktorisierung

Principle 3: Proof of security

- Der Beweis zeigt, dass das System anhand der Annahmen unabstreitbar sicher ist

2.1 Historische Verschlüsselungen

Definition 2.5 (Caesar Cipher)

- **GEN:** Der Schlüssel kann eine beliebige ganze Zahl $k \in \mathbb{Z}$ sein
- **ENC:** $c_i = m_i + k \mod 26$
- **DEC:** $m_i = c_i - k \mod 26$

Bemerkung 2.6.

- Sehr einfach mit Brute Force zu knacken
- Daraus kann man ableiten, dass der Schlüsselraum so groß sein muss, dass man diesen nicht einfach durch ausprobieren durchsuchen kann

Mono-Alphabetic Substitution Cipher

- **GEN:** Der Schlüssel ist eine zufällige Substitution $f(x) = y$ für jeden Buchstaben x
- **ENC:** $c_i = f(m_i)$
- **DEC:** $m_i = f^{-1}(c_i)$

Beispiel 2.7. (Schlüssel einer Substitutions Cipher)

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
X	E	U	A	D	N	B	K	V	M	R	O	C	Q	F	S	Y	H	W	G	L	Z	I	J	P	T

Bemerkung 2.8.
- Ca. $26! = 2^{88}$ verschiedene Schlüssel
- Mittels Häufigkeitsanalyse entschlüsselbar

Exkurs: Häufigkeitsanalyse

In jeder Sprache kommen gewisse Buchstaben häufiger vor als andere. Kennt man diese Verteilung, kann man bei einer Monoalphabetischen Verschlüsselung die Häufigkeit aller Buchstaben ermitteln und diese mit der Buchstabenverteilung der Ausgangssprache vergleichen. Damit können Informationen über den Schlüssel gewonnen werden.

Vigenere Cipher

- Verschlüsselungsverfahren ist ähnlich zur Monoalphabetischen Substitution
- Schlüssel ist ein Wort, welches solange wiederholt wird, bis die Länge des Klartextes erreicht ist
- War hunderte Jahre sicher, heutzutage aber nicht mehr (Kasiski Angriff)

Beispiel 2.9. (Vigenere Verschlüsselung)

Plaintext:	tellhimaboutme
Key:	cafecafecafeca
Ciphertext:	WFRQKJSFEPAYPF

2.2 Perfekte Sicherheit

Definition 2.10 (Perfekte Sicherheit)

Ein Verschlüsselungssystem mit Klartexten $m \in \mathcal{M}$ und Chiffretexten $c \in C$ ist genau dann sicher, wenn Klartext und Chiffretext voneinander unabhängig sind (mit $P(c) > 0$):

$$P(m \mid c) = P(m)$$

Daraus kann man folgern:

$$P(c \mid m) = P(c)$$

Bemerkung 2.11.
- Nach Claude Shannon: Das Verschlüsselungsverfahren ist perfekt sicher, genau dann, wenn die Wahrscheinlichkeitsverteilung auf dem Schlüsselraum die Gleichverteilung ist und es zu jedem Klartext genau einen Schlüssel und Chiffretext gibt.
- Daraus kann $P(m \wedge c) = P(m) \cdot P(c)$ gefolgert werden
- Zudem kann $P(c \mid m_0) = P(c \mid m_1)$
- Perfekte Sicherheit ist unpraktisch, da die Größe des Schlüssels ineffizient ist

Definition 2.12 (One Time Pad)

Sei der Schlüsseltext $c \in C$ und die Klartextnachricht $m \in \mathcal{M}$ und beides von der Länge n:
- **Gen:** Zufälliger Schlüssel: $k \in \mathcal{K} = \{0,1\}^n$
- **Enc:** $c = m \oplus k$
- **Dec:** $m = c \oplus k$

Beweis 2.13.
Man weiß, dass $P(k) = 2^{-n}$ und damit $P(k = m\ XOR\ c) = 2^{-n}$ gilt. Damit kann man folgern:

$$P(c) = P(c = m\ XOR\ k) = P(k = m\ XOR\ c) = 2^{-n}$$

$$P(c \mid m) = P(c = m\ XOR\ k) = P(k = m\ XOR\ c) = 2^{-n}$$

(Man kann nach dem gleichen Prinzip folgern, dass $P(m) = 2^{-n}$ gilt. Dies würde den Beweis aber seiner Eleganz berauben). Anhand dieser Erkentnisse kann schließlich, unter Verwendung von Bayes'Theorem, die Gleichheit des Ausgangstheorems abgeleitet werden:

$$P(m \mid c) = \frac{P(c \mid m) \cdot P(m)}{P(c)} = \frac{2^{-n} \cdot P(m)}{2^{-n}} = P(m)$$

□

Bemerkung 2.14.

- One Time Pad ist perfekt korrekt: $Dec_k(Enc_k(m)) = Dec_k(k \oplus m) = k \oplus k \oplus m = m$.
- One Time Pad erfüllt die Bedingungen für Perfekte Sicherheit

Definition 2.15 (Two Time Pad)

Seien die Schlüsseltexte $c_1, c_2 \in C$ und die Klartextnachrichten $m_1, m_2 \in \mathcal{M}$ von Länge n:

- **Gen:** Zufälliger Schlüssel: $k \in \mathcal{K} = \{0,1\}^n$
- **Enc:** $(c_1, c_2) = (m_1, m_2) \oplus k$
- **Dec:** $(m_1, m_2) = (c_1, c_2) \oplus k$

Bemerkung 2.16.

- **Unsicher:** Für eine der beiden Nachrichten gilt nicht mehr die stochastische Unabhängigkeit
- Kennt man beide Nachrichten und einen Teil des Klartextes, kann man alles entschlüsseln

Definition 2.17 (Perfekte Ununterscheidbarkeit (Indistinguishability))

Ein Kryptosystem Π = (Gen, Enc, Dec) ist perfekt Ununterscheidbar, wenn für jeden Angreifer $\mathcal{A}$ gilt:

$$P[PrivK^{eav}_{\mathcal{A},\Pi} = 1] = \frac{1}{2}$$

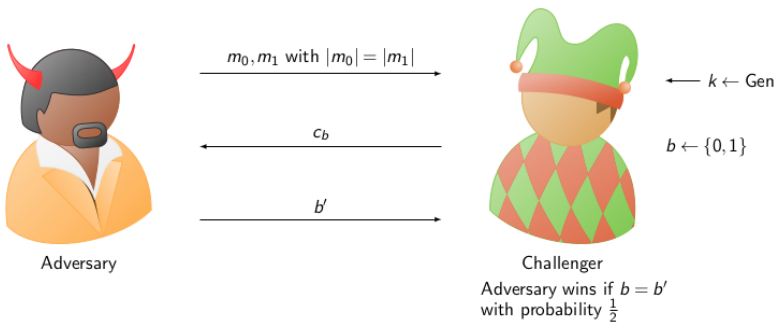
Experiment $PrivK^{eav}_{\mathcal{A},\Pi}$

Sei Π = (Gen, Enc, Dec) ein Kryptosystem mit Klartextraum $\mathcal{M}$. Sei $\mathcal{A}$ der Angreifer/Gegenspieler, welcher formell als Algorithmus angesehen werden kann. Das Experiment findet zwischen dem Angreifer und einem imaginären Herausforderer C statt:

- $\mathcal{A}$ erzeugt zwei Nachrichten $m_0, m_1 \in \mathcal{M}$ und gibt diese C
- C erzeugt einen zufälligen Schlüssel und ein zufälliges Bit $b \in \{0,1\}$. Dann erzeugt C den Chiffretext $Enc(m_b) = c$ und schickt diesen $\mathcal{A}$
- $\mathcal{A}$ verarbeitet c und schickt $b' \in \{0,1\}$ an C
- Der Ausgang des Experimentes ist 1, wenn $b' = b$ gilt. In diesem Fall hat $\mathcal{A}$ gewonnen und man schreibt $PrivK^{eav}_{\mathcal{A},\Pi} = 1$

Alternative Darstellung:

$PrivK^{eav}_{\mathcal{A},\Pi}$
$(m_0, m_1) \leftarrow \mathcal{A}, m_0, m_1 \in \mathcal{M}$
$k \leftarrow Gen$
$b \leftarrow \{0,1\}$
$c \leftarrow Enc(m_b)$
$b' \leftarrow \mathcal{A}(c)$
if $b' = b$ return 1 else return 0



Bemerkung 2.18.

- Theorem und Experiment gehören zusammen und bauen aufeinander auf
- Es gilt: Jeglicher Angreifer in diesem Experiment nicht besser sein kann, als ein Angreifer der rät
- Es gilt: Perfekte Sicherheit ⇔ Perfekte Ununterscheidbarkeit
- Zur Vereinfachung wird angenommen, dass beiden Nachrichten gleich lang sind, da man ansonsten einen Spezialfall für Padding benötigt

Beispiel 2.19. (Ununterscheidbarkeit und Vigenere-Chiffre)

Anhand des aufgezeigten Experimentes kann nun Beispielsweise bewiesen werden, dass Vigenere unsicher ist. Dafür verwendet der Angreifer die Texte $m_0 = aaaaa...$ und $m_1 = adfiwofn...$, also ein Text mit beliebig vielen gleichen Buchstaben und ein Text mit beliebig vielen zufälligen Buchstaben. Der Chiffretext von m_0 sollte also, ähnlich zum Schlüssel bei Vigenere, Zyklenweise identisch sein, wohingegen m_1 weiterhin komplett zufällig ist. Dadurch kann der Angreifer die Chiffretexte auseinanderhalten und die Ununterscheidbarkeit ist nicht gegeben.

---

3 Grundlagen berechenbarer kryptographischer Sicherheit

3.1 Präzisere Formulierung symmetrischer Verfahren

Definition 3.1 (Grundlegende Begrifflichkeiten und Abkürzungen)

- Ein **polynomialzeit Algorithmus** kann in polynomieller Zeit berechnet werden, d.h. die Funktion der Laufzeit abhängig von der Inputgröße wächst nicht wesentlicher schneller als ein Polynom
- Ein **probabilistischer polynomialzeit Algorithmus** berechnet für jede zufällige Eingabe in polynomieller Zeit das Ergebnis

Definition 3.2 (Grundschemata Berechenbarer Symmetrische Verschlüsselung)

Ein **Private Key Encryption** Schemata $\Pi = (Gen, Enc, Dec)$ wird über einen Klartextraum $\mathcal{M} = \{0,1\}^*$ anhand von drei probablistischen polynomialzeit Algorithmen definiert:

- $k \leftarrow Gen(1^n)$: **Schlüsselgenerierung** (Key-Generation) erzeugt als Output einen Schlüssel k anhand einer gewissen Wahrscheinlichkeitsverteilung aus dem Schlüsselraum $\mathcal{K}$ und kann dafür einen Inputparameter 1^n verwenden
- $c \leftarrow Enc_k(m)$: **Verschlüsselung** (Encryption) nimmt als Input den Schlüssel k und die Klartextnachricht $m \in \mathcal{M}$ und erzeugt daraus als Output den Chiffretext $c \in C$
- $m \leftarrow Dec_k(c)$: **Entschlüsselung** (Decryption) nimmt als Input den Schlüssel k und den Chiffretext c und erzeugt daraus als Output den ursprünglichen Klartext m

Bemerkung 3.3.

- Für diese Schemata gilt immer die **perfekte Korrektheit** $Dec_k(Enc_k(m)) = m$
- Der Unterschied zur vorhergehenden Definition ist, dass nun auch $P = / \neq NP$ miteinbezogen wird

3.2 Vernachlässigbarkeit

Definition 3.4 (Vernachlässigbare Funktionen)

Eine Funktion f ist **vernachlässigbar** wenn es zu jedem Polynom p ein N gibt, sodass für alle Zahlen n mit $n > N$ gilt:

$$f(n) < \frac{1}{p(n)}$$

Beispiel 3.5. (Vernachlässigbare Funktionen)

(a) $\frac{1}{2^x}$: Vernachlässigbar, da 2^x exponentiell ist

(b) $\frac{1}{x^{100}}$: Nicht Vernachlässigbar, da x^{100} nicht exponentiell ist. In anderen Worten: Sei $p(x) = x^{101}$. Setzt man dies ins Theorem ein so erhält man:

$$f(x) = \frac{1}{x^{100}} \not< \frac{1}{x^{101}}$$

(c) $h(x)$ mit $h(x) < f(x) \forall x$. Offensichtlich ist das Vernachlässigbar:

$$h(x) < f(x) < \frac{1}{p(x)} \Rightarrow h(x) < \frac{1}{p(x)}$$

(d) $\frac{1}{2^x} + \frac{1}{7^x}$: Vernachlässigbar, da Addition vernachlässigbarer Funktionen

(e) $\frac{1}{2^x} \cdot \frac{1}{x^7}$: Vernachlässigbar, da Verknüpfung einer vernachlässigbaren Funktion mit positivem Polynom.

(f) $\frac{f(x)}{g(x)}$ Nicht Vernachlässigbar. Gilt nämlich $f(x) = g(x)$ so ist das Ergebnis immer 1, also eine Konstante.

(g) 2^{-100}: Nicht Vernachlässigbar, da es ein Konstanter Wert ist.

Bemerkung 3.6.

- Vernachlässigbare Funktionen werden auch als $negl(n)$ geschrieben
- Es wurde sich hier für den analytischen Ansatz, statt den stochastischen Ansatz, entschieden, da man damit besser Unterschiede bei verschieden leistungsstarken Rechenmaschinen abdecken kann

- Jede Funktion die exponentiell gegen null geht ist vernachlässigbar: Gilt $f(n) = 1/f'(n)$ und ist $f'(n)$ exponentiell, so ist $f(x)$ vernachlässigbar: Polynome wachsen nämlich nie schneller als exponentielle Funktionen
- Konstante Werte sind nicht vernachlässigbar: $p(n) = n$ ist ein Gegenbeispiel anhand des Theorems
- Seien $f_1(x)$ und $f_2(x)$ vernachlässigbare Funktionen, dann ist $f_3(x) = f_1(x) + f_2(x)$ vernachlässigbar: Es gibt ein $x > N_{2p}$ sodass $f(x) < \frac{1}{2p(x)}$ und ein $x > N'_{2p}$, sodass $g(x) < \frac{1}{2p(x)}$ gilt. Daraus folgt $f(x) + g(x) < \frac{1}{p(x)}$ für alle $x > max\{N_{2p}, N'_{2p}\}$
- Sei $f_1(x)$ eine vernachlässigbare Funktion und $p(x)$ ein positives Polynom, dann ist $f_2(x) = p(x) \cdot f_1(x)$ vernachlässigbar: Sei $q(x)$ ein Polynom. Dann existiert ein N_{pq}, sodass $f(x) < \frac{1}{p(x)q(x)}$ für alle $x > N_{pq}$ gilt. Dies kann man zu $f(x)q(x) < \frac{1}{p(x)}$ umformen.

3.3 Berechenbare Ununterscheidbarkeit

Definition 3.7 (Berechenbare Ununterscheidbarkeit (Indistinguishability))

Ein Kryptosystem Π hat eine ununterscheidbare Verschlüsselung, wenn für jeden PPT (propabilistischen polynomialzeit) Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion existiert, sodass gilt:

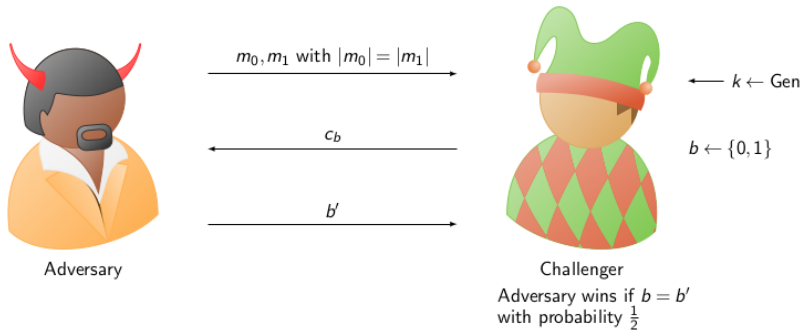
$$P[PrivK^{eav}_{\mathcal{A},\Pi} = 1] \leq \frac{1}{2} + negl(n)$$

Anmerkung: Die Wslk. wird über das Zufallsprinzip von $\mathcal{A}$ und des Experimentes berechnet.

Experiment $PrivK^{eav}_{\mathcal{A},\Pi}(n)$

Dieses Experimant ist wieder mit einem Gegenspieler $\mathcal{A}$ und einem Challenger:

$PrivK^{eav}_{\mathcal{A},\Pi}(n)$
Sei $m_0, m_1 \in \mathcal{M}, \lvert m_0\rvert = \lvert m_1\rvert$
$(m_0, m_1) \leftarrow \mathcal{A}(1^n)$
$k \leftarrow Gen(1^n)$
$b \leftarrow \{0, 1\}$
$c \leftarrow Enc(m_b)$
$b' \leftarrow \mathcal{A}(c)$
if $b' = b$ return 1 else return 0

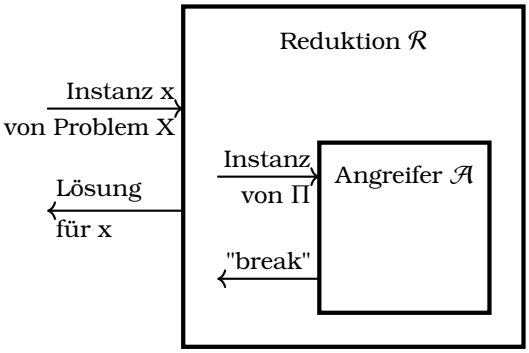


Man schreibt $PrivK^{eav}_{\mathcal{A},\Pi}(n) = 1$ wenn der Output des Experimentes 1 ist und $\mathcal{A}$ gewonnen hat.

Bemerkung 3.8.

- **Abkürzung:** ING-EAV-Single-Sicherheit (indistinguishable single encryption eavesdropper security) bzw. EAV-Single-Sicherheit
- Das ist die grundlegenste Definition von berechenbarer Sicherheit
- Damit wird die Sicherheit von Ciphertext Only Angriffen definiert, wobei der Gegenspieler nur einen einzigen Ciphertext sieht, bzw. wenn für jede Nachricht ein neuer Schlüssel erzeugt wird
- Der Unterschied von $PrivK^{eav}_{\mathcal{A},\Pi}(n)$ zu $PrivK^{eav}_{\mathcal{A},\Pi}$ ist, dass nur polynomialzeit Algorithmen und ein zusätzlicher Sicherheitsparameter betrachetet werden
- Gilt perfekte Ununterscheidbarkeit, so folgt daraus auch berechenbare Ununterscheidbarkeit

3.4 Grundidee des Reduktionsbeweises



Bemerkung 3.9 (Theoretische Vorgehensweise eines Reduktionsbeweises).

1. Ein zugrundeliegendes Problem, hier X, wird als schwierig angesehen und kann daher von einem PPT Algorithmus nur mit vernachlässigbarer Wahrscheinlichkeit effizient gelöst werden
2. Wir nehmen an, dass ein Kryptosystem Π nicht sicher ist und daher ein effizienter Angreifer $\mathcal{A}$ gegen dieses Kryptosystem existiert. Wie dieser funktioniert ist egal. Nehme man zudem an, dass $\mathcal{A}$ Π mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(n)$ brechen kann
3. Man konstruiert einen Angreifer $\mathcal{A}'$, auch Reduktion $\mathcal{R}$ genannt, welcher Problem X anhand von $\mathcal{A}$ (Black Box Prinzip) lösen kann. Dafür übergibt $\mathcal{A}'$ geschickt die Daten von x (indem man z.b. als Challenger bei $PrivK^{eav}_{\mathcal{A},\Pi}(n)$ agiert), bzw. Abwandlungen davon, an $\mathcal{A}$ und versucht aus dessen Output x zu lösen. $\mathcal{A}'$ sollte anhand des Output x mit mind. einer invertierten polynomiellen Wahrscheinlichkeit $1/p(n)$ brechen
4. Daraus kann man folgern, dass $\mathcal{A}'$ das Problem X mit Wahrscheinlichkeit $\epsilon(n)/p(n)$ löst, welche folglich auch nicht vernachlässigbar ist. Dadurch hat $\mathcal{A}'$ etwas geschafft, was wir als unmöglich angenommen haben
5. Abschließend stellt man also fest, dass es sich um einen Widerspruch zu der definierten Annahme handelt, und damit das Schemata Π sicher ist

3.5 Pseudorandomgeneratoren

Theorem 3.10 (Pseudorandomnes)

Eine Verteilung $\mathcal{D}$ über eine Menge von Zeichenketten der Länge l ist **pseudorandom**, wenn $\mathcal{D}$ ununterscheidbar zu einer gleichverteilten (echt zufälligen) Menge von Zeichenketten der Länge l ist.

Bemerkung 3.11.

- Ein einzelnes Element kann zufällig sein, sondern nur eine Menge an Elementen
- Ich übersetze in dieser Zusammenfassung *uniform* mit *echtzufaellig*
- In der echten Welt gibt es (*fast*) keinen perfekten Zufall, welcher zu einer Gleichverteilung führen würde. Es gibt immer Faktoren, egal wie minimal, die das Ergebnis verfälschen und damit "nur" pseudorandom machen (z.b. hat jeder Würfel, aber einer gewissen Genauigkeit Herstellungsmängel und ist damit nur Pseudorandom. Das Argument brauch ich beim nächsten Pen and Paper Abend...)

