

# Kryptographie II

## Asymmetrische Kryptographie

Alexander May

Fakultät für Mathematik  
Ruhr-Universität Bochum

Wintersemester 2015/16

# Erinnerung an Kryptographie I

## Symmetrische Kryptographie

- Parteien besitzen gemeinsamen geheimen Schlüssel.
- Erlaubt Verschlüsselung, Authentifikation, Hashen, Auswerten von Pseudozufallspermutationen.
- **Frage:** Wie tauschen die Parteien einen Schlüssel aus?

## Nachteile

- 1  $U$  Teilnehmer benötigen  $\binom{U}{2} = \Theta(U^2)$  viele Schlüssel.
- 2 Jeder Teilnehmer muss  $U - 1$  Schlüssel sicher speichern. Update erforderlich, falls Teilnehmer hinzukommen oder gelöscht werden.
- 3 Schlüsselaustausch funktioniert nicht in offenen Netzen.

# Schlüsselverteilungs-Center (KDC)

**Partielle Lösung:** Verwenden vertrauenswürdige Instanz

- IT-Manager eröffnet Key Distribution Center (KDC).
- Teilnehmer besitzen gemeinsamen, geheimen Schlüssel mit KDC.
- Alice schickt Nachricht "Kommunikation mit Bob" an KDC.
- Alice authentisiert Nachricht mit ihrem geheimen Schlüssel.
- KDC wählt einen *Session-Key*  $k$ , d.h. einen neuen Schlüssel.
- KDC schickt Verschlüsselung  $Enc_{k_A}(k)$  an Alice.
- KDC schickt Verschlüsselung  $Enc_{k_B}(k)$  an Bob.

*Alternativ im Needham Schröder Protokoll:*

KDC schickt  $Enc_{k_B}(k)$  an Alice und diese leitet an Bob weiter.

# Vor- und Nachteile von KDCs

## Vorteile

- Jeder Teilnehmer muss nur *einen* Schlüssel speichern.
- Hinzufügen/Entfernen eines Teilnehmers erfordert Update *eines* Schlüssels.

## Nachteile

- Kompromittierung von KDC gefährdet das gesamte System.
- Falls KDC ausfällt, ist sichere Kommunikation nicht möglich.

## Praktischer Einsatz von KDCs

- Kerberos (ab Windows 2000)

# Diffie Hellman Gedankenexperiment

## Szenario

- Alice will eine Kiste zu Bob schicken.
- Post ist nicht zu trauen, d.h. die Kiste muss verschlossen werden.
- Sowohl Alice als auch Bob besitzen ein Schloss.

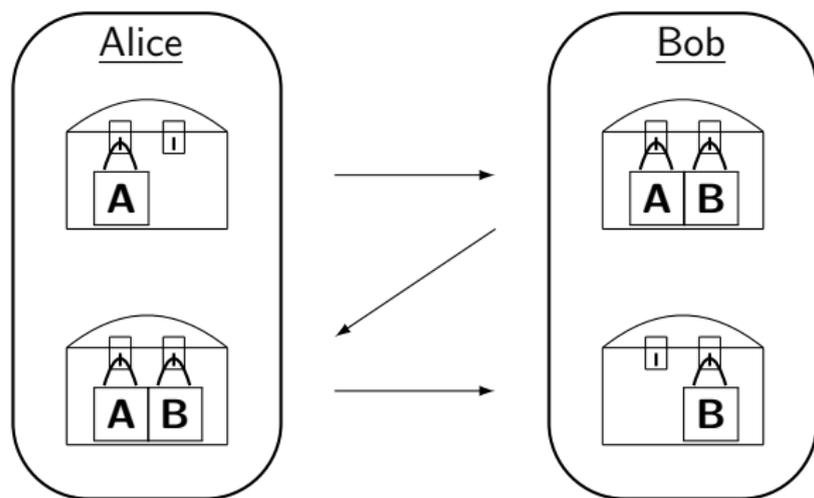
## Algorithmus 3-Runden Diffie-Hellman Austausch

- 1 Alice sendet die Kiste an Bob, verschlossen mit ihrem Schlüssel.
- 2 Bob sendet die Kiste zurück, verschlossen mit seinem Schlüssel.
- 3 Alice entfernt ihr Schloss und sendet die Kiste an Bob.
- 4 Bob entfernt sein Schloss und öffnet die Kiste.

**Beobachtung:** Viele Funktionen sind inherent asymmetrisch.

- Zudrücken eines Schlosses ist leicht, Öffnen ist schwer.
- Multiplizieren von Zahlen ist leicht, Faktorisieren ist schwer.
- Exponentieren von Zahlen ist leicht,  $\log$  ist (oft) schwer.

# Diffie Hellman Gedankenexperiment



# Diffie-Hellman Schlüsselaustausch (1976)

## Szenario:

- Alice und Bob verwenden öffentlichen Kanal.
- **Ziel:** Beide wollen einen zufälligen Bitstring  $k$  austauschen.
- **Angreifer ist passiv**, d.h. kann nur lauschen, nicht manipulieren.

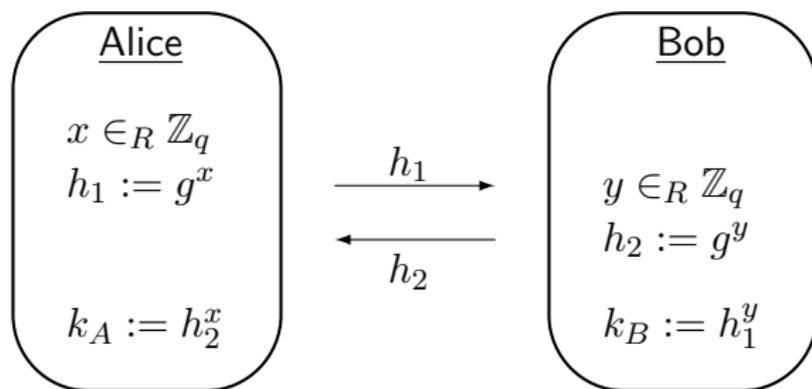
## Systemparameter:

- Sicherheitsparameter  $1^n$
- Schlüsselerzeugung  $(G, q, g) \leftarrow \mathcal{G}(1^n)$ 
  - ▶  $\mathcal{G}$  ist probabilistischer polynomial-Zeit (in  $n$ ) Algorithmus
  - ▶  $G$  ist multiplikative Gruppe mit Ordnung  $q$  und Generator  $g$ .