Definition 3.12 (Pseudorandom Generator)

Sei p ein beliebiges Polynom und G ein deterministischer polynomialzeit Algorithmus, welcher für die Eingabe $s \in \{0,1\}^n$ mit $n \in \mathbb{N}\backslash\{0\}$ eine Zeichenkette der Länge $l(n)$ ausgibt. G ist ein **Pseudorandom Generator PRG**, falls folgende Eigenschaften erfüllt sind:

- **Expansion:** Für jedes n gilt $l(n) > n$
- **Pseudorandomness:** Für einen stochastischen polynomialzeit berechenbaren Unterscheider D gilt:

$$| Pr[D(r) = 1] - Pr[D(G(s)) = 1] | \leq negl(n)$$

Dabei wird $Pr[D(r) = 1]$ anhand der gleichverteilen (echten zufälligen) Wahl von $r \in \{0,1\}^n$ und anhand der Zufallsprinzip von D berechnet. Die zweite Wahrscheinlichkeit wird anhand der gleichverteilen (echten zufälligen) Wahl von $s \in \{0,1\}^n$ und dem Zufallsprinzip von D berechnet.

Beispiel 3.13.

Sei $G : \{0,1\}^n \to \{0,1\}^{n+1}$ ein PRG. Sind die folgenden Konstruktionen auch PRG?

- $G_1(x) := G(x)||1$: Nein. Bekommt der Unterscheider D eine pseudozufällige Zeichenkette, so hat diese immer eine 1 am Ende und D kann eine eins ausgeben, wodurch er mit 100% richtig liegt. Bekommt D nun eine echt zufällige gibt er bei einer 1 am Ende wieder 1 aus, aber bei einer 0 am Ende schließlich 0. Damit hat er eine 50% Erfolgswahrscheinlichkeit:

$$| Pr[D(r) = 1] - Pr[D(G(s)) = 1] | = \left| \frac{1}{2} - 1 \right| = \frac{1}{2} \not\leq negl(n)$$

- $G_2(x) = G(x)||G(x)$: Ähnliche Argumentation wie G_1

- $G_3(x||b) = G(x)||b$ mit $|b| = 1$: G_3 ist auch weiterhin ein PRG. Zuersteinmal sind die nötigen Bedingungen (determinismus, polynomialzeit, Expansion) trivialerweise erfüllt. Für die Pseudorandomness nutzt man einen Reduktionsbeweis:
  1. Grundlegende Annahme: G ist ein PRG.
  2. Annahme: G_b ist kein PRG und es gibt einen Unterscheider D, der den Output von G_b besser von einer echt zufälligen Zeichenkette unterscheiden kann, als mit vernachlässigbarer Wahrscheinlichkeit
  3. Nun will man einen Unterscheider D' bauen, der anhand von D den PRG G unterscheiden kann: D' bekommt den Input s, hängt da ein echt zufälliges $b \leftarrow \{0,1\}$ dran (das ist nötig, da D einen string benötigt, der ein Bit länger ist). D' gibt nun das aus, was D ausgeben würde
  4. Dadurch entstehen zwei Fälle: 1. D' bekommt eine echt zufällige Zeichenkette, wandelt diese in eine weitere echt zufällige Zeichenkette um, D rät und damit auch D'. 2. Bekommt D' eine Zeichenkette von G_b, so ist die durch anhängen weiterhin pseudorandom, D erkennt dies und damit auch D', wodurch in dem Fall gilt (wenn in diesem Fall 1 zurückgegeben wird):

$$| Pr[D(r) = 1] - Pr[D(G(s)) = 1] | = \left| \frac{1}{2} - 1 \right| = \frac{1}{2} \not\leq negl(n)$$

  5. Somit könnte man einen Unterscheider D' entwickeln, der G mit nicht vernachlässigbarer Wahrscheinlichkeit unterscheidet, was einen Widerspruch darstellt

- $G_4(x) = G(x||0)$: Ein Gegenbeispiel hierfür wäre der PRG G_3. Wird dieser verwendet, ist am Ende immer eine 0 und die gleiche Argumentation wie für G_1 kann verwendet werden

---

Bemerkung 3.14.

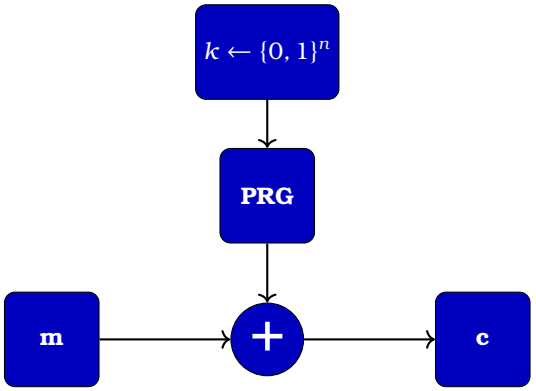
- In anderen Worten: Ein Pseudorandom Generator nutzt eine kleine zufällige Zeichenkette um daraus eine größere pseudozufällige Zeichenkette zu erzeugen
- PRG sind praktisch, da die Erzeugung von echten Zufallszahlen aufwändig ist, und man damit aus wenigen Zufallszahlen viele machen kann
- **Erklärung des Unterscheiders:** Der Unterscheider D bekommt einen String s (dieser ist echt zufällig oder pseudo zufällig). D verarbeitet die Eingabe und gibt 1 zurück, wenn er denkt s ist echt zufällig und 0, wenn er denkt s ist pseudorandom. (Die Ausgabe 1 und 0 kann auch vertauscht werden). Mit $Pr[D(G(s)) = 1]$ wird die Wahrscheinlichkeit gegeben, dass D 1 ausgibt, wenn er ein pseudozufälliges s bekommt. Mit $Pr[D(s) = 1]$ wird die Wahrscheinlichkeit gegeben, dass D 1 ausgibt, wenn er ein echt zufälliges s bekommt.
- Um zu beweisen, dass der PRG nicht funktioniert versucht man einen Unterscheider zu konstruieren, welcher besser als mit vernachlässigbarer Wahrscheinlichkeit richtig entscheiden kann
- Um die Sicherheit zu beweisen nutzt man einen Reduktionsbeweis, insofern der neue PRG auf einem richtigen PRG (oder etwas vergleichbarem) aufbaut
- In der Praxis verwendet man teilweise Lineare Kongruenzgleichungen oder logistische Funktionen als PRG, da diese effizient zu berechnen sind. Diese eignen sich aber nicht für kryptographische Anwendungen

3.6 Symmetrische Verschlüsselung mit fester Länge anhand von PRG

Definition 3.15 (Π_{PRG})

Sei G ein pseudorandom Generator PRG mit Expansionsfaktor $l(n)$. Damit kann ein symmetrisches Verfahren Π_{PRG} für Nachrichten der Länge $l(n)$ definiert werden:

- $Gen(1^n)$: Schlüssel $k \leftarrow \{0,1\}^n$ wird mit gleichverteilter Wahrscheinlichkeit erzeugt
- $Enc_k(m)$: Nachrichten $m \in \{0,1\}^{l(n)}$ werden mit $c = m \oplus G(k)$ verschlüsselt
- $Dec_k(c)$: Erzeugt den Klartext $m = c \oplus G(k)$



Satz 3.16 (Ununterscheidbarkeit des PRG Kryptosystems)

Sei G ein Pseudorandom Generator, dann ist Π_{PRG} ein symmetrisches Verschlüsselungsverfahren fester Länge, welches berechenbar Ununterscheidbar gegenüber einem abhörendem Angreifer ($PrivK^{eav}_{\mathcal{A},\Pi}(n)$) ist.

Beweis 3.17 (Mittels Reduktion).

- Grundlegendes Problem: G ist ein sicherer PRG, d.h. es gibt keinen effizienten Unterscheider D
- Annahme: Π_{PRG} ist unsicher, d.h. es existiert ein Angreifer $\mathcal{A}$ welcher im Experiment $PrivK^{eav}_{\mathcal{A},\Pi}(n)$ besser als mit vernachlässigbarer Wslk. unterscheiden kann, um welche Nachricht es sich handelt
- Damit kann man einen Unterscheider D konstruieren, welcher G effizient unterscheidet:
  1. D bekommt als Input einen String $w \in \{0,1\}^{l(n)}$
  2. D starte das $PrivK^{eav}_{\mathcal{A},\Pi}(n)$ mit A und erhält $m_0, m_1 \in \{0,1\}^{l(n)}$
  3. D wählt zufällig ein $b \in \{0,1\}$ und erzeugt $c = w \oplus m_b$
  4. D gibt c an A und erhält b'. Der Output von D ist 1 falls $b' = b$, ansonsten 0
- Es entstehen zwei Fälle:
  1. $w \in \{0,1\}^{l(n)}$ wird echtzufällig gewählt und A muss versuchen ein One-Time-Pad zu unterscheiden, was durch informationstheoretische Sicherheit unmöglich ist, und damit A und folglich auch D eine Erfolgswahrscheinlichkeit von $1/2$ haben
  2. $w \in \{0,1\}^{l(n)}$ wird vom PRG erzeugt. A kann dies besser als mit vernachlässigbarer Wslk. unterscheiden, wodurch auch die Ergebnisse von D besser als mit vernachlässigbarer Wslk. richtig sind
- **Fazit:** Da das grundlegende Problem (G ist ein PRG) gebrochen wurde entsteht ein Widerspruch. Der Widerspruch beweist, dass die Sicherheit von G auch die Sicherheit von Π_{PRG} impliziert

□

3.7 Sicherheit bei mehrfacher Verschlüsselung mit gleichem Schlüssel

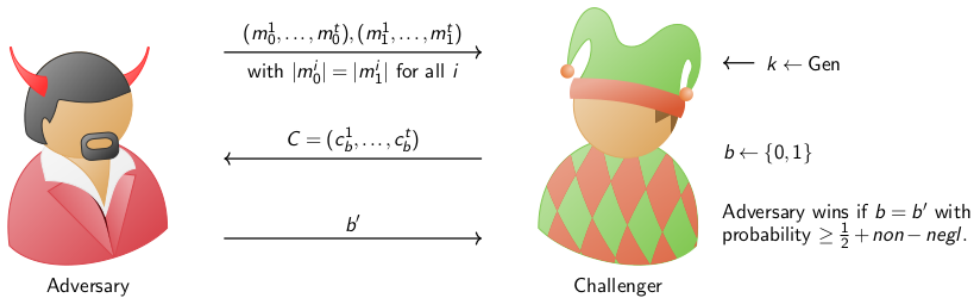
Definition 3.18 (Sicherheit für mehrfache Verschlüsselung mit gleichem Schlüssel)

Ein Kryptosystem Π hat mehrfach ununterscheidbare Verschlüsselung gegenüber einem Abhörer, wenn für jeder probabilistische polynomialzeit Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ existiert, sodass gilt:

$$P[PrivK^{mult}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Experiment $PrivK^{mult}_{\mathcal{A},\Pi}(n)$

Dieses Experiment ist wieder mit einem Gegenspieler $\mathcal{A}$ und einem Challenger:



$PrivK^{mult}_{\mathcal{A},\Pi}(n)$
$(m_0^1,\ldots,m_0^t,m_1^1,\ldots,m_1^t) \leftarrow \mathcal{A}(1^n), \lvert m_0^i\rvert = \lvert m_1^i\rvert \forall i \in [1,t]$
$k \leftarrow Gen(1^n)$
$b \leftarrow \{0,1\}$
$C = (c_b^1,\ldots,c_b^t) \leftarrow (Enc_k(m_b^1),\ldots,Enc_k(m_b^t))$
$b' \leftarrow \mathcal{A}(C)$
if $b' = b$ return 1 else return 0

Man schreibt $PrivK^{mult}_{\mathcal{A},\Pi}(n) = 1$ wenn der Output des Experimentes 1 ist und $\mathcal{A}$ gewonnen hat.

Bemerkung 3.19.

- **Abkürzung:** ING-EAV-Mult-Sicherheit (indistinguishable multiple encryption eavesdropper security), bzw- EAV-Mult-Sicherheit
- **Vergleich zu der vorherigen Sicherheitsdefinitionen:** Diese Sicherheitsdefinition ist stärker als die vorrangegangene Definition, bei der nur eine Nachricht verschlüsselt und unterschieden wurde, da EAV-Single ein Spezialfall von EAV-Mult ist
- **Konstruktion:** Ein deterministische Kryptosystem kann die Sicherheit für mehrfache Verschlüsselung niemals erfüllen: Würde man hierbei als Gegenspieler $M_0 = (0^n, 0^n)$ und $M_1 = (0^n, 1^n)$ für $PrivK^{mult}_{\mathcal{A},\Pi}(n)$ wählen, so könnte man die Verschlüsselungen immer unterscheiden
- **Folgerung:** Kryptosysteme für mehrfache Verschlüsselungen dürfen nicht deterministisch sein und müssen Zufallsvariablen enthalten

3.8 Sicherheit gegen Chosen-Plaintext Attacks

Definition 3.20 (Sicherheit gegen Chosen Plaintext Attacks)

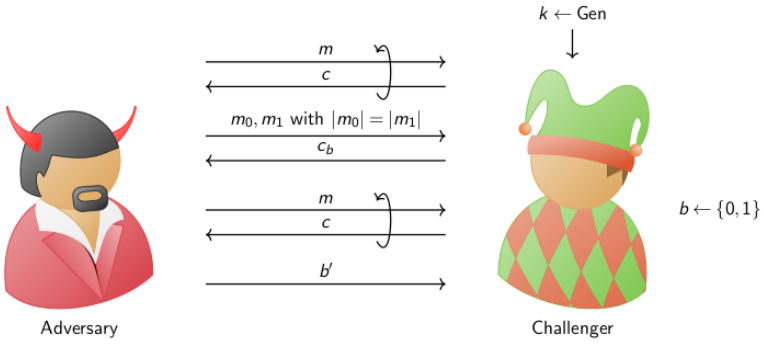
Ein Kryptosystem Π hat ununterscheidbare Verschlüsselung gegenüber einem Chosen Plaintext Attack, wenn für alle probabilistischen polynomialzeit Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ existiert, sodass gilt:

$$P[PrivK^{cpa}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Experiment $PrivK^{cpa}_{\mathcal{A},\Pi}(n)$

Dieses Experimant ist wieder mit einem Gegenspieler $\mathcal{A}$ und einem Challenger:

$PrivK^{cpa}_{\mathcal{A},\Pi}(n)$
$k \leftarrow Gen(1^n)$
$(m_0, m_1) \leftarrow \mathcal{A}^{Enc_k(\cdot)}$, $\mid m_0 \mid = \mid m_1 \mid$
$b \leftarrow \{0,1\}$
$c_b \leftarrow Enc_k(m_b)$
$b' \leftarrow \mathcal{A}^{Enc_k(\cdot)}(c_b)$
if $b' = b$ return 1 else return 0

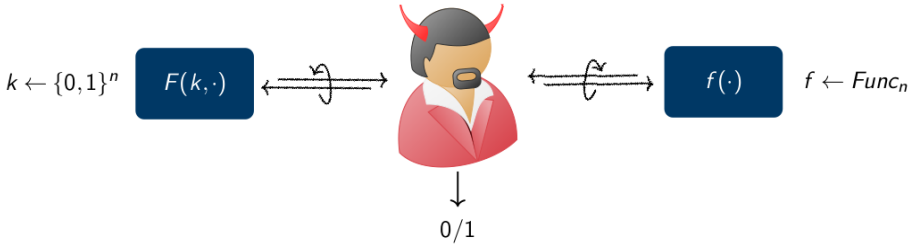


Mit $\mathcal{A}^{Enc_k(\cdot)}$ ist gemeint, dass der Gegenspieler $\mathcal{A}$ Zugang zu einem Verschlüsselungsorakel hat, welches beliebige Nachrichten von $\mathcal{A}$ mit dem Schlüssel des Challengers k verschlüsselt. $\mathcal{A}$ kann das Orakel beliebig oft nutzen. Man schreibt $PrivK^{cpa}_{\mathcal{A},\Pi}(n) = 1$ wenn der Output des Experimentes 1 ist und $\mathcal{A}$ gewonnen hat.

Bemerkung 3.21.

- **Abkürzung:** ING-CPA-Sicherheit (Indistinguishable chosen plaintext attack secure) bzw. CPA-Sicherheit
- **Erinnerung:** Chosen Plaintext Angriffe ermöglichem einen Angreifer die Verschlüsselung beliebiger Nachrichten mit dem Verfahren, welches die Angegriffenen Parteien nutzen
- **Anmerkung Einfache und Mehrfache Verschlüsselung:** Ein Kryptosystem das CPA-Sicher für einfache Verschlüsselung mit gleichem Schlüssel ist, ist auch CPA-Sicher für mehrfache Verschlüsselung mit gleichem Schlüssel und andersrum, d.h. (ING-CPA-Single-Secure $\Leftrightarrow$ ING-CPA-Mult-Secure) = ING-CPA-Secure! (Beweis durch Hybrid Argument, unwichtig)
- **Vergleich zu den vorherigen Sicherheitsdefinitionen:** Da das ING-EAV-Mult und ING-EAV-Single Experiment im CPA-Experiment enthalten sind, kann man per Reduktion zeigen, dass CPA-Sicherheit auch ING-EAV-Mult-Sicherheit und ING-EAV-Single-Sicherheit impliziert (Andersum gilt dies nicht). D.h. auch, dass CPA-Sicherheit stärker als die vorherigen Sicherheitsdefinitionen ist
- **Anmerkung:** Diese Definition stellt das Minimum an Sicherheit dar, welches heute für Kryptosysteme gelten sollte

3.9 Pseudorandom Funktionen



Definition 3.22 (Pseudorandom Funktionen PRF)

Sei $F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ eine effiziente, längenerhaltende, schlüsselabhängige Funktion. F ist eine pseudorandom Funktion PRF, wenn für alle probabilistischen polynomialzeit Unterscheider D eine vernachlässigbare Funktion $negl(n)$ existiert, sodass gilt:

$$| P[D^{F(k,\cdot)}(1^n) = 1] - P[D^{f(\cdot)}(1^n) = 1] | \leq negl(n)$$

Dabei wird $P[D^{F(k,\cdot)}(1^n) = 1]$ anhand der gleichverteilen (echten zufälligen) Wahl von $k \in \{0,1\}^n$ und anhand der Zufallsprinzip von D berechnet. Die zweite Wahrscheinlichkeit wird anhand der gleichverteilen (echten zufälligen) Wahl von $f \in Func_n$ und dem Zufallsprinzip von D berechnet.

Beispiel 3.23.

Sei $F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ eine PRF. Sind die folgenden Konstruktionen auch Pseudorandom Funktionen? (Längenerhaltung wird Ignoriert!)

- $F'_k(x) = F_k(x)||0$: Keine PRF, ein Unterscheider D gibt ein beliebiges x an das Orkale und erhält y und gibt einfach das letzte Bit von y aus.
- $F'_k(x) = F_k(x \oplus 1^n)$: Ja, $F'_k(x)$ ist weiterhin eine PRF. Beweis durch Reduktion:
  1. Grundlegende Annahme: F_k ist eine PRF
  2. Annahme: F'_k ist keine PRF und es gibt einen Unterscheider D, der den Output von F'_k besser von einer echt zufälligen Zeichenkette unterscheiden kann, als mit vernachlässigbarer Wahrscheinlichkeit
  3. Konstruktion von D' um F_k anhand von D zu unterscheiden: D' nimmt eine beliebige Zeichenkette (Länge n) s berechnet damit $s' = s \oplus 1^n$. s' gibt D' nun an sein Orakel, welches $x = F_k(s') = F_k(s \oplus 1^n)$ berechnet. Dann gibt D' x an D und folglich gibt D' das Ergebnis von D aus
  4. Es entstehen zwei Fälle: 1. D' bekommt eine echt zufällige Funktion als Orakel und kann daher nur mit einer Wslk. von $1/2$ anhand von D den richtigen Output generieren. 2. D' bekommt die Funktion $F_k(x)$ als Orakel, welche D folglich besser als vernachlässigbarer Wslk. unterscheiden kann, wodurch auch D' die Eingabe besser als mit vernachlässigbarer Wslk. unterscheiden kann
  5. Dadurch entsteht ein Widerspruch und F_k ist eine PRF
- $F'_k(x) = F_k(x)||F_k(x \oplus 1^n)$: Keine PRF, ein Unterscheider D gibt zuerst $m_0 = 0^n$ an das Orakel und erhält $y_0^0||y_0^1$, danach gibt er $m_1 = 1^n$ an das Orakel und erhält $y_1^0||y_1^1$. Bei F'_k gilt immer $y_0^0 = y_1^1$ bzw. $y_0^1 = y_1^1$ bei einer echt zufälligen Funktionen nur mit vernachlässigbarer Wslk.
- $F'_k(x_1||x_2) = F_k(x_1)||F_k(x_2)$ mit $x_1, x_2 \in \{0,1\}^n$: Kein PRF, ein Unterscheider D gibt dem Orakel $x||x$ und erhält $y_1||y_2$. Bei F'_k gilt immer $y_1 = y_2$ bei einer echt zufälligen Funktionen nur mit vernachlässigbarer Wslk.
- $F'_k(x) = F_k(0||x)||F_k(1||x)$ mit $x \in \{0,1\}^{n-1}$: Ja, $F'_k(x)$ ist weiterhin eine PRF. Grund: F_k hat immer unterschiedliche Eingaben, wodurch der Output immer pseudozufällig ist. Beweis durch Reduktion:
  1. Grundlegende Annahme: F_k ist eine PRF
  2. Annahme: F'_k ist keine PRF und es gibt einen Unterscheider D, der den Output von F'_k besser von einer echt zufälligen Zeichenkette unterscheiden kann, als mit vernachlässigbarer Wahrscheinlichkeit

3. Konstruktion von D' um F_k anhand von D zu unterscheiden: D' gibt seinem Orakel zuerst $0\|x$ mit und erhält a. Danach gibt er dem Orakel $1\|x$ mit und erhält b. x kann beliebig gewählt von Länge n sein. D' gibt D nun $a\|b$ mit und anschließend gibt D' den Output von D aus
4. Es entstehen zwei Fälle: 1. D' bekommt eine echt zufällige Funktion als Orakel und kann daher nur mit einer Wslk. von $1/2$ anhand von D den richtigen Output generieren. 2. D' bekommt die Funktion $F_k(x)$ als Orakel, welche D folglich besser als vernachlässigbarer Wslk. unterscheiden kann, wodurch auch D' die Eingabe besser als mit vernachlässigbarer Wslk. unterscheiden kann
5. Dadurch entsteht ein Widerspruch und F_k ist eine PRF

- $F'_k(x) = F_k(x\|0)\|F_k(x\|1)$ mit $x \in \{0,1\}^{n-1}$: Ja, ähnliche Argumentation wie oben
- $F'_k(x) = F_k(0\|x)\|F_k(x\|1)$ mit $x \in \{0,1\}^{n-1}$: Nein, da man gleiche Inputs erzeugen kann. Ein Unterscheider D erstellt zuerst eine Zeichenkette $m_0 = 0^{n-1}\|1$, gibt die dem Orakel und erhält eine Zeichenkette der Form $m_r es0 = a\|b$. Dann erstellt D die Zeichenkette $m_1 = 0^n$ und gibt die dem Orakel und bekommt immer $m_r es1 = c\|a$, insofern er mit der Pseudozufälligen Funktion interagiert. Wenn er mit der echt zufälligen Funktion interagiert, erhält D dies nur mit vernachlässigbarer Wslk.

Bemerkung 3.24.

- $Func_n$ ist die Menge aller Funktionen, die ein n-bit Eingabe auf ein n-bit Ausgabe projeziert
- Typischerweiße wird für den Nachweis ein key k echt zufällig gewählt und die zu Prüfende Funktion mit k festgesetzt (Für einen Durchlauf des Experimentes). Der $\mathcal{D}$ bekommt niemals den key
- Anstatt nun also Zufällig aussehende Zeichenkette zu betrachten, wie es bei PRG der Fall war, betrachtet man sich hier zufällig aussehende Funktionen. D.h. Ein Unterscheider bekommt eine Funktion, hier auch Orakel genannt, mit der er beliebig arbeiten kann. Das Orakel ist aber deterministisch, d.h. für die gleiche Eingabe gibt es den gleichen Output. Das Ziel ist zu unterscheiden, ob diese Funktion echt zufällig oder die konstruierte pseudozufällige Funktion ist. Kann ein Angreifer dies nicht unterscheiden, so ist die konstruierte Funktion wirklich pseudozufällig, ansonsten nicht
- **Anmerkung Längenerhaltend:** Annahme zur Vereinfachung die nicht zwingend notwendig ist

Satz 3.25 (Existenz von Pseudorandom Funktionen)

Pseudorandom Funktionen existieren genau dann wenn Pseudoranom Generatoren existieren.

Beweis 3.26.

- Beweisidee: Aus PRF kann man PRG erzeugen und andersrum
- **1. Fall PRG mittel PRF:** Sei F_k eine PRF, dann ist $G(k) = F_k(0^n)\|F_k(1^n)$ ein PRG
- **2. Fall PRF mittel PRG:** Möglich mit Goldreich Goldwasser Micali Konstruktion. Diese wird später genauer erläutert

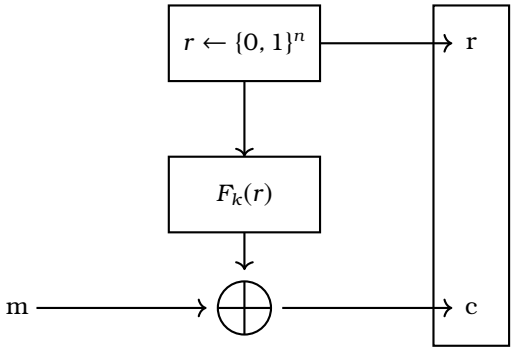
□

3.10 Verschlüsselung mittels PRF mit Sicherheit vor CPA

Definition 3.27 (Π_{PRF})

Sei F_k ein pseudorandom Funktion PRF. Damit kann ein symmetrisches Verfahren Π_{PRF} für Nachrichten der Länge n definiert werden:

- $Gen(1^n)$: Schlüssel $k \leftarrow \{0,1\}^n$ wird mit gleichverteilter Wahrscheinlichkeit erzeugt
- $Enc_k(m)$: Wähle $r \leftarrow \{0,1\}^n$ mit gleichverteilter Wahrscheinlichkeit und berechne $c = (r, s) = (r, F_k(r) \oplus m)$
- $Dec_k(c)$: Erhalte $c = (r, s)$ und berechne $m = F_k(r) \oplus c$



Satz 3.28 (CPA-Sicherheit von Π_{PRF})

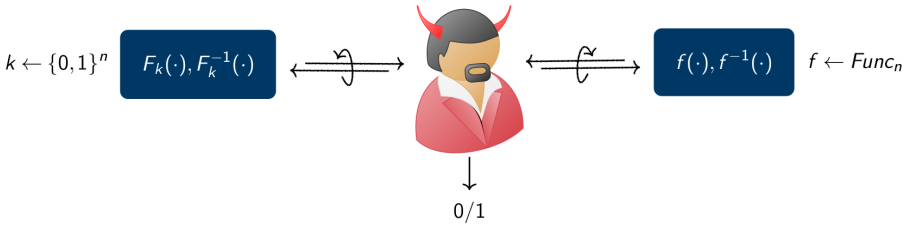
Ist F eine pseudorandom Funktion, dann ist Π_{PRF} sicher vor Chosen-Plaintext Angriffen.

Beweis 3.29.

- Grundlegende Annahme: F_k ist eine PRF
- Annahme: Π_{PRF} ist unsicher, d.h. es gibt einen Angreifer A der das CPA-Experiment besser als mit vernachlässigbarer Wslk gewinnt:
$$P[P^{cpa}_{\Pi,A}(1^n) = 1] = \frac{1}{2} + \epsilon$$
- **Einschub (Kurze genauere Analyse von A):** Kommuniziert A mit einer echt zufälligen Funktion, bzw. einer PRF, so kann A im Zuge des Experimentes mehrfach, also $q(n)$-mal, Nachrichten entschlüsseln lassen. Bei jeder Entschlüsselung hat A die Wahrscheinlichkeit $\frac{1}{2^n}$ das CPA-Experiment mit Wslk. 1 zu gewinnen. Daraus folgt:
$$P[P^{cpa}_{\Pi,A}(1^n) = 1] = \frac{1}{2} + \frac{q(n)}{2^n}$$
- Konstruktion von R_A um F_k anhand von A zu unterscheiden: R_A bekommt entweder F_k oder eine echt zufällige Funktion gegeben. Mit dieser kann R_A das Kryptosystem Π_{PRF} simulieren. Damit kann er als Challenger im CPA-Experiment agieren und A als Gegenspieler nutzen. R_A gibt anschließend den Output von A aus. Es entstehen zwei Fälle:
  1. R_A bekommt eine echt zufällige Funktion und simuliert das CPA-Experiment. In diesem Fall ist das Kryptosystem im CPA-Experiment informationstheoretisch sicher. A kann dieses also nur mit Wslk $P[D^{f(k,\cdot)}(1^n)] = \frac{1}{2} + \frac{p(n)}{2^n}$ unterscheiden
  2. R_A bekommt eine echt zufällige Funktion und simuliert das CPA-Experiment. A kann dies nun mit Wslk $P[D^{F(k,\cdot)}(1^n)] = \frac{1}{2} + \frac{p(n)}{2^n} + \epsilon$ unterscheiden
- Kombiniert man beide Fälle so erhält man folgendes Resultat
$$\mid P[D^{F(k,\cdot)}(1^n) = 1] - P[D^{f(\cdot)}(1^n) = 1] \mid = \epsilon > negl(n)$$
- Das führt zu einem Widerspruch, da dies implizieren würde, dass D effizient PRF unterscheiden kann. Daher muss Π_{PRF} CPA-Sicher sein

□

3.11 Pseudorandom Permutationen



Definition 3.30 (Pseudorandom Permutierer)

Sei $F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ einer effizienter, längenerhaltender, schlüsselabhängiger Permutierern. F ist ein starker pseudorandom Permutierer, wenn für alle probabilistischen polynomialzeit Unterscheider D eine vernachlässigbare Funktion $negl(n)$ existiert, sodass gilt:

$$P[D^{F_k(\cdot),F_k^{-1}(\cdot)}(1^n) = 1] - P[D^{f_k(\cdot),f_k^{-1}(\cdot)}(1^n) = 1] < negl(n)$$

Dabei wird die erste Wahrscheinlichkeit über die echt zufällige Wahl von $k \in \{0,1\}^n$ und das Zufallsprinzip von D berechnet und die zweite Wahrscheinlichkeit wird über die echt zufällige Wahl von $f \in Perm_n$ und das Zufallsprinzip von D berechnet.

Bemerkung 3.31.

- PRP sind bijektive PRF und damit ein Spezialfall von PRF der die invertierte Berechnung zulässt
- Wenn F ein PRP ist, dann ist F auch eine PRF
- Der Unterscheider hat in diesem Fall ein Orakel welches sowohl F_k als auch F_k^{-1} berechnet
- Ähnlich zu PRP sind Bit-Permutierer. Diese erhalten die 1er und 0er der Eingabe und vertauschen nur die jeweiligen Positionen. Bit-Permutierer sind keine starken PRP, da ein Unterscheider D die Anzahl der 1er und 0er speichert und für die Ein und Ausgabe für sein Orakel abgleicht. Damit kann er einen Bit-Permutierer erkennen

3.12 Operationsmodie für Blockchiffren

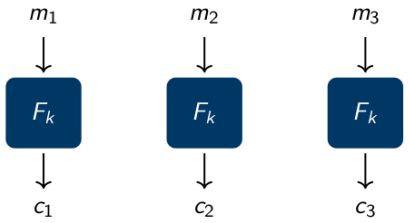
Definition 3.32 (Blockchiffren)

Bisher konnten die Kryptosystem Nachrichten beliebiger Länge bearbeiten. Bei **Blockchiffren** ist dies nicht der Fall. Hierbei werden Nachrichten in festen Blöcke aufgeteilt und diese jeweils verschlüsselt. Der letzte Block wird überlicherweise gepadded, d.h. aufgefüllt damit er die nötige Länge hat.

Bemerkung 3.33.

- Blockchiffren sind (je nach Wahl) sichere Instanzen eines starken pseudorandom Permutierers mit fester Schlüssel und Blocklänge
- Die folgenden Operationsmodie beschreiben unterschiedliche Umsetzungen von Blockchiffren
- Alle Modie die vorgestellt werden sind nicht CCA-Sicher!

3.12.1 Electronic Code Book (ECB) Modus



ECB mode

secure mode

Satz 3.34 (ECB und CPA-Sicherheit)

ECB ist nicht CPA-sicher, da ECB deterministisch ist.

Satz 3.35 (ECB und EAV-Sicherheit)

ECB ist nicht EAV-Mult-sicher.

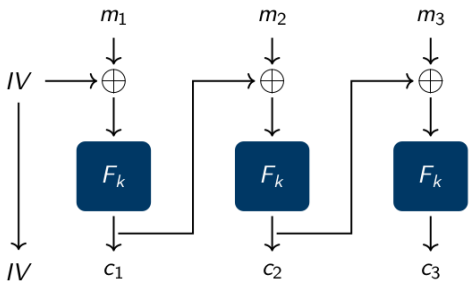
Beweis 3.36.

Als Gegenspieler A im EAV-Mult Experiment kann die Nachricht $m_0 = 0^{2n}$ und die Nachricht $m_0 = 0^n \| 1^n$ dem Challenger geben. Ist der zurückkommende Crpytotext von der Form $c\|c$, dann gibt A 0 aus, ansonsten 1. Damit gewinnt A immer das Experiment und ECB ist nicht EAV-Mult-Sicher. □

3.12.2 Cipher Block Chaining (CBC) Modus

Satz 3.37 (CBC ist CPA-Sicherheit)

Wenn IV echt zufällig ist und F_k ein PRP ist, dann ist CBC auch CPA-sicher.



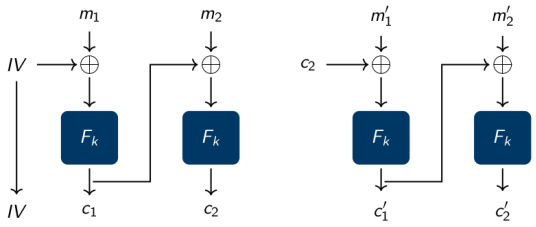
Bemerkung 3.38.

- CPA-Beweis ist ähnlich wie der für den CTR Modus
- IV ein ein echt zufällig ausgwählter Initial Vektor der Länge n
- CBC erfüllt die Eigenschaften der korrekten Ver und Entschlüsselung, daher muss F_k auch ein PRP sein

3.12.3 Chained Cipher Block Chaining (Chained CBC) Modus

Bemerkung 3.39.

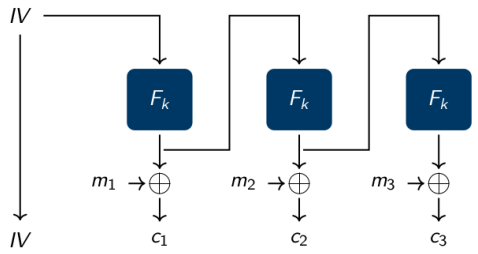
- Der letzte Block des vorrangegangenen Chiffretextes ist der IV des nächsten Chiffretextes
- Dadurch kennt ein Angreifer einige IV und das Kryptosystem ist nicht mehr CPA-sicher
- Daher gilt: Auch kleine Veränderungen an einem Kryptosystem können es schon unsicher machen



3.12.4 Output Feedback (OFB) Modus

Satz 3.40 (OFB ist CPA-Sicherheit)

Wenn IV echt zufällig ist und F_k ein PRF ist, dann ist OFB auch CPA-sicher.



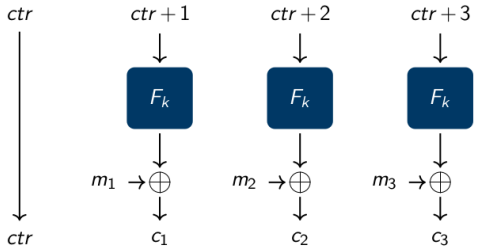
Bemerkung 3.41.

- CPA-Beweis ist ähnlich wie der für den CTR Modus
- Der Vorteil von OFB gegenüber CBC ist, dass hierbei Preprocessing angewendet werden kann (F_k kann zeitlich mit der vorhergehenden Verschlüsselung berechnet werden), wodurch OFB effizienter sein kann

3.12.5 Counter (CTR) Modus

Satz 3.42 (CTR ist CPA-Sicherheit)

Wenn F_k ein PRF ist, dann ist CTR auch CPA-sicher.



Bemerkung 3.43.

- Im Gegensatz zu den anderen Varianten (bis auf ECB) ist dieser Modus gut parallelisierbar und daher effizienter

Beweis 3.44.

1. Grundlegenste Annahme: Die PRF ist sicher
2. Annahme: Π_{CTR} ist unsicher, d.h. es gibt ein A der das CPA-Experiment effizient gewinnt:
$$P[P^{cpa}_{\Pi_{CTR},A}(1^n) = 1] = \frac{1}{2} + \epsilon$$
3. Konstruktion von R_A um PRF anhand von A zu unterscheiden: R_A bekommt entweder die PRF oder eine echt zufällige Funktion f gegeben. Es entstehen zwei Fälle:
   (a) R_A bekommt die PRF. D nutzt Π_{CTR} mit der PRF und simuliert damit und mit A als adversary das CPA-Experiment. A kann Π_{CTR} besser als mit vernachlässigbarer Wahrscheinlichkeit brechen und R_A gibt einfach den Output von A aus
   (b) R_A bekommt die echt zufällige Funktion f: Damit muss man nun zuerst das zugrundeliegende Kryptosystem anpassen, sodass dieses f verwendet und zu Π'_{CTR} wird. A kann Π'_{CTR} nun nicht effizient brechen: A bekommt vom Challenger (IV, c) und kann $p(n)$ polynomial oft Anfragen an sein Verschlüsselungsorakel senden. Von diesem bekommt A die Antworten (IV_i, c_i). Dabei sind nun die Überscheidungen interessant: Gilt $IV = IV_i$, so gibt es eine Überschneidung. Da CTR aber sequentiell hochzählt erhält man auch bei $IV + x = IV_i + y$ für beliebig mögliche x, y eine Überschneidung. Anhand des Geburtstagsparadoxon kann man daraus die Wahrscheinlichkeit für eine Überschneidung berechnen, nämlich $2p(n)^2/2^n$. Tritt eine Überschneidung auf, kann A das Experiment mit Wahrscheinlichkeit 1 richtig beantworten. Ansonsten muss A raten. Zusammengefasst ergibt sich also die Erfolgswahrscheinlichkeit:
$$P[P^{cpa}_{\Pi'_{CTR},A}(1^n) = 1] = \frac{1}{2} + \frac{2p(n)^2}{2^n} \leq \frac{1}{2} + negl(n)$$
   D nutzt nun Π'_{CTR} mit f und simuliert damit und mit A als adversary das CPA-Experiment. Diesmal kann A Π'_{CTR} nicht besser als mit vernachlässigbarer Wahrscheinlichkeit brechen. Trotzdem gibt R_A wieder einfach den Output von A aus
4. Fässt man dies zusammen, so kann R_A die PRF besser als mit vernachlässigbarer Wahrscheinlichkeit entscheiden:
$$P[D^{PRF}(1^n) = 1] - P[D^f(1^n) = 1] > negl(n)$$
5. Dadurch entsteht ein Widerspruch zur Grundlegenden Annahme, welcher die CPA-Sicherheit von Π_{CTR} beweist
6. **Anmerkung:** Es wurde ING-CPA-Sicherheit für einzelne Nachrichten bewiesen, was aber auch die gleiche Sicherheit für mehrfache Nachrichten impliziert

□

3.13 Sicherheit gegenüber Chosen Ciphertext Angriffen

Definition 3.45 (CCA-Sicherheit)

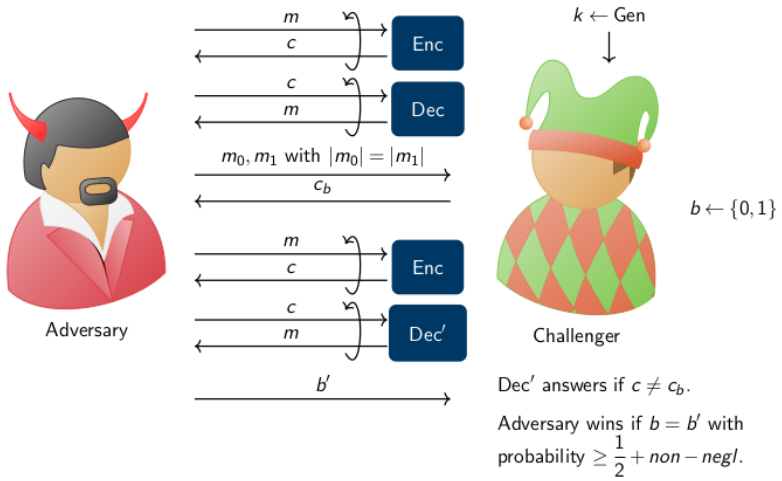
Ein Kryptosystem Π hat ununterscheidare Verschlüsselung gegenüber einem Chosen Ciphertext Angriff, auch CCA-Sicherheit genannt, falls für jeden probabilistischen polynomialzeit Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ existiert, sodass gilt:

$$P[PrivK^{cca}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Experiment $PrivK^{cca}_{\mathcal{A},\Pi}(n)$

Dieses Experiment ist wieder mit einem Gegenspieler $\mathcal{A}$ und einem Challenger:

$PrivK^{cca}_{\mathcal{A},\Pi}(n)$
$k \leftarrow Gen(1^n)$
$(m_0, m_1) \leftarrow \mathcal{A}^{Enc_k(\cdot),Dec_k(\cdot)}(1^n)$ mit $\lvert m_0\rvert = \lvert m_1\rvert$
$b \leftarrow \{0,1\}$
$c_b \leftarrow Enc_k(m_b)$
$b' \leftarrow \mathcal{A}^{Enc_k(\cdot),Dec_k(\cdot)}(c_b), c \neq c_b$
if $b' = b$ return 1 else return 0



Dabei bedeutet $c \neq c_b$, dass der Gegenspieler $\mathcal{A}$ alle Ciphertexte bis auf c_b seinem Entschlüsselungorakel mitgeben kann. Es wird $PrivK^{cca}_{\mathcal{A},\Pi}(n) = 1$ geschrieben falls die Ausgabe des Experimentes 1 ist, was impliziert, dass $\mathcal{A}$ erfolgreich war.