# 2-Runden Diffie-Hellman Schlüsselaustausch

## Protokoll 2-Runden Diffie-Hellman Schlüsselaustausch

- 1 Alice: Wähle  $x \in_R \mathbb{Z}_q$ . Sende  $h_1 = g^x$  an Bob.
- 2 Bob: Wähle  $y \in_R \mathbb{Z}_q$ . Sende  $h_2 = g^y$  an Alice.
- 3 Alice: Berechne  $k_A = h_2^x$ .
- 4 Bob: Berechne  $k_B = h_1^y$ .



# Korrektheit und Schlüsselerzeugung

**Korrektheit:**  $k_A = k_B$

- Alice berechnet Schlüssel  $k_A = h_2^x = (g^y)^x = g^{xy}$ .
- Bob berechnet Schlüssel  $k_B = h_1^y = (g^x)^y = g^{xy}$ .

**Schlüsselerzeugung:**

- Gemeinsamer Schlüssel  $k_A = k_B \in G$  ist ein Gruppenelement, kein Zufallsstring  $k \in \{0, 1\}^m$ .
- Konstruktion von Zufallsstring mittels sog. *Zufallsextraktoren*.
- Sei  $k_A$  ein zufälliges Gruppenelement aus  $G$ .
- Zufallsextraktor liefert bei Eingabe  $k_A$  einen Schlüssel  $k \in \{0, 1\}^m$ , ununterscheidbar von einem Zufallsstring derselben Länge.

**Übung:** Schlüssel  $k$  + sichere symmetrische Verschlüsselung liefert zusammen ein beweisbar sicheres Verfahren.

# Spiel zur Unterscheidung des Schlüssels

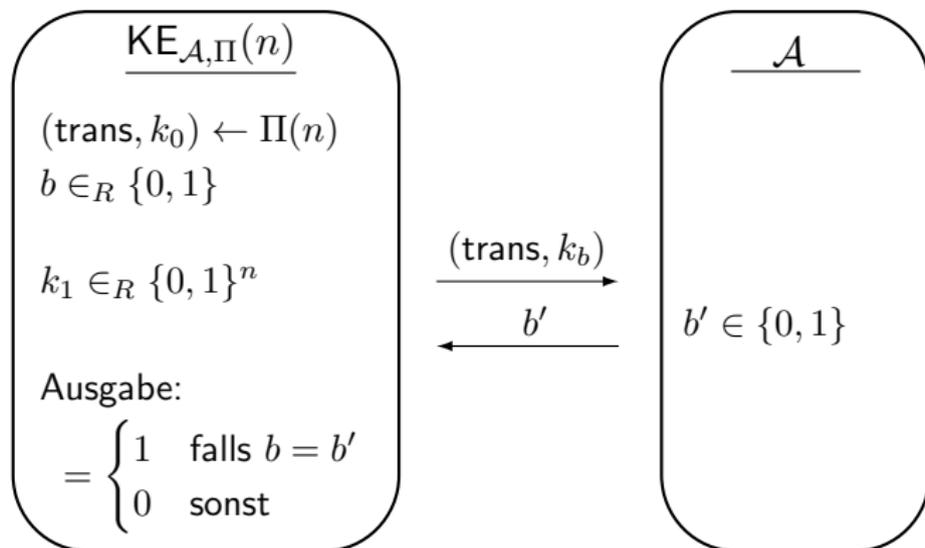
## Spiel Schlüsselaustausch $KE_{\mathcal{A},\Pi}(n)$

Sei  $\Pi$  ein Schlüsselaustausch Protokoll für Gruppenelemente aus  $G$ .  
Sei  $\mathcal{A}$  ein Angreifer für  $\Pi$ .

- 1  $(k_0, \text{trans}) \leftarrow \Pi(n)$ , wobei  $k_0$  der gemeinsame Schlüssel und  $\text{trans}$  der Protokollablauf ist.
- 2 Wähle  $k_1 \in_R \{0, 1\}^n$  und  $b \in_R \{0, 1\}$ .
- 3  $b' \leftarrow \mathcal{A}(\text{trans}, k_b)$ . Ausgabe  $\begin{cases} 1 & \text{falls } b' = b \\ 0 & \text{sonst} \end{cases}$ .

- $\mathcal{A}$  gewinnt, falls  $KE_{\mathcal{A},\Pi}(n) = 1$ .
- D.h.  $\mathcal{A}$  gewinnt, falls er erkennt, welches der korrekte Schlüssel  $k$  des Protokolls  $\Pi$  und welches der zufällige Schlüssel  $k' \in_R G$  ist.
- $\mathcal{A}$  kann trivialerweise mit Ws  $\frac{1}{2}$  gewinnen. (Wie?)

# Spiel zur Unterscheidung des Schlüssels



# Sicherheit Schlüsselaustausch

## Definition $\text{negl}(n)$

*Erinnerung aus Krypto I*

Eine Funktion  $f : \mathbb{N} \rightarrow \mathbb{R}^+$  heißt *vernachlässigbar*, falls für jedes Polynom  $p(n)$  und alle hinreichend großen  $n$  gilt  $f(n) < \frac{1}{p(n)}$ .