Bemerkung 3.46.

- **Abkürzung:** Diese Sicherheit wird auch als ING-CCA-Sicherheit (Indistinguishable chosen ciphertext attack secure) bzw. CCA-Sicherheit bezeichnet
- **Erinnerung:** Bei einem Chosen Ciphertext Angriff kann ein Angreifer beliebige Nachrichten (bis auf die abgehörte Nachricht) ver- und entschlüsseln lassen
- **Anmerkung Einfache und Mehrfache Verschlüsselug:** Ein Kryptosystem das CCA-Sicher für einfache Verschlüsselung mit gleichem Schlüssel ist, ist auch CCA-Sicher für mehrfache Verschlüsselung mit gleichem Schlüssel und andersrum, d.h. ING-CCA-Single ⇔ ING-CCA-Mult!
- **Vergleich zu den vorherigen Sicherheitsdefinitionen:** Da das CPA Experiment im CCA Experiment enthalten ist, kann man per Reduktion zeigen, dass CCA-Sicherheit auch CPA-Sicherheit impliziert (Andersum gilt dies nicht). D.h. auch, dass CCA-Sicherheit stärker als die vorherigen Sicherheitdefinitionen ist
- **Unverkettbarkeit**: CCA-Sicherheit impliziert Unverkettbarkeit (non malleability) von Nachrichten, d.h. verändert der Angreifer im Experiment den Ciffretext nur minimal, so verrät die dazugehörige Entschlüsselung nur noch vernachlässigbare viel Informationen über den ürsprünglichen Klartext
- **Konstruktion:** CCA-sichere Verfahren können mit dem bisherigen Wissen nicht implementiert werden. Dafür braucht man z.b. MACs, welche später erläutert werden

Satz 3.47 (CCA-Sicherheit und bisherige Kryptosysteme)

Die bisher vorgestellten Kryptosysteme sind nicht CCA-sicher!

Beweis 3.48.

- Auch CCA-sichere Kryptosysteme dürfen nicht deterministisch sein, daher sind alle deterministischen Verfahren nicht CCA-sicher

- Die restlichen bisherigen Verfahren bauen auf PRF und der XOR-Operation auf und funktionieren folgendermaßen:
$$Enc(k, m) = (r, s) = (r, F(k, r) \oplus m)$$
- Ein Gegenspieler A im CCA Experiment sähe folgendermaßen aus:
  1. Setze $m_0 = 0^n$ und $m_1 = 1^n$
  2. A bekommt (r, s) und erzeugt daraus (r, s') indem er das erste Bit von s flippt
  3. A sendet (r, s') an sein Entschlüsselungsorakel und erhält entweder $0||1^{n-1}$ oder $1||0^{n-1}$
  4. Er kann die Nachricht unterscheiden, insofern $n > 2$ ist und gewinnt das Experiment

□

Beispiel 3.49.

Beweis, dass CBC nicht CCA-sicher ist:
1. Der Gegenspieler A schickt die beiden Nachrichten $m_0 = 0^n$ und $m_1 = 1^n$
2. A bekommt $IV||c_1$ und erzeug $(IV \oplus \partial)||c_1$ mit einem beliebigen ∂
3. A schickt $(IV \oplus \partial)||c_1$ an sein Entschlüsselungsorakel und erhält als Ausgabe $m' = m_b \oplus \partial$
4. A kann ∂ herausrechnen und die Nachrichten unterscheiden

Beweis, dass CTR nicht CCA-sicher ist:
1. Der Gegenspieler A schickt die beiden Nachrichten $m_0 = 0^n$ und $m_1 = 1^n$
2. A bekommt $c = m_b \oplus PRF(...)$ und erzeug damit $c' = m_b \oplus PRF(...) \oplus \partial$
3. A schickt c' an sein Entschlüsselungsorakel und erhält $m_b \oplus \partial$
4. A kann ∂ herausrechnen und die Nachrichten unterscheiden

3.13.1 Orakel Padding Angriffe*

Zugrundeliegendes System

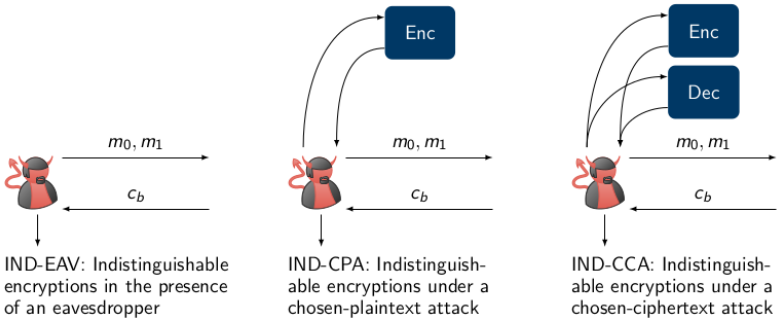
- Bei den Block Chiffren muss der letzte Block gepadded werden
- Sei L die Blocklänge und b die Anzahl an bytes die angehängt werden soll
- Beim Verschlüsseln (beispielsweise mit CBC-Modus) wird schließlich die Anzahl an bytes die gepadded werden hinzugepadded, sodass beim Entschlüsseln überprüft werden kann, ob das Padding verändert wurde oder nicht
- Wurde das Padding verändert, entsteht ein PaddingIncorrectError
- Somit ist der Wertebereich $b \in \{1, ..., L\}$, da $b = 0$ durch die Überprüfung beim Entschlüsseln nicht möglich ist

Erläuterung des Angriffes

- Zur Vereinfachung wird ein Beispiel mit dem Ciphertext (IV, c_1, c_2) betrachtet und mit den unbekannten Nachrichten m_1 und m_2 (in CBC Modus)
- Es gilt $m_2 = F_k^{-1}(c_2) \oplus c_1$ und $m_2 = m_2'||0xb, ..., 0xb$
- Der Angriff:
  1. **Die Länge des Paddings herausfinden:** Man modifiziert das letzte Byte von c_1. Falls man einen PaddingIncorrectError bekommt, modifiziert man das vorletzte Byte usw. Das wiederholt man solange, bis man keinen Error bekommt und die Anzahl der benötigten Schritte gibt die Länge des Paddings an (Man nimmt c_1 da Veränderung hierbei auch die gleichen Bytes von c_2 beeinträchtigen und man somit auch den Spezialfall $b = L$ endeckt)
  2. **Entschlüsselung einzelner Bytes:**
     - m_2 endet mit $0xB||0xb...||0xb$, wobei $0xB$ das ite Byte von m_2 ist
     - Dann erstellt man eine Zeichenkette, mit der man das Padding und ite Bit von c_1 verändert: $\delta_i = 0x00||...||0xi||0x(b+1)...||0x(b+1) \oplus 0x00||...||0x00||0xb..||0xb$ (Mit dem hinteren Teil negiert man das vorherige Padding)
     - Man verschickt $(IV, c_1 \oplus \delta_i, c_2)$
     - Man erhält solange einen Error, bis $0x(B \oplus i) = 0x(b+1)$ gilt, woraus man auf das Byte des Klartextes schließen kann
     - Auf Ähnliche Weise können auch die weiteren Bytes entschlüsselt werden

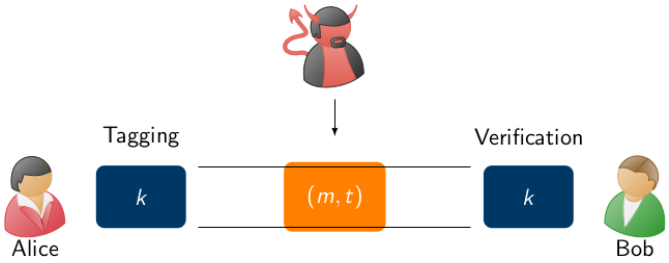
Bemerkung 3.50.
- Dieser praktische Angriff zeigt, dass CCA-Sicherheit nicht nur ein theoretisches Konstrukt ist
- Die Rückgabe des Errors stellt ein vereinfachtes Entschlüsselungsorakel dar

3.14 Zusammenfassung der einzelnen Sicherheitdefinitionen



EAV-Single ⊂ EAV-Mult ⊂ CPA-Single = CPA-Mult = CPA ⊂ CCA-Single = CCA-Mult = CCA

4 Message Authentication Codes



Definition 4.1 (Message Authentication Codes)

Ein Message Authentication Codes (MAC) ist ein Tupel aus drei probabilistischen polynomialzeit Algorithmen die folgendermaßen definiert sind:

- $k \leftarrow Gen(1^n)$: Der Schlüsselgenerierungsalgorithmus erzeugt einen Schlüssel k sodass $|k| \geq n$ gilt
- $t \leftarrow Mac(k, m)$: Der Tag-Generierungs-Algorithmus nimmt als Input den Schlüssel k und eine Nachricht $m \in \{0,1\}^*$ und erzeugt daraus den Tag t. Da der Algorithmus randomisiert sein kann, schreibt man auch $t \leftarrow Mac_k(m)$ (Pfeil wg. Random)
- $b = Vrfy(k, m, t)$: Der Verifizierungsalgorithmus nimmt als Eingabe den Schlüssel k, eine Nachricht m und einen Tag t. Die Ausgabe ist ein bit b mit $b = 1$, wenn die Verifikation korrekt ist (d.h. zielmäßig die Integrität von m erhalten wurde), und ansonsten wird $b = 0$ ausgegeben. Die Allgemeine Annahme ist, dass Vrfy deterministisch ist (Daher auch das '=')

Bemerkung 4.2.

- Das grundlegende Ziel ist erstmal die Sicherstellung der Integrität einer Nachricht
- MACs bieten nach dieser Definition noch keine Sicherstellung der Vertraulichkeit!
- Auch für MACs gilt Korrektheit: $Vrfy_k(m, Mac_k(m)) = 1$
- Ein MAC fester Länge (fixed length MAC) ist für Nachrichten genau einer Länge definiert

Definition 4.3 (Kanonische Verifikation)

Ein deterministischer MAC hat kanonische Verifikation, wenn der Verifikationsalgorithmus einfach auch den Tag mittels des Tag-Generierungs-Algorithmus berechnen und auf Gleicheit prüfen kann:

$$Vrfy(k, m, t) = \begin{cases} Mac_k(m) = t & , output\ b = 1 \\ Mac_k(m) \neq t & , output\ b = 0 \end{cases}$$

Definition 4.4 (Sicherheit für MACs)

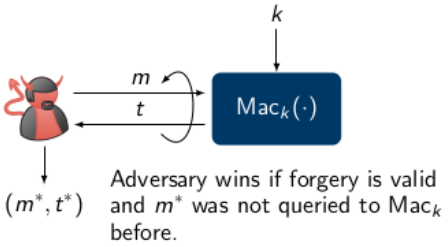
Ein MAC Π_{MAC} ist sicher (fälschungsicher, existentially unforgeable under an adaptive chosen message attack), wenn es für jeden probabilistischen polynomialzeit Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ gibt, sodass gilt:

$$P[MacForge_{\mathcal{A},\Pi_{MAC}}(n) = 1] \leq negl(n)$$

Experiment $MacForge_{\mathcal{A},\Pi}(n)$

Dieses Experiment ist wieder mit einem Gegenspieler $\mathcal{A}$ und einem Challenger:

$MacForge_{\mathcal{A},\Pi}(n)$
$k \leftarrow Gen(1^n)$
$(m', t') \leftarrow \mathcal{A}^{Mac_k(\cdot)}(1^n)$
Sei Q das Set aller Anfragen
if $Vrfy_k(m', t') = 1$ und $m' \notin Q$ return 1
else return 0



Ein Angreifer besitzt hierbei also ein Orakel $Mac_k(\cdot)$, welches zu beliebigen Nachrichten einen Tag erzeugt. Es wird $MacForge_{\mathcal{A},\Pi}(n) = 1$ geschrieben falls $\mathcal{A}$ erfolgreich war.

Bemerkung 4.5.

- Der Angreifer gewinnt nicht, wenn er einfach eine bereits bekannte Nachricht (mit Tag) ausgibt, da dies einen Replay-Angriff modellieren würde
- MACs bieten per Definition keinen Schutz vor Replay Angriffen

Definition 4.6 (MACs anhand von PRF)

Sei F eine Funktion, dann kann damit ein MAC $\Pi_{MAC-PRF}$ fester Länge für Nachrichten der Länge n definiert werden:

- $Gen(1^n)$: Gibt einen echt zufälligen Schlüssel $k \leftarrow \{0,1\}^n$ aus
- $Mac_k(m)$: Gibt einen Tag $t = F_k(m)$ aus. Gilt $|m| \neq |k|$ dann wird nichts ausgegeben
- $Vrfy_k(m,t)$: Gibt 1 zurück genau dann wenn $t = F_k(m)$ gilt. Gilt $|m| \neq |k|$ oder $t \neq F_k(m)$ dann wird 0 ausgegeben

Satz 4.7 (Sicherheit von $\Pi_{MAC-PRF}$)

Ist F aus $\Pi_{MAC-PRF}$ eine PRF, dann ist $\Pi_{MAC-PRF}$ für Nachrichten der Länge n sicher (fälschungssicher).

Beweis 4.8 (Grundlegende Idee des Reduktionsbeweises).

- Grundlegendste Annahme: F ist eine sichere PRF
- Annahme des effizienten Angreifers A gegen $\Pi_{MAC-PRF}$
- Konstruktion des Reduzierer R_D:
  1. R_D bekommt eine PRF. R_D simuliert damit $MacForge_{\mathcal{A},\Pi}(n)$ mit A und gibt das Ergebnis des Experimentes aus, wodurch PRF besser als $negl(n)$ unterschieden wird
  2. R_D bekommt echt zufällige Funktion. Damit ersetzt er die PRF aus $\Pi_{MAC-PRF}$ und erhält $\Pi'_{MAC-PRF}$. Damit simuliert R_D $MacForge_{\mathcal{A},\Pi}(n)$ mit A und gibt das Ergebnis des Experimentes aus, wodurch f **nicht** besser als $negl(n)$ unterschieden wird
- Somit könnte R_D die PRF effizient unterscheiden. Der Widerspruch beweist die Sicherheit

□

Beispiel 4.9.

Sei F_k eine PRF. Sind die folgenden Konstruktionen weiterhin sichere MACs?

- Um die Nachricht $m = m_1||m_2$ zu authentisieren berechnet man den Tag $t = t_1||t_2 = F_k(m_1)||F_k(F_k(m_2))$: Dieser MAC ist unsicher, da ein Angreifer mit dem Orakel einen zweitern MAC für $m_3||m_4$ mit $t_3||t_4$ erzeugen könnte und daraus $m_1||m_4$ mit $t_1||t_4$ erzeugen könnte, wodurch er $MacForge_{\mathcal{A},\Pi}(n)$ gewinnt
- Um die Nachricht $m = m_1||m_2...||m_l$ zu authentisieren berechnet man den Tag $t = \oplus_{i \leq l} F_k(m_i)$: Da XOR-Verwendet wird geht folgendes: Zuerst schickt man $m_1||m_2...m_{l-2}||1^n||1^n$ und erhält damit Tag t. Nun gilt, dass auch $m_1||m_2||...||m_{l-2}$ als Nachricht den gleichen Tag hat, wodurch man $MacForge_{\mathcal{A},\Pi}(n)$ gewinnt

Bemerkung 4.10.

- In beiden Fällen sind universelle Fälschungen (universal forgeries) möglich, d.h. man kann MACs zu beliebigen Nachrichten erstellen
- Oftmals basieren Angriffe auf MACs auf dem Auseinandernehmen und geschicktem Zusammenbauen mehrerer Nachrichten

4.1 CBC-MAC

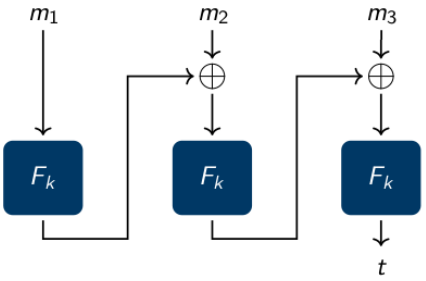
Definition 4.11 (CBC-MAC fester Länge)

Sei F eine Function und sei l eine feste Längenfunktion. Das CBC-MAC Schema damit ist folgendermaßen definiert:

- $Gen(1^n)$: Gibt einen zufällig ausgewählten Schlüssel $k \leftarrow \{0,1\}^n$ zurück
- $Mac_k(m)$: Nimmt als Eingabe eine Nachricht der Länge $l(n) \cdot n$ entgegen. Sei $l(n) = l$ dann folgt:
  - Zerlege m in $m_1, \ldots, m_l$ mit $|m_i| = n$
  - Setze $t_0 = 0^n$ und bereche damit für $i = 1, \ldots, l$:

$$t_i = F_k(t_{i-1} \oplus m_i)$$

  - Gib t_l als tag aus
- $Vrfy_k(m,t)$: Gib 1 zurück genau dann wenn $t = Mac_k(m)$ gilt. Ansonsten gib 0 zurück



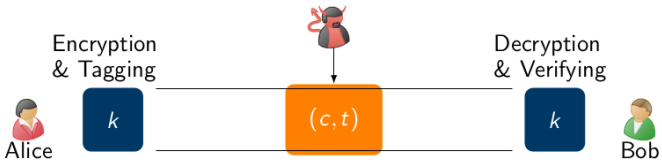
Satz 4.12 (Sicherheit von CBC-MAC fester Länge)

Sei l polynomial. Ist F eine PRF, dann ist die obrige Konstruktion sicher für Nachrichten der festen Länge $l(n) \cdot n$.

Bemerkung 4.13 (Vergleich von CBC-MAC mit CBC-Verschlüsselung).

- Sender und Empfänger müssen im Vornherein die Länge der Nachricht festsetzen
- CBC Verschlüsselung nutzt zufällige IV, CBC-MAC nicht (hier ist IV immer 0^n)
- CBC-MAC mit zufälligen IV wäre unsicher, da ein Angreifer damit den ersten Block beliebig verändern könnte: Man fängt $(IV, (m_1, m_2, \ldots), t)$ ab, ändert m_1 beliebig zu m_1', passt danach IV dementsprechend an (sodass $IV \oplus m_1 = IV' \oplus m_1'$ gilt) und schickt $(IV', (m_1', m_2, \ldots), t)$ weiter
- CBC Verschlüsselung gibt alle Zwischenergebnisse (c_i) aus, CBC-MAC nur das Endergebnis
- CBC-MAC mit Zwischenergebnissen wäre unsicher, da ein Angreifer einfach den hinteren Teil einer Nachricht (und die dazugehörigen Tags) abschneiden und nur den vorderen Teil mit validen Tags weiterschicken könnte

4.2 Authenticated Encryption



Definition 4.14 (Authentizität)

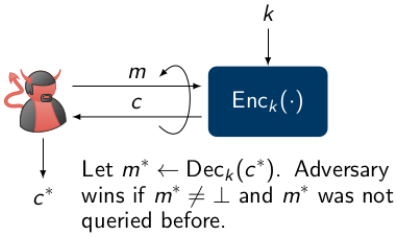
Ein Kryptosystem Π erreicht Authentizität (authentication) wenn für alle probabilistischen polynomial-zeit Angreifer eine vernachlässigbare Funktion $negl(n)$ existiert, sodass gilt:

$$P[Auth_{\mathcal{A},\Pi}(n) = 1] \leq negl(n)$$

Experiment $Auth_{\mathcal{A},\Pi}(n)$

Dieses Experiment ist wieder mit einem Gegenspieler $\mathcal{A}$ und einem Challenger:

$Auth_{\mathcal{A},\Pi}(n)$
$k \leftarrow Gen(1^n)$
$c' \leftarrow \mathcal{A}^{Enc_k(\cdot)}(1^n)$
$m' \leftarrow Dec_k(c')$
Sei Q das Set aller Anfragen
if $m' \neq \perp$ und $m' \notin Q$ return 1
else return 0



Es wird $Auth_{\mathcal{A},\Pi}(n) = 1$ geschrieben falls $\mathcal{A}$ erfolgreich war.

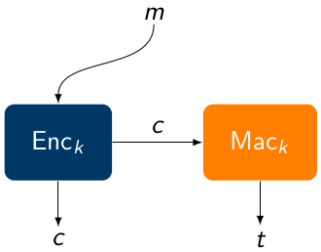
Definition 4.15 (Authentische Verschlüsselung)

Ein symmetrisches Kryptosystem ist eine authentisches Kryptosystem, wenn es CCA-Sicherheit und Authentizität erreicht.

Definition 4.16 (Authentisches Kryptosystem Π_{Auth})

Sei $\Pi = (Gen, Enc, Dec)$ ein symmetrisches Kryptosystem und $\mathcal{M} = (Mac, Vrfy)$ ein MAC, bei denen der Key echt zufällig gewählt wird. Damit kann das Authentisches Kryptosystem Π_{Auth} folgendermaßen definiert werden:

- $Gen'(1^n)$: Wähle zwei Schlüssel $k, k' \leftarrow \{0,1\}^n$ echt zufällig und gebe diese aus
- $Enc'_{k,k'}(m)$: Berechne $c \leftarrow Enc_k(m)$ und $t \leftarrow Mac_{k'}(c)$ und gib (c, t) aus
- $Dec'_{k,k'}(c, t)$: Ist $Vrfy_{k'}(c, t) = 1$ gib $Dec_k(c)$ aus, sonst $\perp$



Bemerkung 4.17.

- Grundlegendes Ziel: Vertraulichkeit und Integrität sicherstellen
- Das zugrundeliegende Prinzip ist Encrypt-then-Authenticate
- Die Variante Authenticate-then-Encrypt könnte nur CPA-sicher, nicht aber CCA-sicher sein (ermöglicht Oracle Padding Angriffe)
- Die parallele Variante Encrypt-and-Authenticate könnte nicht immer Vertraulichkeit gewährleisten (keine EAV-Sicherheit bei deterministischen MAC)
- **Hinweis Tags:** Ein MAC hat einzigartige Tags, wenn es zu jedem key und jeder Nachricht nur einen Tag gibt

Satz 4.18 (Sicherheit von Π_{Auth})

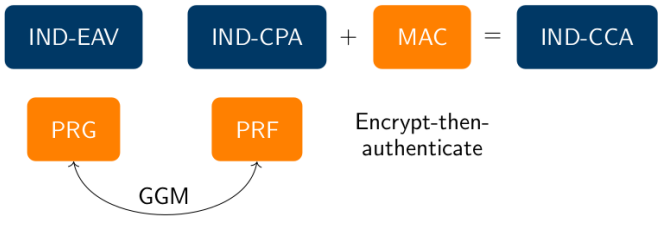
Sei das Π_{Auth} zugrundeliegende Kryptosystem Π CPA-sicher und MAC $\mathcal{M}$ sicher (fälschungssicher) mit einzigartigen Tags, dann erfüllt Π_{Auth} die Eigenschaften eines authentischen Kryptosystems.

Beweis 4.19 (Grundlegende idee).

- Erläuterung valide: Ein Ciphertext c, t gilt als valide, wenn der Verifizierer diesen akzpetiert
- Die Sicherheit von $\mathcal{M}$ impliziert, dass ein Angreifer keinen neuen Tag erzeugen kann, wodurch dieser auch keinen neuen Ciphertext erstellen kann, um das Authentizitätsexperiment zu gewinnen.
- Dies impliziert zum einen die Authentizität, macht aber zum anderen auch das Entschlüsselungsorakel $Dec'_{k,k'}(c, t)$ nutzlos, da abgeänderte bzw. selbst erstellte Ciphertexte durch den MAC nicht valide sind und daher das Entschlüsselungsorakel nicht funktioniert
- Damit reduziert sich die CCA-Sicherheit des Kryptosystems Π_{Auth} auf eine CPA-Sicherheit, welche per Definition sichergestellt ist

□

4.3 Zusammenfassung bisher erreichter Sicherheit



Bemerkung 4.20.

- CCA-Sicherheit geht theoretisch auch ohne MAC, wird im Allgemeinen aber nicht gemacht

5 Hash-Funktionen

Definition 5.1 (Hash-Funktion)

Eine hash-Funktion besteht aus zwei probabilistischen polynomialzeit Algorithmen (Gen, H):
- $Gen(1^n)$: Erzeugt einen Schlüssel s
- $H(x)$: Es existiert ein Polynom l, sodass H eine Eingabe $x \in \{0,1\}^*$ zu der Ausgabe $H^s(x) \in \{0,1\}^{l(n)}$ verarbeitet

Bemerkung 5.2.
- Wenn die Länge des Inputs einer Hashfunktion festgesetzt ist, nennt man es Hash-Funktionen fester Länge (fixed length hash function)
- Im Allgemeinen muss der Schlüssel s kein Geheimnis sein
- In der Praxis sind Hash-Funktionen meistens ohne Schlüssel und daher fest definiert. Dennoch sind diese praktisch kollisionsresistent

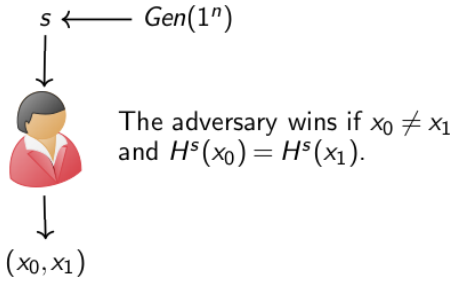
Definition 5.3 (Kollisionsresistenz)

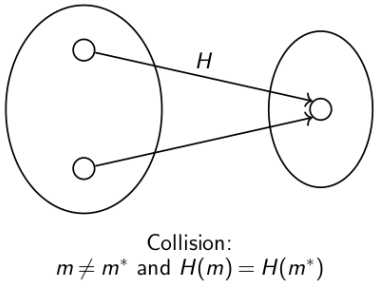
Eine Hash-Funktion Π ist kollisionsresistent, wenn für alle probabilistischen polynomialzeit Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion existiert, sodass gilt:

$$P[HashColl_{\mathcal{A},\Pi}(n) = 1] \leq negl(n)$$

Experiment $PrivK^{cca}_{\mathcal{A},\Pi}(n)$

Dieses Experimant ist wieder mit einem Gegenspieler $\mathcal{A}$ und einem Challenger:





$HashColl_{\mathcal{A},\Pi}(n)$
$s \leftarrow Gen(1^n)$
$(x_0, x_1) \leftarrow \mathcal{A}(s)$
if $x_0 \neq x_1$ and $H^s(x_0) = H^s(x_1)$ return 1
else return 0

Es wird $HashColl_{\mathcal{A},\Pi}(n) = 1$ geschrieben falls de Ausgabe des Experimentes 1 ist, was impliziert, dass $\mathcal{A}$ erfolgreich war und eine Kollision gefunden hat.

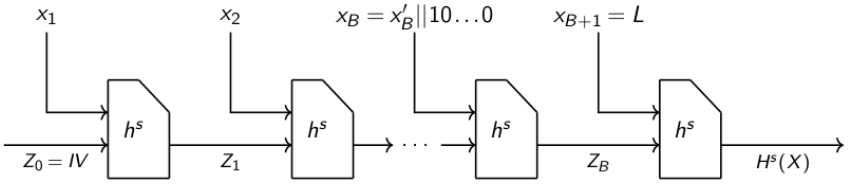
Bemerkung 5.4.
- Dadurch, dass der Gegenspieler $\mathcal{A}$ den Schlüssel s erhält, kann dieser die Hashfunktion simulieren. Er hat also in gewisser Weise ein Verschlüsselungsorakel
- Da Hash Funktionen meist größere Eingaben auf kleinere Ausgaben abbilden, sind diese eine sujektive Funktion und Kollisionen sind möglich. Solche Hash-Funktionen bezeichnet man auch als komprimierend
- Damit die Hash Funktion also sicher ist, muss das Finden von Kollisionen vernachlässigbar wahrscheinlich sein

Beispiel 5.5. (Kombinieren von Hash-Funktionen)

Seien $H_0 : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$ und $H_1 : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$ zwei Hashfunktionen.
- Ist $H^{s0\|s1}(m) = H_0^{s0}(H_1^{s1}(m))$ kollisionsresistent, wenn nur eine Hashfunktion kollisionsresistent ist? Nein: Ist H_1^{s1} nicht kollisionsresistent, dann kann man zwei m finden, für die auch $H_0 : \{0,1\}^n$ eine Kollision erzeugt

- Ist $H^{s0||s1}(m) = H_0^{s0}(m)||H_1^{s1}(m)$ kollisionsresistent, wenn nur eine Hashfunktion kollisionsresistent ist? Ja: Ein Angreifer müsste für H_0^{s0} und H_1^{s1} eine Kollision finden, damit $H^{s0||s1}(m)$ eine Kollision produziert. Da aber eine davon kollisionsresistent ist, ist dies nicht möglich
- Sei H_0 kollisionsresistent. Ist $H(m) = m_0||H_0(m_1)$ mit $m = m_0||m_1$ und $|m_0| = |m_1|$ auch kollisionsresistent? Ja: Ein Angreifer müsste wieder die kollisionsresistenz von H_0 brechen umd $H(m)$ zu brechen

5.1 Merkle Damgard Transformation



Definition 5.6 (Merkle Damgard Transformation)

Sei (Gen,h) mit $h^s : \{0,1\}^{2n} \to \{0,1\}^n$ eine Hashfunktion. Die Merkle Damngard Transformation MDT (Gen, H) ist dann folgendermaßen definiert:

- *Gen*: Bleibt unverändert wie bei h
- H^s: Für eine Eingabe $x \in \{0,1\}^*$ mit Länge $L < 2^b$ berechne:
  1. Setze $B = \lceil \frac{L}{n} \rceil$ (Anzahl der Blöcke in die x zerzeilt wird)
  2. Padde x, damit dessen Länge ein Vielfaches von n ist
  3. Zerlegt x in n-Bit größe Blocke $\Rightarrow x_1, \ldots, x_B$
  4. Setze $x_{B+1} = L$ wobei L mit n bits codiert wird
  5. Setze $U_0 = 0^n$
  6. Berechne für $i = 1, \ldots, B+1$: $Z_i = h^s(z_{i-1}||x_i)$
  7. Gib Z_{B+1} aus

Bemerkung 5.7.

- Die Merkle Damgard Transformation MDT nutzt eine Hash-Funktion fester Länge um damit beliebig lange Eingaben hashen zu können
- Die zugrundeliegende Hashfunktion muss komprimierend sein
- Zur Vereinfachung wurde hier die Größe habliert, aber auch andere Skalierungen sind (bei Anpassung des Systems) möglich

Satz 5.8 (Kollisionsresistenz von MDT)

Ist die der Merkle Damgard Transformation zugrundeliegende Hashfunktion kollisionsresistent, so ist die Merkle Damgard Tranformation auch kollisionsresistent

Beweis 5.9 (Grundidee).

- Findet man eine Kollision bei MDT, so kann man daraus eine Kollision für die zugrundeliegende Hash-Funktion bilden
- Mittels eines Reduktionsbeweises kann also gezeigt werden, dass die Sicherheit der Hash-Funktion die Sicherheit von MDT impliziert

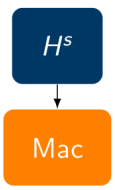
□

5.2 MACs anhand von Hash-Funktionen

Definition 5.10 (Hash and MAC)

Sei $\Pi = (Mac, Vrfy)$ ein MAC fester Länge $l(n)$ und sei $\Pi_H(Gen_h, H)$ eine Hash Funktion mit Ausgabengröße $l(n)$. Ein MAC $\Pi = (Gen', Mac', Vrfy')$ für beliebig lange Nachrichten kann folgendermaßen definiert werden:

- $Gen'(1^n)$: Wähle einen echt zufälligen Schlüssel $k \leftarrow \{0,1\}^n$ und berechne $s \leftarrow Gen_H(1^n)$. Die Ausgabe ist der Schlüssel $k' = (k, s)$
- $Mac'(k', m)$: Berechne aus der Eingabe den Tag $t \leftarrow MAC_k(H^s(m))$
- $Vrfy'(k', m, t)$: Gib 1 aus, genau dann wenn $Vrfy_k(H^s(m), t) = 1$



Bemerkung 5.11.

- Wenn der zugrundeliegende MAC sicher und die hash-Funktion kollisionsresistent ist, dann ist auch Hash and MAC ein sicherer MAC für beliebig lange Nachrichten
- Frage: Kann man auch nur mit Hashfunktionen MACs bauen?
- Antwort: Ja, aber die erste Intuition $H^s(k||m)$ ist unsicher, wenn H auf MDT aufbaut. Dennoch kann man mittels einer Abwandlung von MDT (mit Hash and MAC als Grundidee) sichere MACs anhand von Hashing bauen. Diesen Ansatz nennt man HMAC
- HMAC-Konstruktionen sind heutzutage Standard, da sie (meist) sicher und zudem effizient sind

5.3 Geburtstagsangriff

Bemerkung 5.12.

- Basiert auf dem Geburtstagsparadox: Damit unter einer Menge an Leute die Wahrscheinlichkeit größer als 50% ist, dass zwei Leute am selben Tag Geburtstag haben, muss die Gruppe nur aus 23 oder mehr Personen bestehen. Dies widerspricht einfacher Intuition
- Diese Idee kann man auf den Brute Force Angriff zum Knacken von Kollisionsresistenz anwenden
- Dadurch entsteht eine höhere Wslk. eine Kollision zu finden, als bei Brute Force: Dank des Geburtstagangriffes entsteht bei einer Hashfunktion mit Ausgabelänge l bei $q \approx 2^{l/2}$ Eingaben mit mehr als 50% Wslk. eine Kollision
- Fazit: Die Ausgabelänge der Hashfunktion bestimmt die Sicherheit und muss in der Praxis entsprechend hoch gewählt werden (Sicherheit von 2^{80} benötigt 160 bit Output anstatt 80)

5.4 Random Oracle Model

Definition 5.13 (Zufallsorakel)

- Es gibt einen Gnom mit einem großem Buch und ein paar D&D Würfeln, der in einer schwarzen Box lebt
- Jeder (Angreifer oder Challenger) kann der Box ein Paar Daten als Input geben
- Bekommt der Gnom Daten die er vorher noch nie gesehen hat, generiert er einen echt zufälligen neuen Output mit seinen Würfeln. Zudem schreibt der Gnom den Input und den dazugehörigen Output in sein Buch
- Wenn der Gnom einen bereits bekannten Input ekommt, schaut er in sein Buch und gibt den dazugehörigen, bereits erzeugten, Output aus

Bemerkung 5.14.

- **Problem:** Kollisionsresistenz reicht oft nicht aus, um die Sicherheit eines Kryptosystems zu beweisen
- **Lösung:** Statt der Hashfunktion nimmt man die idealisierte Version, also das Zufallsorakel (da es besser ist lieber einen ungenaue als garkeinen Beweis zu haben)
- **Hinweis:** Die Kommunikation mit dem Zufallsorakel ist privat, d.h. ein Challenger weiß nicht, dass ein Angreifer das Orakel verwendet
- **Folgerung:** Es eignet sich also an, beim Beweis eines Kryptosystems mit Hashfunktion das Random Oracle Model zu verwenden und bei der praktischen Verwendung dieses erst durch die echte Hash-Funktion zu ersetzen
- **ROM und Reduktion:** Bei einem Reduktionsbeweis muss das ROM vom Reduzierer simuliert werden. Dadurch ist der Reduzierer extrem mächtig, da er zum einen die Anfragen kennt und zum anderen die Ausgabe des ROM programmieren kann. Das enstehende Problem ist, dass dies in der echten welt nicht mehr möglich ist
- **Weiteres:** Das ROM kann sowohl als kollisionsfreie Hashfunktion, als auch als PRF und PRG verwendet werden
- **Vorteile des ROM:** Ermöglicht die Konstruktion sehr effizienter Kryptosysteme. Kryptosystem die mit ROM sicher sind, sind in der Praxis auch meist mit einer sicheren Hash-Funktion sicher. Ansosnten kann man auch einfach die Hash-Funktion tauschen
- **Nachteile des ROM:** Das ROM ist kontrovers diskutiert. Es gibt Kryptosystem die sicher mit einem ROM sind, aber unsicher für egal welches Primitiv, durch dass man das ROM ersetzt. Zudem können nicht alle Annahmen des ROM in der echten Welt erfüllt werden

6 Theoretische Konstruktion und Zusammenhänge kryptographischer Primitive

6.1 Einwegfunktionen und Einwegpermutierer

Definition 6.1 (Einwegfunktionen)

Eine Funktion $f : \{0,1\}^* \to \{0,1\}^*$ ist eine Einwegfunktion wenn gilt:

- **Einfach zu berechnen:** Es existiert ein polynomieller Algorithmus um $f(x)$ für alle x zu berechnen
- **Schwierig zu invertieren:** Für jeden probabilistischen polynomialzeit Algorithmus $\mathcal{A}$ existiert eine vernachlässigbare Funktion $negl(n)$ sodass gilt:

$$P[Invert_{\mathcal{A},f}(n) = 1] \leq negl(n)$$

Experiment $Invert_{\mathcal{A},f}(n)$

$Invert_{\mathcal{A},f}(n)$
$x \leftarrow \{0,1\}^*$
$y \leftarrow f(x)$
$x' \leftarrow \mathcal{A}(y)$
if $f(x') = y$ return 1
else return 0

Es wird $Invert_{\mathcal{A},f}(n) = 1$ geschrieben falls die Ausgabe des Experimentes 1 ist, was impliziert, dass $\mathcal{A}$ erfolgreich war.

Bemerkung 6.2.

- **Engisch** One-Way-Funktion (OWF)/ One-Way-Permutation (OWP)
- Mit exponentieller Zeit könnte jede Einwegfunktion geknackt werden (brute force), aber wg. berechenbarer Sicherheit können wir diesen Fall ausschließen
- **Einwegpermutierer:** Das sind längenerhaltende bijektive Einwegfunktionen, also ein Spezialfall
- **Funktionsfamilie:** Die obrige Definition betrachtet eine einzige Funktion mit unendlich großem Werte- und Defintionsbereich. Eine Funktionsfamilie hingegen ist ein Algorithmus mit Generierungs-,Berechnungs- und Validierungsalgorithmus für Funktionen die anhand des Generierungalgorithmus mit festem Werte- und Definitionsbereich erzeugt werden
- **Beispiel:** Man kann annehmen, dass die Faktorisierungsproblem eine Einwegfunktion ist
- **Beispielfrage:** Sei $f(x)$ eine OWF. Ist dann $f'(x\|y) = f(x)\|y$ auch eine OWF? Ja, weil nur 2 Bits verraten werden

6.2 PRG aus OWP anhand von Hardcore Predicates

Definition 6.3 (Hardcore Prädikate)

Eine function $hc : \{0,1\}^* \to \{0,1\}$ ist ein Hardcore Prädikat für eine Funktion f, wenn hc in polynomieller Zeit berechnet werden kann und für alle probabilistischen polynomialzeit Algorithmen $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ existiert, sodass gilt:

$$\Pr_{x \leftarrow \{0,1\}^*}[\mathcal{A}(1^n, f(x)) = hc(x)] \leq \frac{1}{2} + negl(n)$$

Dabei wird die Wahrscheinlichkeit über die echte Zufälligkeit von x und das Zufallsprinzip von hc berechnet.

Bemerkung 6.4.

- **Grundidee:** Harcore Prädikate geben an, dass eine beliebige Funktion Informationen über die Eingabe verschleiert. Ihr Output ist ein einziger boolscher Wert (0 oder 1)
- **Vereinfacht:** Ein Hardcore Prädikat ist eine Funktion die für beliebige x mit $hc(x)$ einfach zu berechnen sind, aber kennt man nur $f(x)$ (und ggf. die Funktion f) so ist $hc(x)$ daraus schwierig zu berechnen
- **Triviales Beispiel:** Für $f(x_1, x_2) = x_1$ ist $hc(x) = x_2$ ein Hardcore Prädikat

- **Einfache aber falsche Idee:** Jedes Bit der Eingabe miteinander zu verXoren ist kein Hardcore Prädikat für alle Funktionen (auch wenn man meinen würde, dass ja mind 1 Bit aus dem Input verschleiert wird): Sei f und g eine Einwegfunktion, dann wäre $f(x) = g(x)\| \oplus_{i=1}^{n} x_i$ ein Gegenbeispiel und weiterhin eine Einwegfunktion
- **Hardcore Prädikate und Einwegfunktionen:** Es gibt zu jeder Einwegfunktion ein Hardcore Prädikat (Es wurde gezeigt, dass aus jeder Einwegfunktion eine Einwegfunktion mit Hardcore Prädikat erzeugt werden kann)

Satz 6.5 (PRG anhand von OWP)

Sei f ein OWP und sei hc ein Hardcore Prädikat für f. Dann ist $G(s) = f(s)\|hc(s)$ ein PRG mit Expansion $l(n) = n + 1$.

Bemerkung 6.6.

- **Beweisidee:** Da f und hc pseudorandom sind, wenn s echt zufällig gewählt wurde, ist auch G pseudorandom
- **PRG anhand von OWF:** Das ist theoretisch möglich, aber der Beweis ist extrem kompliziert

6.3 PRG mit stärkerem Expansionsfaktor und Hybridargumenten

Satz 6.7 (PRG Expansionstheorem)

Ist G ein PRG mit Expansionsfaktor $n + 1$, dann existiert zu einem beliebiges Polynom $poly(n)$ ein PRG G' mit Expansionsfaktor $poly(n)$.