**Notation:** Wir bezeichnen eine bel. vernachlässigbare Fkt mit  $\text{negl}(n)$ .

## Bsp:

- $\frac{1}{2^n}$ ,  $\frac{1}{2^{\sqrt{n}}}$ ,  $\frac{1}{n^{\log \log n}}$  sind vernachlässigbar.
- $\frac{1}{2^{\mathcal{O}(\log n)}}$  ist nicht vernachlässigbar.
- Es gilt  $q(n) \cdot \text{negl}(n) = \text{negl}(n)$  für jedes Polynom  $q(n)$ .

## Definition Sicherheit Schlüsselaustausch

Ein Schlüsselaustausch Protokoll  $\Pi$  ist sicher gegen passive Angriffe, falls für alle probabilistischen Polynomialzeit (ppt) Angreifer  $\mathcal{A}$  gilt  $\text{Ws}[KE_{\mathcal{A}, \Pi}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$ .

Der Wsraum ist definiert über die zufälligen Münzwürfe von  $\mathcal{A}$  und  $\Pi$ .

# dlog Problem

## Definition Diskrete Logarithmus (dlog) Annahme

Das *Diskrete Logarithmus Problem* ist hart bezüglich  $\mathcal{G}$ , falls für alle ppt Algorithmen  $\mathcal{A}$  gilt

$$|\text{Ws}[\mathcal{A}(g, q, g^x) = x]| \leq \text{negl.}$$

Der Wsraum ist definiert bezüglich der zufälligen Wahl von  $x \in \mathbb{Z}_q$  und der internen Münzwürfe von  $\mathcal{A}$  und  $\mathcal{G}$ .

*dlog Annahme*: Das dlog Problem ist hart bezüglich  $\mathcal{G}$ .

- Unter der dlog Annahme können die geheimen Schlüssel  $x, y$  bei Diffie-Hellman nur mit vernachlässigbarer Ws berechnet werden.
- D.h. die dlog Annahme ist eine notwendige Sicherheitsannahme.
- Problem: Die Berechnung von  $g^{xy}$  aus  $g^x, g^y$  könnte einfach sein.

# CDH Problem

## Definition Computational Diffie-Hellman (CDH) Annahme

Das *Computational Diffie-Hellman Problem* ist hart bezüglich  $\mathcal{G}$ , falls für alle ppt Algorithmen  $\mathcal{A}$  gilt  $\text{Ws}[\mathcal{A}(g, q, g^x, g^y) = g^{xy}] \leq \text{negl}$ .

Wsraum: zufällige Wahl von  $x, y \in_R \mathbb{Z}_q$ , interne Münzwürfe von  $\mathcal{A}$ ,  $\mathcal{G}$ .

*CDH Annahme*: Das CDH Problem ist hart bezüglich  $\mathcal{G}$ .

- Unter der CDH-Annahme kann ein DH-Angreifer Eve den Schlüssel  $k_A = g^{xy}$  nur mit vernachlässigbarer Ws berechnen.

### Problem:

- Sei CDH schwer, so dass Angreifer Eve  $k_A$  nicht berechnen kann.
- Benötigen aber, dass  $k_A$  ein zufälliges Gruppenelement in  $G$  ist.
- Unterscheiden von  $g^{xy}$  und  $g^z$ ,  $z \in_R \mathbb{Z}_q$  könnte einfach sein.

# DDH Problem

## Definition Decisional Diffie-Hellman (DDH) Annahme

Das *Decisional Diffie-Hellman Problem* ist hart bezüglich  $\mathcal{G}$ , falls für alle ppt Algorithmen  $\mathcal{A}$  gilt

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

Wsraum: zufällige Wahl von  $x, y, z \in_R \mathbb{Z}_q$ , interne Münzwürfe von  $\mathcal{A}$ ,  $\mathcal{G}$ .

*DDH Annahme*: Das DDH Problem ist hart bezüglich  $\mathcal{G}$ .

- Unter der DDH-Annahme kann Eve den DH-Schlüssel  $g^{xy}$  nicht von einem zufälligen Gruppenelement unterscheiden.

# Sicherheitsbeweis des DH-Protokolls

## Satz Sicherheit des Diffie-Hellman Protokolls

Unter der DDH-Annahme ist das DH-Protokoll  $\Pi$  sicher gegen passive Angreifer  $\mathcal{A}$ .

**Beweis:** Es gilt  $\text{Ws}[KE_{\mathcal{A},\Pi}(n) = 1]$

$$\begin{aligned} &= \frac{1}{2} \cdot \text{Ws}[KE_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] + \frac{1}{2} \cdot \text{Ws}[KE_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] \\ &= \frac{1}{2} \cdot \text{Ws}[\mathcal{A}(G, g, q, g^x, g^y, g^{xy}) = 0] + \frac{1}{2} \cdot \text{Ws}[\mathcal{A}(G, g, q, g^x, g^y, g^z) = 1] \\ &= \frac{1}{2} \cdot (1 - \text{Ws}[\mathcal{A}(G, g, q, g^x, g^y, g^{xy}) = 1]) + \frac{1}{2} \text{Ws}[\mathcal{A}(G, g, q, g^x, g^y, g^z) = 1] \\ &= \frac{1}{2} + \frac{1}{2} \cdot (\text{Ws}[\mathcal{A}(G, g, q, g^x, g^y, g^z) = 1] - \text{Ws}[\mathcal{A}(G, g, q, g^x, g^y, g^{xy}) = 1]) \\ &\leq \frac{1}{2} + \frac{1}{2} \cdot |\text{Ws}[\mathcal{A}(G, g, q, g^x, g^y, g^{xy}) = 1] - \text{Ws}[\mathcal{A}(G, g, q, g^x, g^y, g^z) = 1]| \\ &\leq \frac{1}{2} + \frac{1}{2} \cdot \text{negl}(n) = \frac{1}{2} + \text{negl}(n) \quad \text{nach DDH-Annahme.} \end{aligned}$$

# Public-Key Verschlüsselung

## Szenario: Asymmetrische/Public Key Verschlüsselung

- Schlüsselpaar  $(pk, sk)$  aus öffentlichem/geheimem Schlüssel.
- Verschlüsselung  $Enc_{pk}$  ist Funktion des öffentlichen Schlüssels.
- Entschlüsselung  $Dec_{sk}$  ist Funktion des geheimen Schlüssels.
- $pk$  kann veröffentlicht werden, z.B. auf Webseite, Visitenkarte.
- $pk$  kann über öffentlichen (authentisierten) Kanal verschickt werden.