Beweis 6.8 (Grundidee für Expansionsfaktor $n + 2$).

- Zur Konstruktion G' nutzen wir einfach zweimal G: Sei s der Initiale echt zufällige Seed, dann ist $G(s) = m_0\|m_1\|...\|m_{n+1}$. Anschließend nimmt man die ersten n Bits von m und berechnet $G(m_0\|...\|m_n) = x$. Mittels Konkationation erhält man also $G'(s) = x\|m_{n+1}$
- Sei D ein polynomieller Unterscheider für G'. Zuerst werden drei Wahrscheinlichkeitsverteilungsfolgen definiert:
  1. $\{H_n^0\}_{n=1,...}$: Wähle $t_0 \leftarrow \{0,1\}^n$ echtzufällig und gib $G'(t_0)$ aus (normale Ausführung von G') **// Erstes Extrem**
  2. $\{H_n^1\}_{n=1,...}$: Wähle $t_1 \leftarrow \{0,1\}^{n+1}$ echtzufällig und zerlege es in $m_0\|m_1\|...\|m_{n+1}$. Gib $G(m_0\|...\|m_n)\|m_{n+1}$ aus
  3. $\{H_n^2\}_{n=1,...}$: Wähle $t_2 \leftarrow \{0,1\}^{n+2}$ echtzufällig und gib t_2 aus **// zweites Extrem**
- **Schreibweise:** Mit $t_2 \leftarrow H_n^i$ wird die zufällige Erzeugung eines Strings der Länge $n + 2$ anhand der Wahrscheinlichkeitsverteilung H_n^i beschrieben
- Zu zeigen ist, dass die beiden Extrema nicht unterscheidbar sind:

$$\left| \Pr_{s\leftarrow\{0,1\}^n}[D(G'(s)) = 1] - \Pr_{r\leftarrow\{0,1\}^{n+2}}[D(r) = 1] \right| = \left| \Pr_{t_2\leftarrow H_n^0}[D(t_2) = 1] - \Pr_{t_2\leftarrow H_n^2}[D(t_2) = 1] \right| \leq negl(n)$$

- Dafür müssen alle dazwischenliegenden Übergänge mittels Reduktion bewiesen werden:
  1. Zuerst wird der erste Übergang betrachtet:

$$| \Pr_{t_2\leftarrow H_n^0}[D(t_2) = 1] - \Pr_{t_2\leftarrow H_n^1}[D(t_2) = 1] | \leq negl'(n)$$

Nehme man an D sei effizient. Dann kann man D' konstruieren, welcher einen String t' der Länge $n + 1$ bekommt, diesen zerlegt $t'_n\|t'_{n+1}$ und damit $t_2 = G(t')\|t'_{n+1}$ berechnet und das Ergebnis an D gibt. D' gibt dann das von D aus. **1. Fall:** D' bekommt einen echt zufälligen String. Es entsteht H_n^1. Weder D noch D' können effizient unterscheiden:

$$\Pr_{t_1\leftarrow\{0,1\}^{n+1}}[D'(t_1) = 1] = \Pr_{t_2\leftarrow H_n^1}[D(t_2) = 1] \approx \frac{1}{2}$$

2. Fall: D' bekommt einen von G erzeugten String. Mit den Berechnungen von D' bekommt D eine Eingabe die auch von G' kommen könnte. Es entsteht H_n^0. Daher könnte D das richtig entscheiden und D' könnte den PRG brechen. Das ist ein Widerspruch und es gilt:

$$\Pr_{s\leftarrow\{0,1\}^n}[D'(G(s)) = 1] = \Pr_{t_2\leftarrow H_n^0}[D(t_2) = 1] \approx \frac{1}{2}$$

Damit ist die ursprüngliche Aussage bewiesen

2. Als nächstes wird folgender Übergang betrachtet:

$$|\ \Pr_{t_2 \leftarrow H_n^1}[D(t_2) = 1] - \Pr_{t_2 \leftarrow H_n^2}[D(t_2) = 1]\ | \leq negl''(n)$$

Die Argumentation ist ähnlich, wieder erzeugt man einen Reduzierer und führt dies zu einem Widerspruch

- Zusammengefasst erhält man also:

$$|\ \Pr_{s \leftarrow \{0,1\}^n}[D(G'(s)) = 1] - \Pr_{r \leftarrow \{0,1\}^{n+2}}[D(r) = 1]\ | = |\ \Pr_{t_2 \leftarrow H_n^0}[D(t_2) = 1] - \Pr_{t_2 \leftarrow H_n^2}[D(t_2) = 1]\ | \leq negl(n)$$

$$\leq |\ \Pr_{t_2 \leftarrow H_n^0}[D(t_2) = 1] - \Pr_{t_2 \leftarrow H_n^1}[D(t_2) = 1]\ | + |\ \Pr_{t_2 \leftarrow H_n^1}[D(t_2) = 1] - \Pr_{t_2 \leftarrow H_n^2}[D(t_2) = 1]\ | \leq negl'(n) + negl''(n)$$

- Damit ist anhand der pseudorandom Eigenschaft von G bewiesen, dass auch G' ein PRG ist

□

Bemerkung 6.9.

- Dieser Spezialfall bildet die Grundlage für Hybridargumente
- Hybridargumente sind praktisch, wenn ein kryptographisches Primitiv mehrfach oder zyklisch verwendet wird
- Mittels dem Hybridargument werden alle Übergänge (Anwendungen eines Primitives aus Ergebnissen vorheriger Primitivanwendungen oder Initialinput) gleichzeitig bewiesen
- Sind nämlich alle Übergänge sicher (vernachlässigbar unterscheidbar), dann sind auch die Extrema sicher (vernachässigbar unterscheidbar) und das Kryptosystem ist sicher
- **Die Grundidee:**
  1. Definiere Konstruktion G' basierend auf einem sicheren Primitiv G
  2. Definiere D als effizienter Angreifer für einen beliebigen Übergang von G'
  3. Definiere die einzelnen Übergänge formal im Bezug auf die beweisbare Sicherheit (z.b. Wahrscheinlichkeitsverteilungsfolgen)
  4. Stelle die beiden Extrema des Hybridbeweises (z.b. Verteilung der Ausgabe G' und einer echt zufälligen Zeichenkette) dar
  5. Definiere anhand der Extrema, was zu beweisen ist
  6. Beweise gleichzeitig alle Übergänge von G':
     (a) Definiere einen Unterscheider D', der das grundlegende Primitiv G unterscheiden soll, indem er dieses zu einer zufälligen Instanz von D umwandelt
     (b) Analysiere D' mittels Fallunterscheidung (Zufallseingabe wird zu H_j, PRG wird zu H_{j-1})
     (c) Gäbe es also einen Übergang, den D effizient unterscheiden könnte, kann D' effizient das Primitiv unterscheiden
  7. Fasse Ergebnisse zusammen und lege dar, dass Sicherheit von G auch die Sicherheit von G' impliziert

Beweis 6.10 (Mit Hybridargumenten).

- Konstruktion von G' mit Expansionsfaktor $n + p(n)$ für ein Polynom p und die Eingabe $s \in \{0,1\}^n$:
  1. Setze $t_0 = s$. Berechne für $i = 1, \ldots, p(n)$:
     (a) Seien s_{i-1} die ersten n Bits von t_{i-1} und σ_{i-1} der Rest
     (b) Berechne $t_i = G(s_{i-1})||\sigma_{i-1}$
  2. Gib $t_{p(n)}$ aus
- $\mathcal{D}$ wird als Unterscheider für G' festgesetzt
- Nun können wir die Wahrscheinlichkeitsverteilungsfolgen nicht mehr alle einzeln beschreiben. Daher werden nun die $p(n)$-vielen Verteilungen folgendermaßen definiert: Für $0 \leq j \leq p(n)$ sei H_n^j Wahrscheinlichkeitsverteilungsfolge von Zeichenketten der Länge $n + p(n)$ sodass:

$$H_n^j:\ \text{Wähle } t_j \leftarrow \{0,1\}^{n+j} \text{ zufällig dann berechne die j+1te Iteration von G' und gib } t_{p(n)} \text{ aus}$$

- Besonders interessant dabei sind folgende Verteilungen (Extrema des Hybridargumentes):
  - H_n^0, welches die finale Ausgabe von $G'(s)$ ist
  - $H_n^{p(n)}$, was eine echt zufällige Zeichenkette der Länge $n + p(n)$ ist
- Zu Zeigen ist wieder, dass die Extreme ununterscheidbar sind:

$$\left| \Pr_{s \leftarrow \{0,1\}^n}[D(G'(s)) = 1] - \Pr_{r \leftarrow \{0,1\}^{n+p(n)}}[D(r) = 1] \right| = \left| \Pr_{t \leftarrow H_n^0}[D(t) = 1] - \Pr_{t \leftarrow H_n^{p(n)}}[D(t) = 1] \right|$$

- Als nächstes wird der Unterscheider $\mathcal{D}'$ genauer definiert:
  1. Wähle ein $j \leftarrow \{1, ..., p(n)\}$ und ein $\sigma'_j \leftarrow \{0,1\}^{j-1}$ (bei j=1 ist σ leer)
  2. Erzeuge $t_j = w \| \sigma'_j$. Berechne G' beginnend bei Iteration $j+1$ und erzeuge damit $t_{p(n)} \in \{0,1\}^{n+p(n)}$
  3. Gib $\mathcal{D}(t_{p(n)})$ aus
- Setze n fest. $\mathcal{D}'$ wählt zufällig $j = j'$. Daraus ergeben sich 2 Fälle:
  1. Wenn w echt zufällig ist, dann ist $t_{j'}$ auch echt zufällig und es gilt:

$$\Pr_{w \leftarrow \{0,1\}^{n+1}}[D'(w) = 1 | j = j'] = \Pr_{t \leftarrow H_n^{j'}}[D(t) = 1]$$

  Da die j mit gleicher Wslk. ausgewählt werden gilt:

$$\Pr_{w \leftarrow \{0,1\}^{n+1}}[D'(w) = 1] = \frac{1}{p(n)} \sum_{j'=1}^{p(n)} \Pr_{w \leftarrow \{0,1\}^{n+1}}[D'(w) = 1 | j = j'] = \frac{1}{p(n)} \sum_{j'=1}^{p(n)} \Pr_{t \leftarrow H_n^{j'}}[D(t) = 1]$$

  2. Sei $w = G(s)$ für ein echt zufälliges $s \in \{0,1\}^n$. Sei $t_{j'-1} = s \| \sigma'_{j'}$. Da $t_{j'-1}$ echt zufällig ist, ist die Verteilung von $t = t_p(n)$ identisch zu $H_n^{j'-1}$:

$$\Pr_{s \leftarrow \{0,1\}^n}[D'(G(s)) = 1 | j = j'] = \Pr_{t \leftarrow H_n^{j'-1}}[D(t) = 1]$$

  Daraus folgt:

$$\Pr_{s \leftarrow \{0,1\}^n}[D'(G(s)) = 1] = \frac{1}{p(n)} \sum_{j'=1}^{p(n)} \Pr_{s \leftarrow \{0,1\}^n}[D'(G(s)) = 1 | j = j'] = \frac{1}{p(n)} \sum_{j'=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j'}}[D(t) = 1]$$

- Zusammengefasst ergibt sich also:

$$\left| \Pr_{s \leftarrow \{0,1\}^n}[D'(G(s)) = 1] - \Pr_{w \leftarrow \{0,1\}^{n+1}}[D'(w) = 1] \right|$$

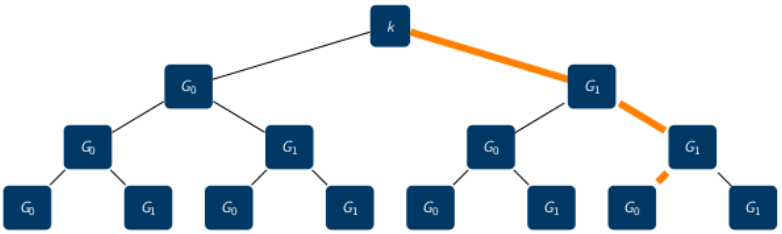
$$= \frac{1}{p(n)} \left| \sum_{j'=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j'}}[D(t) = 1] - \sum_{j'=1}^{p(n)} \Pr_{t \leftarrow H_n^{j'}}[D(t) = 1] \right|$$

$$= \frac{1}{p(n)} \left| \Pr_{t \leftarrow H_n^0}[D(t) = 1] - \Pr_{t \leftarrow H_n^{p(n)}}[D(t) = 1] \right|$$

- Nachdem angenommen wurde, dass G ein sicherer PRG ist, kann der erste Teil der Gleichung als vernachlässigbar ausgewertet werden, was impliziert das die zu Zeigende Aussagen auch vernachlässigbar ist. Das wiederum impliziert die Sicherheit von G'.

□

6.4 Mit Bäumen von PRG zu PRF



$$F_k(110) = G_0(G_1(G_1(k)))$$

Definition 6.11 (Goldreich-Goldwasser-Micali Konstruktion)

Sei $G : \{0,1\}^n \to \{0,1\}^{2n}$ ein PRG mit Expansionsfaktor $2n$ der folgendermaßen definiert ist:

$$G(x) = G_0(x) \| G_1(x) \text{ mit } G_b(x) \in \{0,1\}^n$$

Damit wird eine längenerhaltende PRF $F_k(a) : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ mit $a = a_1, ..., a_n$, wobei a_i ein

Bit aus a entspricht, definiert:

$$F_k(a) = G_{a_n}(G_{a_{n-1}}(\ldots G_{a_1}(k)\ldots))$$

Bemerkung 6.12.

- **Beispiele**: $F_k(10) = G_0(G_1(k)), F_k(11) = G_1(G_1(k))$
- $G_b(x)$ definiert die erste bzw. zweite Hälfte von $G(x)$

Satz 6.13 (PRF aus PRG)

Wenn G ein sicherer PRG mit Expansionsfaktor $2n$ ist, dann ist ein F, welches durch die Goldreich-Goldwasser-Micali (GGM) Konstruktion erzeugt wurde eine sichere PRF

Beweis 6.14 (Grundidee).

- Wahrscheinlichkeitsverteilungsfolgen:

  H_n^j : Wähle $t_j \leftarrow \{0,1\}^{n+j}$ zufällig dann berechne die j+1te Iteration von GGM und gib $GGM(t_j)$ aus

- Unterscheider D':
  1. Input: pseudorandom oder echt zufälliges s
  2. Wähle zufälliges $j \leftarrow \{1,\ldots,|a|\}$
  3. Simuliere das PRF-Experiment mit D, indem man s als Key verwendet und nur $|a| - j$ viele Iteration von GGM berechnet
  4. Gibt aus was D am Ende ausgibt

□

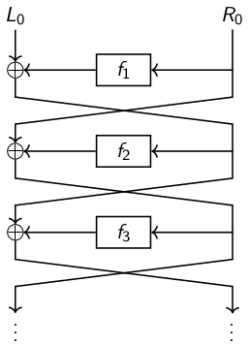
6.5 Feistel Netzwerke für PRP

Definition 6.15 (Feistel Netzwerk)

Sei $f_{k_i} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ eine längenerhaltende Funktion abhängig von einem Schlüssel k_i. Dann ist eine schlüsselabhängige Permutation $F^r : \{0,1\}^{2n} \times \{0,1\}^{2n} \to \{0,1\}^{2n}$ mit r Feistel Runden folgendermaßen definiert:

- **Input:** Ein Schlüssel $k = (k_0, \ldots, k_r)$ mit $|k_i| = n$ als Schlüssel die aus k erzeugt werden. Zudem ein Eingabestring $x \in \{0,1\}^{2n}$ der $x = (L_0, R_0)$ mit $|L_0| = R_0| = n$ initialisiert
- **Berechnung:**

$$L_{i+1} = R_i \text{ und } R_{i+1} = L_i \oplus f_{k_i}(R_i)$$



Bemerkung 6.16.

- **Grundidee:** Konstruktion einer invertierbaren Funktion mittels nicht invertierbarer Primitive
- **Invertierbarkeit:** Auch ohne die einzelnen Funktionen invertieren zu können, kann das Feistel Netzwerk invertiert werden. Dafür berechnet man rückwärts:

$$R_{i-1} = L_i \text{ und } L_{i-1} = R_i \oplus f_{k_i}(R_{i-1})$$

- Somit erfüllt das Feistel Netzwerk alle Eigenschaften eines Permutierers
- Mit ein oder zwei Runden ist das Netzwerk noch nicht pseudorandom (Grund: Setze man R_0 fest und sei $L_0 = 0^n$ und $L_0' = 1^n$ und erhalte (L_2, R_2) und L_2', R_2'. Bei der Feistelkonstruktion gilt immer: $L_0 \oplus L_2 = L_0' \oplus L_2'$. Dadurch ist das Schemata unterscheidbar)
- Mit drei Runden, also F^3, wird der einfachste PRP mit beweisbarer pseudorandom Eigenschaft erzeugt
- Mit vier Runden, also F^4, kann man schon einen starken PRP erzeugen

6.6 Rückrichtung: OWF aus PRG

Satz 6.17 (OWF aus PRG)

Wenn PRG existieren, dann existieren auch OWF.

Bemerkung 6.18.

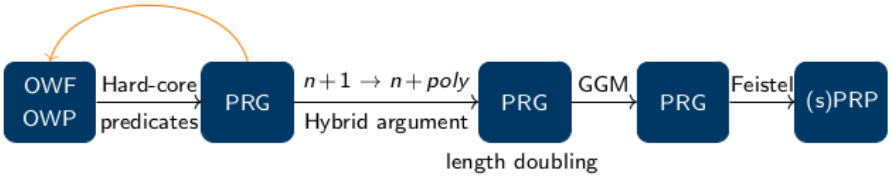
- Beweis nicht wichtig
- Grundidee: Sei G ein PRG. G zu invertieren wäre Äquivalent dazu G zu unterscheiden, was der Definition von G widerspricht

Satz 6.19 (OWF aus PKE)

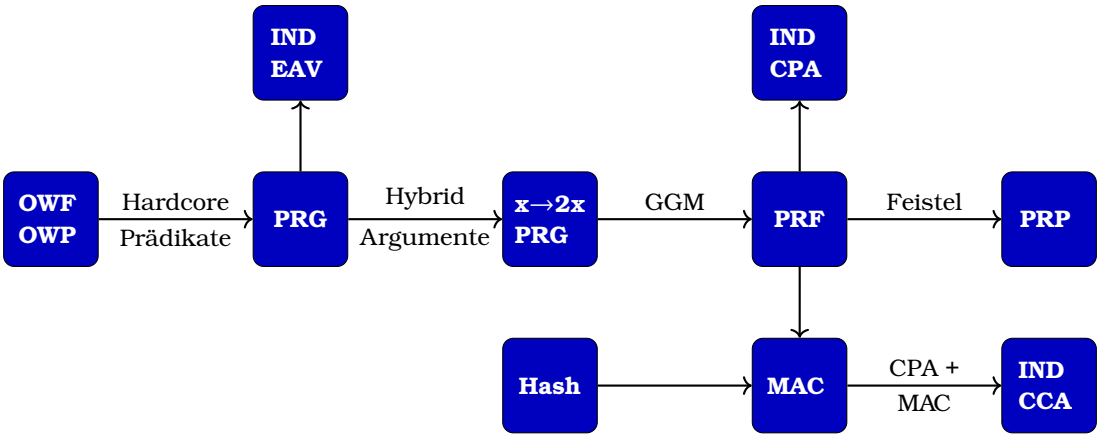
Existieren EAV-sichere symmetrische Kryptoverfahren, die Nachrichten von doppelter Länge des Schlüssels verschlüsseln können, dann existieren auch Einwegfunktionen.

Bemerkung 6.20.

- Beweis nicht wichtig



7 Überblick symmetrischer Kryptographie



8 Praktische kryptographische Primitive*

8.1 Grundlegendes

Stream-Cipher vs. Block-Cipher

- **Stream-Cipher:** Verschlüsselt jedes Element des Klartextes einzeln (’Pro Element ein Schlüssel’, Verschlüsselungen beliebiger Länge, Beispiel PRG-Verschlüsselung)
- **Block-Cipher:** Zerlegt den Klartext in Blöcke und verschlüsselte die Blöcke einzeln. Ggf. werden die Blöcke zusätzlich noch abhängig von vorherigen Blöcken verschlüsselt, bedingt durch die Definition des verwendeten Operationsmodus (Verschlüsselungen fester Länge, Beispiel DES, AES)

Linear-Feedback Shift Register

- Historisch verwendete Hardware die als PRG fungiert(e)
- Sind keine kryptographisch sicheren PRG.

Substitutions-Permutations Netzwerk SPN

- Konfusion: Beziehung zw. Klartext und Chiffretext verschleiern
- Diffusion: Bit aus dem Klartext soll viele Bits aus dem Chiffretext beeinflussen
- SPN versuchen Konfusions und Diffusions zu implementieren
- Grundidee einer Runde für input x:
  1. x’ = x mit Schlüssel verrechnen
  2. x” = x’ mit Substitutionsboxen verrechnen
  3. Bits von x” permutieren
- Für viel Diffusion und Konfusion sind mehrere Runden nötig (Avalanche Effekt)
- SPN können einfach invertiert werden (wenn man den Schlüssel kennt), weshalb sie sich für kryptographische Anwendungen eignen

8.2 Blockchiffren

Data Encryption Standard DES

- **Funktionsweise:** 16-Runden Feistel-Netzwerk mit 64 bit Blöcken und 56 Bit Schlüssellänge
- **Unsicher:** 56 Bits sind zu wenig. Mit Brute Force lösbar
- **Doppeltes DES:** Verbesserung durch zweiten Schlüssel und doppelte Anwendung des DES. Die Sicherheit bleibt wie bei DES, da ein Angreifer, wenn er ein Input/Output Paar kennt mit Meet-In-The-Middle die beiden Schlüssel so schnell wie bei DES berechnen kann
- **Meet-In-the-Middle:** Man durchsucht den Schlüsselraum mit Bruteforce, indem man den die erste Verschlüsselung und die Inverse der zweiten Verschlüsselung berechnet. Man speichert sich das Ergebnis beider Rechnungen. Erzeugt man dabei zwei identische Zwischenergebnisse so hat man beide Schlüssel gefunden
- **Dreifach DES:** Gilt heute als sicher, bietet aber nur etwa 112-bit statt 168-bit Sicherheit, da weiterhin das Meet-In-the-Middle-Prinzip auf einen Übergang angewendet werden kann

Advanced Encryption Standard

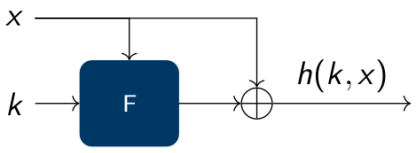
- Entwickelt worden, da DES absehbar unsicher wurde
- AES hat 128-,192- oder 256-Bit Lange Schlüssel
- Basiert auf einem Substitutions-Permutations Netzwerk
- **Grundprinzip**:
  - **Vorbereitung**: Alles initialisieren und Block einmal mit Schlüssel verrechnen
  - **Verschlüsselungsablauf je Runde**:
    - SubBytes: Substitutions mittels Substitutionsbox S-Box
    - ShiftRows: Zeilen permutieren
    - MixColumns: 4 festgelegte Bytes mittels Galois Körper permutieren
    - Schlüssel mit Block verrechnen
    - Abschließend: SubBytes, ShiftRows, Schlüssel mit Block verrechnen
  - **Entschlüsselung**: Verschlüsselungsschritte invers berechnen
- **Sicher:** Die große Schlüssellänge macht AES bis heute sicher

8.3 Hash-Algorithmen

8.3.1 Hashfunktionen aus PRP

Definition 8.1 (Hash aus PRP)

Sei $F : \{0,1\}^n times \{0,1\}^l \to \{0,1\}^l$ eine Funktion, dann ist $h(k,x) = F_k(x) \oplus x$ eine Kompressionsfunktion



8.3.2 Bekannte Hashfunktionen

Bekannte Hashing-Verfahren

- **MD5**: 128 Bit Output. Gilt heute als unsicher
- **SHA1**: 160 Bit Output. Gilt heute als unsicher
- **SHA2**: 224, 256, 384 oder 512 Bit Output. Gilt mehr oder weniger als sicher
- **SHA3**: 256 oder 512 Bit Output. Gilt definitiv als sicher

Teil II
Assymetrische Verschlüsselung

9 Primzahlen und Teilbarkeit

Definition 9.1 (Division mit Rest)

Seien $a, b \in \mathbb{N}$. Teilt man diese erhält man einen **Quotienten** $q \in \mathbb{N}$ und einen **Rest** $r \in \mathbb{N}$:

$$\frac{a}{b} = q \text{ Rest } r$$

Mit dem modulo-Operator kann dies umgeschrieben werden zu: $a \equiv r \mod b$. Dadurch entsteht eine **Kongruenzgleichung**.

Definition 9.2 (Teilbarkeit)

- Seien $a, b, c \in \mathbb{N}$. Dann gilt: $a \mid b \Leftrightarrow \exists c : b = a \cdot c$
- D.h. a teilt also b, wenn b ein Vielfaches von a ist
- Insbesonders gilt: $1 \mid a \ -1 \mid a \ a \mid a \ a \mid 0$
- D.h. Teilbarkeit hängt nicht vom Vorzeichen ab

Definition 9.3 (ggT und kgV)

- ggT(a,b) ist der größte gemeinsame Teiler aus den Teilermengen von a und b
- kgV(a,b) ist das kleinste gemeinsame Vielfache zweier Zahlen

Definition 9.4 (Kongruenzgleichungen und Teilbarkeit)

- Seien $a, b, m \in \mathbb{Z}$. Damit kann die Kongruenzgleichung $a \equiv b \ mod \ m$ gebildet werden
- Im Bezug auf die Teilbarkeit gilt: $a \equiv b \ mod \ m \Leftrightarrow m \mid a - b$
- Die **Kongruenzklasse** a mod m wird als $\overline{a}^{mod\ m}$ geschrieben
- Es gilt $\overline{a}^{mod\ m} = \overline{a}^{mod\ m} \Leftrightarrow a \equiv b \mod m \Leftrightarrow m \mid a - b$

Algorithmus 9.5 (Euklidischer Algorithmus).
Mit dem euklidischen Algorithmus kann ggT(a,b) mit $a, b \in \mathbb{N}$ berechnet werden. Zur Vereinfachung sollte $a > b$ gelten:

$$\begin{array}{cccccc} a & \% & b & = & q_0 & \text{Rest } r_0 \\ b & \% & q_0 & = & q_1 & \text{Rest } r_1 \\ q_0 & \% & q_0 1 & = & q_2 & \text{Rest } r_2 \\ & & & \dots & & \\ q_{n-2} & \% & q_{n-1} & = & q_n & \text{Rest } r_n = 0 \end{array}$$

Abschließend gilt ggT(a,b) = q_n.

Algorithmus 9.6 (Erweiterter Euklidischer Algorithmus).
Mit dem erweiteren euklidischen Algorithmus können folgende Gleichungen gelöst werden:

$$ggT(a, b) = x \cdot a + y \cdot b$$

Dabei gilt $a, b, x, y \in \mathbb{N}$. Dieser Algorithmus ist besonders wichtig um die Inversen in Restklassenkörpern zu berechnen. Er funktioniert wie folgt:

Zeile	Reste R	Quotienten Q	x-Spalte	y-Spalte
0	a	-	1	0
1	b	-	0	1
i	$R_{i-2} \ mod \ R_{i-1}$	$\lfloor R_{i-2}/R_{i-1} \rfloor$	$x_{i-2} - (Q_i \cdot x_{i-1})$	$y_{i-2} - (Q_i \cdot y_{i-1})$
n	$ggT(a, b)$	$\lfloor R_{n-2}/R_{n-1} \rfloor$	x	y
$n+1$	0	$\lfloor R_{n-1}/R_n \rfloor$	-	-

10 Ein Hauch von Algebra

Definition 10.1 (Abel'sche Gruppe)

Gegeben sei eine Menge M mit den Elementen $a, b, c, e \in M$ und darüber eine Verknüpfung '$\circ$'. Dann hat eine Abel'sche Gruppe folgende Eigenschaften:

- Abgeschlossenheit: $a \circ b = c \in M$
- Assoziativgesetz: $a \circ (b \circ c) = (a \circ b) \circ c$
- Kommutativgesetz: $a \circ b = b \circ a$
- Neutrales Element: $a \circ e = e \circ a = a$
- Inverses Element: $\forall a \in M \; \exists b \in M : a \circ b = b \circ a = e$

Fehlt hierbei das Kommutativgesetz, bezeichnet man $(M, \circ)$ als Gruppe.

Bemerkung 10.2.

- Hat M endlich viele Elemente, so ist es eine endliche Gruppe und $|M|$ bzw. $\#M$ gibt die Ordnung (Anzahl an Element) der Gruppe an
- **Beispiel:** $\mathbb{Z}(+)$ ist eine Gruppe, $\mathbb{Z}(\cdot)$ nicht (Inverse fehlt)
- Gruppen Modulo N: $\mathbb{Z}_N(+) = \{0, 1, ..., N-1\}$ ist eine endliche Gruppe. Es gilt $a, b \in \mathbb{Z}_N : a + b \equiv (a + b) \mod N$. Neutrales element ist 0, Inverses ist $N - a \mod N$
- Für Multiplikation gelten die bekannten Regeln. Bei Abelschen Gruppen gilt $a^m \cdot b^m = (a \cdot b)^m$. Bei Endlichen Gruppen gilt zudem: $a^m = 1$ wenn $|M| = m$ gilt
- $\mathbb{Z}_N(\cdot)$ bildet für alle $N > 1$ eine abelsche Gruppe, wenn $a, b \in \mathbb{Z}_N : a \cdot b \equiv (a \cdot b) \mod N$ gilt

Definition 10.3 (Eulersche Phi-Funktion)

Gibt für eine Zahl n die Mächtigkeit an, wie viele Zahlen $a \in [0, \ldots, n-1]$ kein Teiler von n sind.: $\varphi(n) = \#\{a : 0 \leq a \leq n-1, \; ggT(a, n) = 1\}$

Bemerkung 10.4.

- Kennt man die Primzfaktorzerlegung von N kann dies effizient berechnet werden:

$$\varphi(N) = \prod_i p_i^{e_i - 1}(p_i - 1)$$

Definition 10.5 (Satz von Euler)

Ist $n \in \mathbb{N}$ und $a \in \mathbb{Z}$ mit $ggT(a, n) = 1$, so gilt: $a^{\varphi(n)} \equiv 1 \mod n$

Definition 10.6 (Kleiner Satz von Fermat)

Für eine Primzahl p und eine ganze Zahl a mit $ggT(a, p) = 1$ gilt: $a^p \equiv a \mod p$ bzw. $a^{p-1} \equiv 1 \mod p$

Definition 10.7 (Zyklische Gruppen)

Sei M eine endliche Gruppe, und $a, e \in M$, wobei e das neutrale Element ist. Die Ordnung von a ist die kleinste natürliche Zahl i, sodass $g^i = e$ gilt.

Bemerkung 10.8.

- Hat g die gleiche Ordnung wie M, so nennt man g auch Generator von M
- Definiert man eine zyklische Gruppe über $Z_p(\cdot)$, wobei p eine Primzahl ist, dann ist die Ordnung $p - 1$

11 Grundlagen und Sicherheit von Assymetrischer Verschlüsselung

Definition 11.1 (Assymetrische Kryptosysteme)

Ein assymetrisches Kryptosystem besteht aus drei probabilistischen polynomialzeit Algorithmen:

- $Gen(1^n)$: Der Schlüsselgenerierungsalgorithmus gibt ein Schlüsselpaar s_k, p_k aus, wobei s_k privater Schlüssel und p_k öffentlicher Schlüssel genannt wird
- $Enc_{p_k}(1^n)$: Der Verschlüsselungsalgorithmus bekommt als Eingabe einen Klartext m und verschlüsselt diesen anhand des öffentlichen Schlüssels zu einem Ciffretext. Das wird auch als $c \leftarrow Enc_{p_k}(m)$ geschrieben
- $Dec_{s_k}(1^n)$: Der Entschlüsselungsalgorithmus bekommt als Eingabe einen Ciffretext c und entschlüsselt diesen anhand des privaten Schlüssel. Das wird auch als $m \leftarrow Dec_{s_k}(c)$ geschrieben

Bemerkung 11.2.

- Für die Korrektheit gilt: $Dec_{s_k}(Enc_{p_k}(m)) = m$

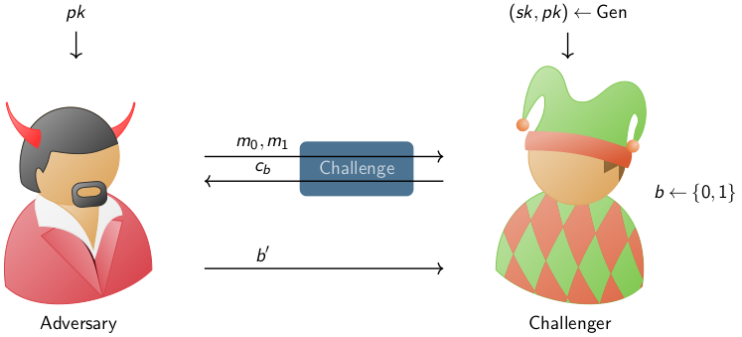
Definition 11.3 (CPA-Sicherheit bei assymetrischen Kryptosystemen)

Ein assymetrisches Kryptosystem Π hat Ununterscheidbare Verschlüsselung gegenüber eines Chosen-Plaintext Angriffs (bzw. ist CPA-Sicher), wenn es für alle probabilistischen polynomialzeit Algorithmen $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ gibt, sodass gilt:

$$P[PubK^{cpa}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Experiment $PubK^{cpa}_{\mathcal{A},\Pi}(n)$

$PubK^{cpa}_{\mathcal{A},\Pi}(n)$
$s_k, p_k \leftarrow Gen(1^n)$
$(m_0, m_1) \leftarrow \mathcal{A}(p_k), \lvert m_0\rvert = \lvert m_1\rvert$
$b \leftarrow \{0,1\}$
$c_b \leftarrow Enc_{p_k}(m_b)$
$b' \leftarrow \mathcal{A}(c_b)$
if $b' = b$ return 1
else return 0



Es wird $PubK^{cpa}_{\mathcal{A},\Pi}(n) = 1$ geschrieben, falls $\mathcal{A}$ erfolgreich war.

Bemerkung 11.4.

- Dadurch das $\mathcal{A}$ nun den öffentlichen Schlüssel hat, hat $\mathcal{A}$ auch ein Verschlüsselungsorakel (da er damit die Verschlüsselung simulieren kann)
- Unterschied zu private Key: Ein Angreifer hat immer das Verschlüsselungsorakel, weshalb die Schwächeren Definitionen nicht anwendbar sind
- Es gilt auch weiterhin, dass deterministische Verfahren nicht CPA-sicher sind UND das CPA-Sicherheit für einzelne Verschlüsselungen CPA-Sicherheit für mehrere Verschlüsselungen impliziert

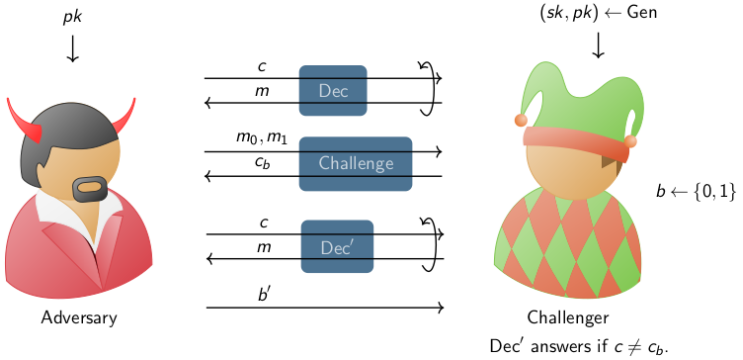
Definition 11.5 (CCA-Sicherheit bei assymetrischen Kryptosystemen)

Ein assymetrisches Kryptosystem Π hat Ununterscheidbare Verschlüsselung gegenüber eines Chosen-Ciphertext Angriffs (bzw. ist CCA-Sicher), wenn es für alle probabilistischen polynomialzeit Algorithmen $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ gibt, sodass gilt:

$$P[PubK^{cca}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Experiment $PubK^{cca}_{\mathcal{A},\Pi}(n)$

$PubK^{cca}_{\mathcal{A},\Pi}(n)$
$s_k, p_k \leftarrow Gen(1^n)$
$(m_0, m_1) \leftarrow \mathcal{A}^{Dec_{s_k}(\cdot)}(p_k), \lvert m_0\rvert = \lvert m_1\rvert$
$b \leftarrow \{0,1\}$
$c_b \leftarrow Enc_{p_k}(m_b)$
$b' \leftarrow \mathcal{A}^{Dec'_{s_k}(\cdot)}(c_b), c \neq c_b$
if $b' = b$ return 1
else return 0



Es wird $PubK^{cca}_{\mathcal{A},\Pi}(n) = 1$ geschrieben, falls $\mathcal{A}$ erfolgreich war.

Bemerkung 11.6.

- Ein CCA-sicheres assymetrisches Verfahren kann mittels KEM realisiert werden

12 Diffie-Hellman Key Exchange

Definition 12.1 (Sicherheit gegenüber einem Abhörer für Schlüsselaustauschprotokolle)

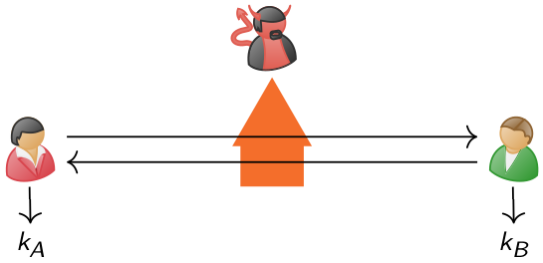
Ein Schlüsselaustausch Protokoll Π is sicher gegenüber Abhörern (bzw. EAV-Sicher), wenn es für jeden probabilistischen polynomialzeit Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ gibt, sodass gilt:

$$P[KE^{eav}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

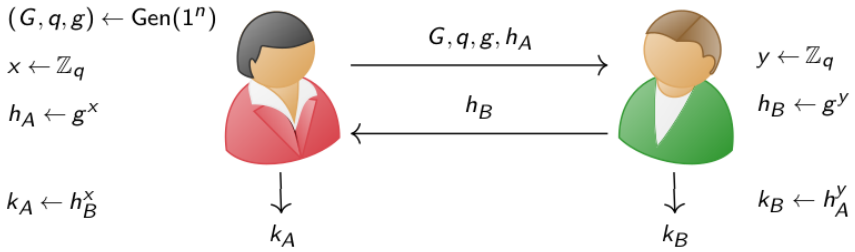
Experiment $KE^{eav}_{\mathcal{A},\Pi}(n)$

Π ist sicher, wenn k dem Angreifer komplett unbekannt ist:

$KE^{eav}_{\mathcal{A},\Pi}(n)$
$(trans, k) \leftarrow < A(1^n), B(1^n) >$
$b \leftarrow \{0,1\}$
if $b = 0$: $k' = k$
if $b = 1$: $k' \leftarrow \{0,1\}^n$
$b' \leftarrow \mathcal{A}(trans, k')$
if $b' = b$ return 1
else return 0



In *trans* werden alle Informationen festgehalten, welche A und B beim Schlüsselaustausch preisgeben. $\mathcal{A}$ bekommt entweder den ausgetauschen Schlüssel k oder einen echt zufälligen String k' und muss rausfinden, was von beidem er bekommen hat. Es wird $KE^{eav}_{\mathcal{A},\Pi}(n) = 1$ geschrieben falls $\mathcal{A}$ erfolgreich war.



Definition 12.2 (Diffie-Hellman Schlüsselaustausch Protokoll)

Standard Eingabe ist weiterhin der Sicherheitsparameter 1^n. Das Protokol zwischen den beiden Akteuren Alice und Bob ist folgendermaßen definiert:

1. Alice berechnet $G(1^n)$ und erhält $(\mathbb{G}, q, g)$
2. Alice wählt echt zufällig ihren privaten Schlüssel $x \in \mathbb{Z}_q$ und berechnet ihren öffentlichen Schlüssel $h_A = g^x$
3. Alice sender $(\mathbb{G}, q, g, h_A)$ an Bob
4. Bob wählt echt zufällig seinen privaten Schlüssel $y \in \mathbb{Z}_q$ und berechnet seinen öffentlichen Schlüssel $h_B = g^y$. Bob sendet h_B an Alice und erzeugt mit $k_B = h_A^y$ den gemeinsamen Schlüssel
5. Alice erhält h_B und erzeugt den gemeinsamen Schlüssel $k_A = h_B^x$

Bemerkung 12.3.

- Das Protokoll ist korrekt: $k_A = h_B^x = (g^y)^x = g^{xy} = (g^x)^y = h_A^y = k_B$
- g ist eine Primitivwurzel (generator)
- Das Protokoll ist nicht sicher vor Man-In-The-Middle Angriffen

Satz 12.4 (Sicherheit des Diffie-Hellman Schlüsselaustausch Protokolls)

Wenn das Decisional Diffie-Hellman Problem schwierig im Bezug auf *Gen* ist, dann ist das Diffie-Hellman Schlüsselaustausch Protokoll sicher gegenüber einem Abhörer.

Bemerkung 12.5.

- Statt bei $b = 1$ für k' einen echt zufälligen n-Bit String zu wählen, wird für diesen Satz k' von *Gen* erzeugt
- **Beweis mittels Reduktion:** Nehme man einen effizienten $\mathcal{A}$ gegen $KE^{eav}_{\mathcal{A},\Pi}(n)$ an, und ein $\mathcal{R}$ der das folgende Decisional DH Problem lösen soll. $\mathcal{R}$ gibt $\mathcal{A}$ direkt seine Eingabe weiter und gibt dessen Output aus. Damit ist $\mathcal{R}$ effizient

12.1 Decisional Diffie-Hellman Annahme

Definition 12.6 (Das Decisional Diffie-Hellman Problem)

Das Decisional Diffie-Hellman Problem DDH ist schwierig im Bezug auf Gen, wenn es für alle probabilistischen polynomialzeit Algorithmen $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ gibt, sodass gilt:

$$|P[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1] - P[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1]| \leq negl(n)$$

Dabei wird die Wahrscheinlichkeit über das Zufallsprinzip von Gen und der echt zufälligen Wahl von $x, y, z \in \mathbb{Z}_q$ berechnet.