## Vorteile:

- Löst Schlüsselverteilungsproblem.
- Erfordert die sichere Speicherung eines einzigen Schlüssels.

## Nachteil:

- Heutzutage deutlich langsamer als sym. Verschlüsselung.

# Public-Key Verschlüsselung

## Definition Public-Key Verschlüsselung

Ein *Public-Key Verschlüsselungsverfahren* ist ein 3-Tupel

$\Pi = (Gen, Enc, Dec)$  von ppt Algorithmen mit

① **Gen:**  $(pk, sk) \leftarrow Gen(1^n)$ , wobei  $pk$  der öffentliche und  $sk$  der geheime Schlüssel ist. Beide Schlüssel besitzen Länge mind.  $n$ .

② **Enc:** Für eine Nachricht  $m \in \mathcal{M}$  und Schlüssel  $pk$  berechne  
$$c \leftarrow Enc_{pk}(m).$$

③ **Dec:** Für einen Chiffretext  $c \in \mathcal{C}$  und Schlüssel  $sk$  berechne  
$$m' := Dec_{sk}(c).$$

Es gilt  $Ws[Dec_{sk}(Enc_{pk}(m)) = m] = 1 - \text{negl}(n)$ .

# Ununterscheidbarkeit von Chiffretexten

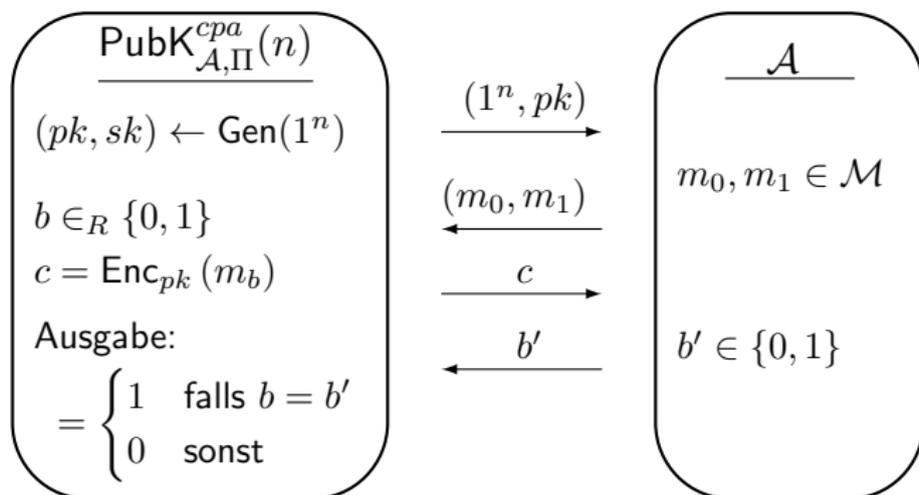
## Spiel CPA Ununterscheidbarkeit von Chiffretexten $PubK_{\mathcal{A}, \Pi}^{cpa}(n)$

Sei  $\Pi$  ein PK-Verschlüsselungsverfahren und  $\mathcal{A}$  ein Angreifer.

- 1  $(pk, sk) \leftarrow Gen(1^n)$
- 2  $(m_0, m_1) \leftarrow \mathcal{A}(pk)$
- 3 Wähle  $b \in_R \{0, 1\}$ .  $b' \leftarrow \mathcal{A}(Enc_{pk}(m_b))$ .
- 4  $PubK_{\mathcal{A}, \Pi}^{cpa}(n) = \begin{cases} 1 & \text{für } b = b' \\ 0 & \text{sonst} \end{cases}$

- Man beachte, dass  $\mathcal{A}$  Orakelzugriff auf  $Enc_{pk}$  besitzt.
- D.h.  $\mathcal{A}$  kann sich beliebig gewählte Klartexte verschlüsseln lassen.  
(chosen plaintext attack = CPA)

# CPA-Spiel



## Definition CPA Sicherheit von Verschlüsselung

Ein PK-Verschlüsselungsverfahren  $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$  heißt *CPA-sicher*, d.h.  $\Pi$  besitzt ununterscheidbare Verschlüsselungen unter CPA, falls für alle ppt  $\mathcal{A}$  gilt

$$\text{Ws}[PubK_{\mathcal{A}, \Pi}^{cpa}(n) = 1] \leq \frac{1}{2} + \text{negl}(n).$$

- **Übung:** *Unbeschränkte*  $\mathcal{A}$  können das Spiel  $PubK_{\mathcal{A}, \Pi}^{cpa}(n)$  mit Ws  $1 - \text{negl}(n)$  gewinnen.

# Unsicherheit deterministischer Verschlüsselung

## Satz Deterministische Verschlüsselung

Deterministische PK-Verschlüsselung ist unsicher gegenüber CPA.

### Beweis:

- $\mathcal{A}$  kann sich  $Enc_{pk}(m_0)$  und  $Enc_{pk}(m_1)$  selbst berechnen.
- D.h. ein Angreifer  $\mathcal{A}$  gewinnt  $PubK_{\mathcal{A}, \Pi}^{cpa}(n)$  mit Ws 1.