Bemerkung 12.7.

- **Berechenbares Diffie Hellman Problem:** Das DDH baut auf dem berechenbaren Diffie-Hellman Problem auf: Kennt $\mathcal{A}$ (G, q, g, g^a, g^b) mit $a, b \in \mathbb{Z}_q$ so ist es schwierig ein c zu finden, sodass $c = g^{ab}$ gilt
- **Frage:** Wenn ein Angreifer A mit Wslk ϵ $CDH_{\mathcal{A},Gen}(n)$ lösen kann, kann man einen Angreifer B konstruieren, der das Decisional Diffie-Hellman Problem mit Wslk $\epsilon - 1/q$ unterscheiden kann?
- **Antwort:** Ja, indem B das $CDH_{\mathcal{A},Gen}(n)$ simuliert und A statt g^a, g^b einfach g^x, g^y mitgibt und von A c bekommt. Es entstehen zwei Fälle: 1. Es gilt $c = g^{xy}$ und B kann mit Wslk ϵ 1 aus. 2. Es gilt mit Wslk $1/q$ $c = g^z$ und B gibt dementsprechend 1 aus. Zusammengefasst kann B also mit Wslk $|1/q - \epsilon|$ unterscheiden
- **Diskreter Logarithmus Problem:** Das berechenbare Diffie Hellman Problem baut auf dem diskreten Logarithmus auf: Kennt $\mathcal{A}$ (G, q, g, h) so ist es schwierig ein $x \in \mathbb{Z}_q$ zu finden, sodass $g^x = h$ gilt
- Der diskrete Logarithmus ist in zyklischen Gruppen mit primer Ordnung schwierig zu lösen
- Hat die Ordnung viele kleine Primfaktoren, so ist das Problem einfach zu lösen
- **Frage:** Wenn ein Angreifer A mit Wslk ϵ $DLog_{\mathcal{A},Gen}(n)$ gewinnt, kann ein B $CDH_{\mathcal{A},Gen}(n)$ mit Wslk ϵ gewinnen?
- **Antwort:** Ja, indem B das $DLog_{\mathcal{A},Gen}(n)$ simuliert und A statt h einfach g^a mitgibt und x' erhält. B gibt dann $g^b \cdot x'$ aus und gewinnt folglich mit Wslk ϵ

12.2 El Gamal Kryptosystem

Definition 12.8 (El Gamal Kryptosystem)

$Gen(n)$
$(\mathbb{G}, q, g) \leftarrow Pgen(1^n)$
$x \leftarrow \mathbb{Z}_q$
$h = g^x$
$s_k = (\mathbb{G}, q, g, x)$
$p_k = (\mathbb{G}, q, g, h)$
return s_k, p_k

$Enc_{p_k}(m)$
$(m \in \mathbb{G})$
$y \leftarrow \mathbb{Z}_q$
$c_1 = g^y$
$c_2 = h^y \cdot m$
return (c_1, c_2)

$Dec_{s_k}(c)$
$(c_1, c_2) = c$
$m = \frac{c_2}{c_1^x}$
return m

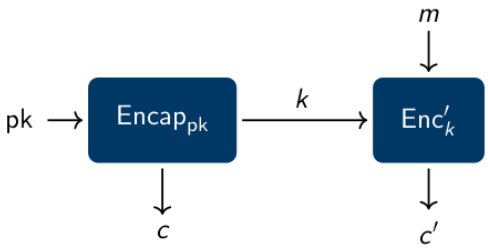
Satz 12.9 (Sicherheit von El Gamal)

Wenn das Decisional Diffie-Hellman Problem schwierig im Bezug auf $\mathbb{G}$ ist, dann ist das El Gamal Kryptosystem CPA-sicher.

Bemerkung 12.10.

- ElGamal ist nicht CCA sicher. Grund: Multiplikativer Homomorphismus: Man nimmt im CCA-Experiment den Ciphertext $(g^y, h^y \cdot m)$ ändert diesen zu $(g^y, h^y \cdot m \cdot m')$ mit beliebigem m' und schickt das dem Entschlüsselungsorakel und erhält m''. Mittels $m = m'' \cdot (m')^{-1}$ erhält man die Ausgangsnachricht

13 Hybride Kryptosysteme



Definition 13.1 (Key Encapsulation Mechanism KEM)

Ein KEM ist ein Tripel aus probabilistischen polynomialzeit Algorithmen:

- $Gen(1^n)$: Der Schlüsselgenerierungsalgorithmus gibt ein paar aus privaten und öffentlichem Schlüssel s_k, p_k aus. Beide haben mind. die Länge n
- $Encaps_{p_k}(1^n)$: Der Verkapselungsalgorithmus nimmt den öffentlichen Schlüssel p_k und gibt den Ciphertext c und einen Schlüssel $k \in \{0,1\}^{l(n)}$ für ein Polynom l aus. Das wird auch $(c,k) \leftarrow Encaps_{p_k}(1^n)$ geschrieben
- $Decaps_{s_k}(c)$: Der deterministische Entkapselungsalgorithmus nimmt den privaten Schlüssel s_k und einen Ciphertext c und gibt einen Schlüssel k (oder bei einem Fehler $\perp$) aus. Das wird auch $k = Decaps_{s_k}(c)$ geschrieben

Bemerkung 13.2.

- In diesem Kontext wird das private Kryptosystem Data-Encapsulation-Mechanism (DEM) genannt

Definition 13.3 (KEM CPA Experiment)

Ein KEM-System ist CPA-sicher, wenn für alle probabilistischen polynomialzeit Algorithmen $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ existiert, sodass gilt:

$$P[KEM^{cpa}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Experiment $KEM^{cpa}_{\mathcal{A},\Pi}(n)$

$KEM^{cpa}_{\mathcal{A},\Pi}(n)$
$(s_k, p_k) \leftarrow Gen(1^n)$
$(c, k_0) \leftarrow Encaps_{p_k}(1^n)$
$k_1 \leftarrow \{0,1\}^n$
$b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}(p_k, c, k_b)$
if $b' = b$ return 1
else return 0

Es wird $KEM^{cpa}_{\mathcal{A},\Pi}(n) = 1$ geschrieben, falls $\mathcal{A}$ erfolgreich war.

Definition 13.4 (KEM CCA Experiment)

Ein KEM-System ist CCA-sicher, wenn für alle probabilistischen polynomialzeit Algorithmen $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ existiert, sodass gilt:

$$P[KEM^{cca}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Experiment $KEM^{cca}_{\mathcal{A},\Pi}(n)$

$KEM^{cca}_{\mathcal{A},\Pi}(n)$
$(s_k, p_k) \leftarrow Gen(1^n)$
$(c, k_0) \leftarrow Encaps_{p_k}(1^n)$
$k_1 \leftarrow \{0,1\}^n$
$b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}^{Decaps'_{s_k}(\cdot)}(p_k, c, k_b), c_i \neq c$
if $b' = b$ return 1
else return 0

Es wird $KEM^{cca}_{\mathcal{A},\Pi}(n) = 1$ geschrieben, falls $\mathcal{A}$ erfolgreich war.

Definition 13.5 (Konstruktion eines Hybriden Kryptosystems)

Sei Π ein KEM-System und sei Π' ein symetrisches Kryptosystem. Ein assymetrisches Kryptosystem Π^{hy} für Nachrichten $m \in \{0,1\}^*$ kann wie folgt definiert werden:

$Gen^{hy}(n)$
$(s_k, p_k) \leftarrow Gen(1^n)$
return (s_k, p_k)

$Enc^{hy}_{p_k}(m)$
$(c, k) \leftarrow Encaps_{p_k}(1^n)$
$c' \leftarrow Enc'_k(m)$
return (c, c')

$Dec^{hy}_{s_k}(c'')$
$(c, c') = c''$
$k = Decaps_{s_k}(c)$
$m = Dec'_k(c')$
return m

Satz 13.6 (CPA-Sicherheit von Π^{hy})

Ist Π ein CPA-sicheres KEM und Π' ein CPA-sicheres symetrisches Kryptosystem, dann ist das hybride Kryptosystem Π^{hy} ein CPA-sicheres assymetrisches Kryptosystem.

Satz 13.7 (CCA-Sicherheit von Π^{hy})

Ist Π ein CCA-sicheres KEM und Π' ein CCA-sicheres symetrisches Kryptosystem, dann ist das hybride Kryptosystem Π^{hy} ein CCA-sicheres assymetrisches Kryptosystem.

14 RSA Kryptosystem

14.1 RSA und das Faktorisierungsproblem

Definition 14.1 (Faktorisierungsproblem)

Das Faktorisierungsproblem ist schwierig bezüglich GenMod, wenn für alle probabilistischen polynomialzeit Algorithmen $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ existiert, sodass gilt:

$$P[Factor_{\mathcal{A},GenMod}(n) = 1] \leq negl(n)$$

Experiment $Factor_{\mathcal{A},GenMod}(n)$

$Factor_{\mathcal{A},GenMod}(n)$
$(N, p, q) \leftarrow GenMod(1^n)$
$(p', q') \leftarrow \mathcal{A}(N)$
if $p' \cdot q' = N$ and $p', q' > 1$ return 1
else return 0

Es wird $Factor_{\mathcal{A},GenMod}(n) = 1$ geschrieben, falls $\mathcal{A}$ erfolgreich war.

Bemerkung 14.2.

- Für RSA wurde nach zusammenhängenden Problemen gesucht: Berechnung der ϕ-Funktion. Kennt man die Faktorisierung nicht, ist die Funktion genauso schwierig zu berechnen wie die Faktorisierung

Definition 14.3 (RSA-Annahme)

Das RSA Problem ist schwierig bezüglich GenRSA, wenn für alle probabilistischen polynomialzeit Algorithmen $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ existiert, sodass gilt:

$$P[RSA - inv_{\mathcal{A},GenRSA}(n) = 1] \leq negl(n)$$

Experiment $RSA - inv_{\mathcal{A},GenRSA}(n)$

$GenRSA(1^n)$
$(N, p, q) \leftarrow GenMod(1^n)$
$\phi(N) = (p-1)(q-1)$
Sample $e > 1$ s.t. $gcd(e, \phi(N)) = 1$
Compute $d = [e^{-1} \mod \phi(N)]$
return (N, e, d)

$RSA - inv_{\mathcal{A},GenRSA}(n)$
$(N, e, d) \leftarrow GenRSA(1^n)$
$y \leftarrow \mathbb{Z}_N^*$
$x \leftarrow \mathcal{A}(N, e, y), x \in \mathbb{Z}_N^*$
if $x^e \equiv y \mod N$ return 1
else return 0

Es wird $RSA - inv_{\mathcal{A},GenRSA}(n) = 1$ geschrieben, falls $\mathcal{A}$ erfolgreich war.

14.2 Definition von RSA

Definition 14.4 (Textbook RSA)

$Gen(1^n)$
$(N, e, d) \leftarrow GenRSA(1^n)$
$p_k = (N, e)$
$s_k = (N, d)$
return (s_k, p_k)

$Enc_{p_k}(m)$
$(N, e) \leftarrow p_k$
$c \equiv m^e \mod N$
return c

$Dec_{s_k}(c)$
$(N, d) = s_k$
$m \equiv c^d \mod N$
return m

Bemerkung 14.5.

- Textbook RSA ist unsicher, wenn ähnliche Nachrichten verschlüsselt werden (da deterministisch)
- **Lösung**: Klartext vorm Verschlüsseln padden und beim Entschlüsseln Padding entfernen

Definition 14.6 (Random Padded RSA)

Padded RSA ist für Nachrichten $m \in \{0,1\}^{\|N\|-l(n)-2}$ definiert, wobei l eine Funktion mit $l(n) \leq 2n - 4 \forall n$ ist:

$Gen(1^n)$
$(N, e, d) \leftarrow GenRSA(1^n)$
$p_k = (N, e)$
$s_k = (N, d)$
return (s_k, p_k)

$Enc_{p_k}(m)$
$(N, e) \leftarrow p_k$
$r \leftarrow \{0,1\}^{l(n)}$
$m' = r \| m \in \mathbb{Z}_N^*$
$c \equiv (m')^e \mod N$
return c

$Dec_{s_k}(c)$
$(N, d) = s_k$
$m' \equiv c^d \mod N$
$m' = r \| m$
return m

Bemerkung 14.7.

- Eine Grundannahme, für die Sicherheit von RSA sicher ist, dass die Nachrichten zufällig gewählt sind. Das versucht Random Padded RSA umzusetzen
- **Sicherheitproblem**: Das Verfahren ist unsicher, wenn $l(n) = O(log(n))$ gilt, aber sicher, wenn $m \in \{0,1\}$, was in den Sicherheitsbeweisen anhand von Hardcore Prädikaten ausgenutzt und gezeigt wird. Die dazwischenliegenden Fälle sind unklar

14.3 RSA Sicherheit

Bemerkung 14.8 (CPA-Sicherheit mit Hardcore Prädikaten).

- Die Sicherheit von RSA kann außerhalb des Random Orakle Modells mit Hardcore Prädikaten für Nachrichten der Größe 1 bewiesen werden
- **RSA Hardcore Prädikat**: Das Least Significant Bit lsb der Zahl die Verschlüsselt wird ist das Hardcore Prädikat. D.h. $(N,e) \leftarrow p_k \Rightarrow y \equiv m^e \mod N$ dann ist $lsb(m)$ das Hardcore Prädikat der RSA-Verschlüsselung
- **Kryptosystem**: Dabei verschlüsselt man $m \in \{0,1\}$, indem man eine Zufallszahl r mit $lsb(r) = m$ verschlüsselt und beim entschlüsseln $lsb(r)$ ausgibt
- **Sicherheit**: Dieses Kryptosystem ist CPA-sicher

Definition 14.9 (RSA-KEM Konstruktion)

Sei $H : \mathbb{Z}_N^* \to \{0,1\}^n$ eine Hash-Funktion.

$Gen(1^n)$
$(N,e,d) \leftarrow GenRSA(1^n)$
$p_k = (N,e)$
$s_k = (N,d)$
return (s_k, p_k)

$Encaps_{p_k}(1^n)$
$(N,e) \leftarrow p_k$
$r \leftarrow \mathbb{Z}_N^*$
$c \equiv r^e \mod N$
$k = H(r)$
return (c,k)

$Decaps_{s_k}(c)$
$(N,d) = s_k$
$r \equiv c^d \mod N$
$k = H(r)$
return k

Bemerkung 14.10.

- Diese Konstruktion benötigt keine Hardcore Prädikate
- Mittels des Hashings wird CCA-Sicherheit gewährleistet

Satz 14.11 (Sicherheit von RSA-KEM)

Wenn das RSA Problem schwierig bezüglich GenRSA ist und H als Zufallsorakel modeliert wird, dann ist die RSA-KEM Konstruktion CCA-sicher.

15 Signaturen

Definition 15.1 (Digitale Signaturen)

Ein digitales Signaturschemata besteht aus drei probabilistischen polynomialzeit Algorithmen:

- $Gen(1^n)$: Der **Schlüsselgenerierungsalgorithmus** gibt ein Paar s_k, p_k aus, wobei s_k der private und p_k der öffentliche Schlüssel ist
- $Sign_{s_k}(m)$: Der **Signieralgorithmus** nimmt den privaten Schlüssel s_k und eine Klartextnachricht $m \in \{0,1\}^*$ und gibt eine Signatur σ aus. Das wird auch als $\sigma \leftarrow Sign_{s_k}(m)$ geschrieben
- $Vrfy_{p_k}(m,\sigma)$: Der deterministische **Verifikationsalgorithmus** nimmt den öffentlichen Schlüssel p_k und eine Signatur σ und gibt 1 aus, wenn die Signatur gültig ist und ansonsten 0. Das wird auch als $b = Vrfy_{p_k}(m,\sigma)$ geschrieben

Bemerkung 15.2.

- **Grundidee**: Ein Akteur kann ein Dokument signieren und jeder andere die Signatur dieses Dokumentes überprüfen. Damit wird Integrität und Zuordnung zu einer Person sichergestellt
- **Vorgehensweise:** Der Sender signiert eine Datei mit seinem privatem Schlüssel. Der Empfänger kann mit dem öffentlichen Schlüssel des Senders die Signatur verifizieren
- Es gilt die Korrektheit: $Vrfy_{p_k}(m, Sign_{s_k}(m)) = 1$

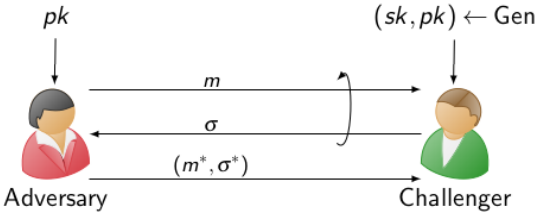
Definition 15.3 (Sicherheit von digitalen Signaturen)

Ein digitales Signaturschemata Π ist fälschungssicher bzw. sicher, wenn für alle probabilistischen polynomialzeit Algorithmen $\mathcal{A}$ eine vernachlässigbare Funktion $negl(n)$ existiert, sodass gilt:

$$P[Sig - forge_{\mathcal{A},\Pi}(n) = 1] \leq negl(n)$$

Experiment $Sig - forge_{\mathcal{A},\Pi}(n)$

$Sig - forge_{\mathcal{A},\Pi}(n)$
$(s_k, p_k) \leftarrow Gen(1^n)$
$(m', \sigma') \leftarrow \mathcal{A}^{Sign(\cdot)}(p_k)$
Sei Q die Menge aller Anfragen
if $Vrfy_{p_k}(m', \sigma') = 1$ and $m' \notin Q$ return 1
else return 0



Es wird $Sig - forge_{\mathcal{A},\Pi}(n) = 1$ geschrieben, falls $\mathcal{A}$ erfolgreich war.

15.1 Signaturen mit Hashing

Definition 15.4 (Hash-and-Sign Paradigma)

Sei $\Pi = (Gen_s, Sign, Vrfy)$ ein Signaturverfahren für Nachrichten der Länge $l(n)$ und sei $\Pi_H = (Gen_H, H)$ eine Hashfunktion, dann lässt sich ein Hash-and-Sign Paradigma $\Pi_{H\&S}$ folgendermaßen definieren:

$Gen(1^n)$
$(s_k, p_k) \leftarrow Gen_S(1^n)$
$s = Gen_H(1^n)$
$p'_k = (p_k, s)$
$s'_k = (s_k, s)$
return (s'_k, p'_k)

$Sign_{s'_k}(m)$
$(s_k, s) = s'_k$
$\sigma = Sign_{s_k}(H^s(m))$
return σ

$Vrfy_{p'_k}(m, \sigma)$
$(p_k, s) = p'_k$
return $Vrfy_{p_k}(H^s(m), \sigma)$

Satz 15.5 (Sicherheit des Hash-and-Sign Paradigma)

Wenn Π sicher und Π_H kollisionsresistent ist, dann ist auch $\Pi_{H\&S}$ sicher.

15.2 RSA Signatur

Definition 15.6 (RSA-FDH Signatur)

RSA full-domain Hash (RSA-FDH) ist für Nachrichten $m \in \{0,1\}^*$ mit $H : \{0,1\}^* \to \mathbb{Z}_N^*$, die Teil des öffentlichen Schlüssels ist, folgendermaßen definiert:

$Gen(1^n)$
$(N, e, d) \leftarrow GenRSA(1^n)$
$s_k = (N, d)$
$p_k = (N, e)$
return (s_k, p_k)

$Sign_{s_k}(m)$
$(N, d) = s_k$
$\sigma \equiv H(m)^d \mod N$
return σ

$Vrfy_{p_k}(m, \sigma)$
$(N, e) = p_k$
if $H(m) \equiv \sigma^e \mod N$ return 1
else return 0

Bemerkung 15.7.

- Bei Textbook RSA Signatur würde einfach H weggelassen werden, d.h. $\sigma \equiv m^d \mod N$ und *if* $m \equiv \sigma^e \mod N$ *return* 1. Hier wurde gleich die sichere Hash-and-Sign Variante gezeigt
- Die Textbook RSA Signatur ist nicht sicher. Beispiel No-Message-Attack: Man wählt ein beliebiges $\sigma \in \mathbb{Z}_N^*$ und erzeugt dazu eine nutzlose, aber gültige Nachricht $m \equiv \sigma^e \mod N$. Somit könnte ein Angreifer mit (m, σ) das $Sig - forge_{\mathcal{A},\Pi}(n)$-Experiment effizient gewinnen

Satz 15.8 (Sicherheit von RSA-FDH)

Wenn das RSA Problem schwierig im Bezug auf GenRSA und ein H, welches als Zufallsorakel modeliert wird, ist, dann ist RSA-FDH sicher.