# Mehrfache Verschlüsselung

## Spiel Mehrfache Verschlüsselung $PubK_{\mathcal{A},\Pi}^{mult}(n)$

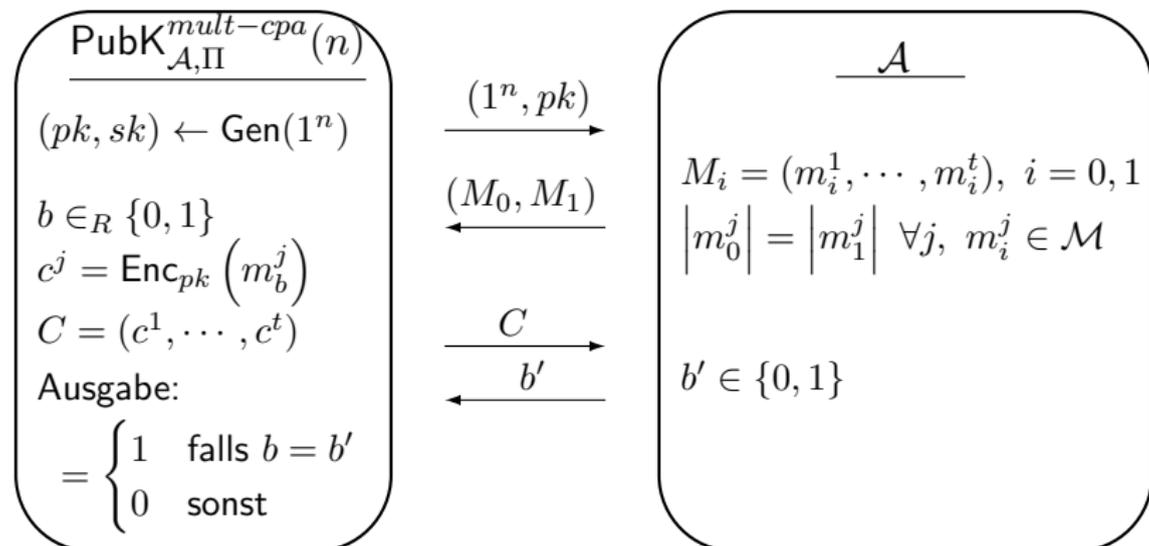
Sei  $\Pi$  ein PK-Verschlüsselungsverfahren und  $\mathcal{A}$  ein Angreifer.

- 1  $(pk, sk) \leftarrow Gen(1^n)$
- 2  $(M_0, M_1) \leftarrow \mathcal{A}(pk)$ , wobei  $M_i = (m_i^1, \dots, m_i^t)$ ,  $i = 0, 1$  und  $|m_0^j| = |m_1^j|$  für  $j \in [t]$ .
- 3 Wähle  $b \in_R \{0, 1\}$ .  $b' \leftarrow \mathcal{A}(Enc_{pk}(m_b^1), \dots, Enc_{pk}(m_b^t))$ .
- 4  $PubK_{\mathcal{A},\Pi}^{mult}(n) = \begin{cases} 1 & \text{für } b = b' \\ 0 & \text{sonst} \end{cases}$ .

## Definition CPA Sicherheit von mehrfacher Verschlüsselung

Ein PK-Verschlüsselungsverfahren  $\Pi = (Gen, Enc, Dec)$  heißt *mult-CPA sicher*, d.h. besitzt ununterscheidbare mehrfache Verschlüsselungen unter CPA, falls für alle ppt  $\mathcal{A}$  gilt  $Ws[PubK_{\mathcal{A},\Pi}^{mult}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$ .

# multCPA-Spiel



# Sicherheit mehrfacher Verschlüsselung

## Satz Sicherheit mehrfacher Verschlüsselung (analog zu Krypto I)

Sei  $\Pi$  ein PK-Verschlüsselungsschema.  $\Pi$  ist mult-CPA sicher gdw  $\Pi$  CPA sicher ist.

**Beweis** “ $\Leftarrow$ ”: Für  $t = 2$ .

- Ein Angreifer  $\mathcal{A}$  gewinnt das Spiel  $PubK_{\mathcal{A},\Pi}^{mult}(n)$  mit  $W_s$

$$\frac{1}{2} W_s[\mathcal{A}(Enc_{pk}(m_0), Enc_{pk}(m_0)) = 0] + \frac{1}{2} W_s[\mathcal{A}(Enc_{pk}(m_1), Enc_{pk}(m_1)) = 1].$$

- Daraus folgt  $W_s[PubK_{\mathcal{A},\Pi}^{mult}(n)] + \frac{1}{2} =$

$$\begin{aligned} & \frac{1}{2} W_s[\mathcal{A}(Enc_{pk}(m_0), Enc_{pk}(m_0)) = 0] + \frac{1}{2} W_s[\mathcal{A}(Enc_{pk}(m_1), Enc_{pk}(m_1)) = 1] \\ & + \frac{1}{2} (W_s[\mathcal{A}(Enc_{pk}(m_0), Enc_{pk}(m_1)) = 0] + W_s[\mathcal{A}(Enc_{pk}(m_0), Enc_{pk}(m_1)) = 1]) \end{aligned}$$

- **Ziel:** Zeigen, dass  $W_s[PubK_{\mathcal{A},\Pi}^{mult}(n)] + \frac{1}{2} \leq 1 + \text{negl}(n)$ .

# Betrachten der Hybride

## Lemma

$$\frac{1}{2} \text{Ws}[\mathcal{A}(\text{Enc}_{pk}(m_0^1), \text{Enc}_{pk}(m_0^2)) = 0] + \frac{1}{2} \text{Ws}[\mathcal{A}(\text{Enc}_{pk}(m_0^1), \text{Enc}_{pk}(m_1^2)) = 1] \leq \frac{1}{2} + \text{negl}(n).$$

**Beweis:** Sei  $\mathcal{A}'$  Angreifer für *einfache* Verschlüsselungen.

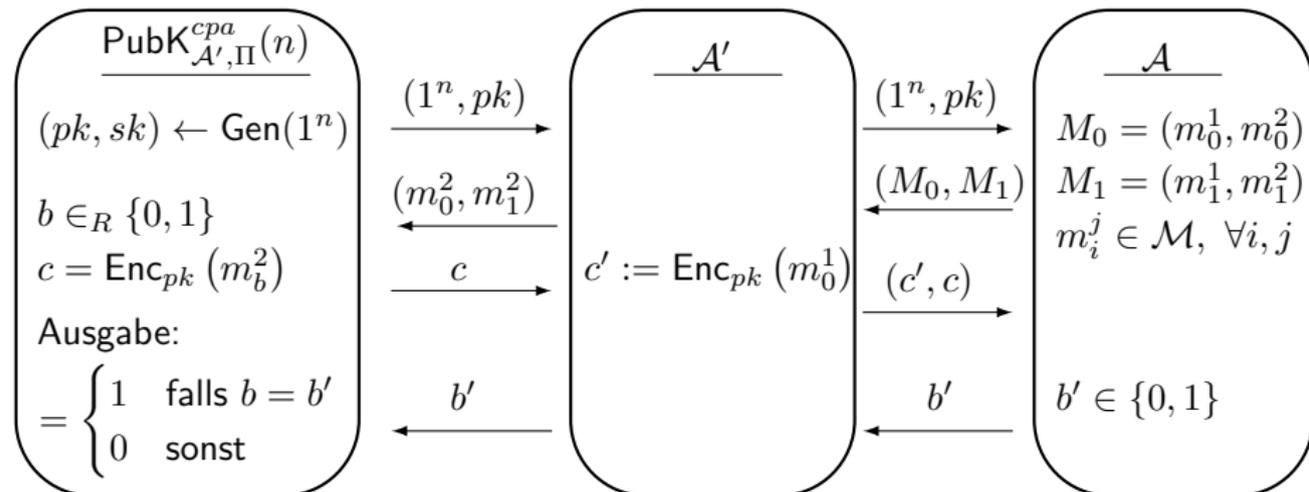
- $\mathcal{A}'$  versucht mittels  $\mathcal{A}$  das Spiel  $\text{PubK}_{\mathcal{A}', \Pi}^{\text{cpa}}(n)$  zu gewinnen.

## Strategie von Angreifer $\mathcal{A}'$

- 1  $\mathcal{A}'$  gibt  $pk$  an  $\mathcal{A}$  weiter.
- 2  $(M_0, M_1) \leftarrow \mathcal{A}(pk)$  mit  $M_0 = (m_0^1, m_0^2)$  und  $M_1 = (m_1^1, m_1^2)$ .
- 3  $\mathcal{A}'$  gibt  $(m_0^2, m_1^2)$  aus.  $\mathcal{A}'$  erhält Chiffretext  $c(b) = \text{Enc}_{pk}(m_b^2)$ .
- 4  $b' \leftarrow \mathcal{A}(\text{Enc}_{pk}(m_0^1), c(b))$ .
- 5  $\mathcal{A}'$  gibt Bit  $b'$  aus.

- $\text{Ws}[\mathcal{A}'(\text{Enc}_{pk}(m_0^2)) = 0] = \text{Ws}[\mathcal{A}(\text{Enc}_{pk}(m_0^1), \text{Enc}_{pk}(m_0^2)) = 0]$  und
- $\text{Ws}[\mathcal{A}'(\text{Enc}_{pk}(m_1^2)) = 1] = \text{Ws}[\mathcal{A}(\text{Enc}_{pk}(m_0^1), \text{Enc}_{pk}(m_1^2)) = 1]$ .

# Strategie des Angreifers bei Hybriden



# Fortsetzung Hybridtechnik

## Beweis(Fortsetzung):

- CPA Sicherheit von  $\Pi$  bei einzelnen Nachrichten impliziert

$$\begin{aligned}\frac{1}{2} + \text{negl}(n) &\geq \text{Ws}[PubK_{\mathcal{A}', \Pi}^{cpa}(n) = 1] \\ &= \frac{1}{2} \text{Ws}[\mathcal{A}'(Enc_{pk}(m_0^2)) = 0] + \frac{1}{2} \text{Ws}[\mathcal{A}'(Enc_{pk}(m_1^2)) = 1] \\ &= \frac{1}{2} \text{Ws}[\mathcal{A}((Enc_{pk}(m_0^1), Enc_{pk}(m_0^2)) = 0)] + \\ &\quad \frac{1}{2} \text{Ws}[\mathcal{A}((Enc_{pk}(m_0^1), Enc_{pk}(m_1^2)) = 1)] \quad \square_{\text{Lemma}}\end{aligned}$$

- Analog kann gezeigt werden, dass

$$\begin{aligned}\frac{1}{2} + \text{negl}(n) &\geq \frac{1}{2} \text{Ws}[\mathcal{A}((Enc_{pk}(m_0^1), Enc_{pk}(m_1^2)) = 0)] + \\ &\quad \frac{1}{2} \text{Ws}[\mathcal{A}((Enc_{pk}(m_1^1), Enc_{pk}(m_1^2)) = 1)]\end{aligned}$$

- Daraus folgt  $\text{Ws}[PubK_{\mathcal{A}, \Pi}^{mult}(n)] + \frac{1}{2} \geq 1 + \text{negl}(n)$ .  $\square$  Satz für  $t = 2$

## Von fester zu beliebiger Nachrichtenlänge

- Beweistechnik für allgemeines  $t$ : Definiere für  $i \in [t]$  Hybride  $C^{(i)} = (Enc_{pk}(m_0^1), \dots, Enc_{pk}(m_0^i), Enc_{pk}(m_1^{i+1}), \dots, Enc_{pk}(m_1^t))$ .
- $Ws[PubK_{\mathcal{A}, \Pi}^{mult}(n) = 1] = \frac{1}{2} \cdot Ws[\mathcal{A}(C^{(t)}) = 0] + \frac{1}{2} \cdot Ws[\mathcal{A}(C^{(0)}) = 1]$ .
- $\mathcal{A}'$  unterscheidet  $Enc_{pk}(m_0^i)$  und  $Enc_{pk}(m_1^i)$  für zufälliges  $i \in [t]$ .
- Entspricht dem Unterscheiden von  $C^{(i)}$  und  $C^{(i-1)}$ .
- Liefert  $Pr[PubK_{\mathcal{A}, \Pi}^{mult}(n)] \leq \frac{1}{2} + t \cdot \text{negl}(n)$   $\square$  Satz.

## Von fester zu beliebiger Nachrichtenlänge

- Sei  $\Pi$  ein Verschlüsselungsverfahren mit Klartexten aus  $\{0, 1\}^n$ .
- Splitte  $m \in \{0, 1\}^*$  auf in  $m_1, \dots, m_t$  mit  $m_i \in \{0, 1\}^n$ .
- Definiere  $\Pi'$  mittels  $Enc'_{pk}(m) = Enc_{pk}(m_1) \dots Enc_{pk}(m_t)$ .
- Aus vorigem Satz folgt:  $\Pi'$  ist CPA-sicher, falls  $\Pi$  CPA-sicher ist.

# Hybride Verschlüsselungsverfahren

**Ziel:** Flexibilität von asym. Verfahren und Effizienz von sym. Verfahren.

**Szenario:**

- Sei  $\Pi = (Gen, Enc, Dec)$  ein PK-Verschlüsselungsverfahren und  $\Pi' = (Gen', Enc', Dec')$  ein SK-Verschlüsselungsverfahren.
- Berechne  $(pk, sk) \leftarrow Gen(1^n)$ .

## Algorithmus Hybride Verschlüsselung

**Eingabe:**  $m, pk$

- 1 Wähle  $k \in_R \{0, 1\}^n$ .
- 2 Verschlüssele  $c_1 \leftarrow Enc_{pk}(k)$  mit asym. Verschlüsselung.
- 3 Verschlüssele  $c_2 \leftarrow Enc'_k(m)$  mit sym. Verschlüsselung.

**Ausgabe:** Chiffretext  $c = (c_1, c_2)$

# Hybride Entschlüsselung

## Algorithmus Hybride Entschlüsselung

**Eingabe:**  $c = (c_1, c_2)$ ,  $sk$

- 1 Entschlüssele  $k \leftarrow Dec_{sk}(c_1)$ .
- 2 Entschlüssele  $m \leftarrow Dec'_k(c_2)$ .

**Ausgabe:** Klartext  $m$

- Effizienzgewinn für  $|m| \gg n$ , sofern  $\Pi'$  effizienter ist als  $\Pi$ .
- **Frage:** Ist hybride Verschlüsselung sicher, falls  $\Pi, \Pi'$  sicher?

# Sicherheit von hybrider Verschlüsselung

## Satz Sicherheit hybrider Verschlüsselung

Sei  $\Pi$  ein CPA-sicheres PK-Verschlüsselungsverfahren und  $\Pi'$  ein KPA-sicheres SK-Verschlüsselungsverfahren.  
Dann ist das hybride Verfahren  $\Pi^{hy}$  CPA-sicher.

### Beweisskizze:

- **Notation**  $X \equiv Y$ : Kein ppt Angreifer kann  $X$  und  $Y$  unterscheiden.
- Sicherheit von  $\Pi'$ :  $Enc_k(m_0) \equiv Enc_k(m_1)$  für  $k \in_R \{0, 1\}^n$ .
- Sicherheit von  $\Pi^{hy}$ : Müssen zeigen dass
$$(Enc_{pk}(k), Enc'_k(m_0)) \equiv (Enc_{pk}(k), Enc'_k(m_1)).$$
- Problem: Erstes Argument könnte beim Unterscheiden des zweiten Arguments helfen.

# Beweis Sicherheit hybrider Verschlüsselung

**Beweisskizze:** Zeigen die folgenden 3 Schritte

- ① Sicherheit von  $\Pi$  liefert

$$(Enc_{pk}(k), Enc'_k(m_0)) \equiv (Enc_{pk}(0^n), Enc'_k(m_0)).$$

Gilt sogar falls  $\mathcal{A}$  die Werte  $k, 0^n$  kennt.

Dass das zweite Argument  $k$  beinhaltet ist daher kein Problem.

- ② Sicherheit von  $\Pi'$  liefert

$$(Enc_{pk}(0^n), Enc'_k(m_0)) \equiv (Enc_{pk}(0^n), Enc'_k(m_1)).$$

Kein Problem mehr, da 2. Argument nicht vom ersten abhängt.

- ③ Sicherheit von  $\Pi$  liefert

$$(Enc_{pk}(0^n), Enc'_k(m_1)) \equiv (Enc_{pk}(k), Enc'_k(m_1)).$$

Transitivität der 3 Ergebnisse liefert schließlich wie gewünscht

$$(Enc_{pk}(k), Enc'_k(m_0)) \equiv (Enc_{pk}(k), Enc'_k(m_1)).$$

# Textbook RSA

## Algorithmus Schlüsselerzeugung *GenRSA*

**Eingabe:**  $1^n$

- 1  $(N, p, q) \leftarrow \text{GenModulus}(1^n)$  mit primen  $n$ -Bit  $p, q$  und  $N = pq$ .
- 2  $\phi(N) \leftarrow (p - 1)(q - 1)$
- 3 Wähle  $e \in \mathbb{Z}_{\phi(N)}^*$ .
- 4 Berechne  $d \leftarrow e^{-1} \bmod \phi(N)$ .

**Ausgabe:**  $(N, e, d)$  mit  $pk = (N, e)$  und  $sk = (N, d)$ .

## Definition Textbook RSA Verschlüsselungsverfahren

Sei  $n$  ein Sicherheitsparameter.

- 1 **Gen** :  $(N, e, d) \leftarrow \text{GenRSA}(1^n)$
- 2 **Enc** :  $c \leftarrow m^e \bmod N$  für eine Nachricht  $m \in \mathbb{Z}_N$ .
- 3 **Dec** :  $m \leftarrow c^d \bmod N$

# RSA Problem und Sicherheit von RSA

## Definition RSA Problem

Das *RSA Problem* ist hart bezüglich  $GenRSA(1^n)$ , falls für alle ppt Algorithmen  $\mathcal{A}$  gilt  $Ws[\mathcal{A}(N, e, m^e \bmod N) = m] \leq \text{negl}(n)$ .

Wsraum: Wahl  $m \in_R \mathbb{Z}_N$  und interne Münzwürfe von  $\mathcal{A}$ ,  $GenRSA$ .

*RSA Annahme*: Das RSA Problem ist hart bezüglich  $GenRSA$ .

## Anmerkungen:

- Falls  $N$  effizient faktorisiert werden kann, ist das RSA Problem nicht hart. (Warum?)
- Berechnen von  $d$  ist so schwer wie Faktorisieren von  $N$ .
- **Offenes Problem:** Impliziert eine Lösung des RSA Problem eine Lösung des Faktorisierungsproblems?
- *Enc* ist deterministisch, d.h. Textbook RSA ist nicht CPA-sicher.
- Unter der RSA-Annahme: Kein ppt Angreifer kann für zufällige  $m \in \mathbb{Z}_N^*$  aus  $(N, e, m^e \bmod N)$  die ganze Nachricht  $m$  berechnen.

# Angriffe auf Textbook RSA

Verschlüsseln von kurzen Nachrichten mit kleinem  $e$

- Sei  $m < N^{\frac{1}{e}}$ . Dann gilt  $c = m^e < N$ . D.h.  $c^{\frac{1}{e}}$  über  $\mathbb{Z}$  liefert  $m$ .
- Realistisch bei hybrider Verschlüsselung:  $N$  1024-Bit und  $e = 3$ ,  $m$  ist 128-Bit Schlüssel für symmetrische Verschlüsselung.

## Hastad Angriff auf RSA

- **Szenario:** Verschlüsselung desselben  $m$  unter verschiedenen  $pk$ .
- Sei  $pk_1 = (N_1, 3)$ ,  $pk_2 = (N_2, 3)$ ,  $pk_3 = (N_3, 3)$ . Angreifer erhält  $c_1 = m^3 \bmod N_1$ ,  $c_2 = m^3 \bmod N_2$  und  $c_3 = m^3 \bmod N_3$ .
- Berechne mittels Chinesischem Restsatz eind.  $c \in \mathbb{Z}_{N_1 N_2 N_3}$  mit

$$\begin{cases} c = c_1 \bmod N_1 \\ c = c_2 \bmod N_2 \\ c = c_3 \bmod N_3 \end{cases}.$$

- Es gilt  $c = m^3 < (\min\{N_1, N_2, N_3\})^3 < N_1 N_2 N_3$ , d.h.  $m = c^{\frac{1}{3}}$ .

# Padded RSA

## Definition Padded RSA Verschlüsselungsverfahren

Sei  $n$  ein Sicherheitsparameter und  $\ell$  eine Fkt. mit  $\ell(n) \leq 2n - 2$ .

- 1 **Gen** :  $(N, e, d) \leftarrow \text{GenRSA}(1^n)$
- 2 **Enc** : Für  $m \in \{0, 1\}^{\ell(n)}$  und  $r \in_R \{0, 1\}^{|N| - \ell(n) - 1}$  berechne
$$c \leftarrow (r||m)^e \bmod N.$$
- 3 **Dec** :  $r||m := c^d \bmod N$ . Gib die untersten  $\ell(n)$  Bits aus.

## Anmerkungen

- Für  $\ell(n) = 2n - \mathcal{O}(\log n)$  kann  $r$  in polyn. Zeit geraten werden.
- Für  $\ell(n) = cn$ , konstantes  $c < 2$ , ist kein CPA Angriff bekannt.
- Für  $\ell(n) = \mathcal{O}(\log n)$  kann CPA-Sicherheit gezeigt werden.
- Weitverbreitete standardisierte Variante von Padded RSA: PKCS #1 version 1.5 mit  $c := (0^8||0^610||r||0^8||m)^e \bmod N$ .

# Einfache **symmetrische** Verschlüsselung

## Algorithmus ONE-TIME GRUPPENELEMENT

Sei  $n$  ein Sicherheitsparameter.

- 1 **Gen:** Schlüsselerzeugung  $(G, g) \leftarrow \mathcal{G}(1^n)$ , wobei  $G$  eine Gruppe und  $g \in_R G$  ein zufälliger gemeinsamer geheimer Schlüssel ist.
- 2 **Enc:** Verschlüssele  $m \in G$  als  $c \leftarrow m \cdot g$ .
- 3 **Dec:** Entschlüssele  $c \in G$  als  $m := c \cdot g^{-1}$ .

### Anmerkung:

- Das One-Time Pad erhalten wir als Spezialfall mit  $(G, \cdot) = (\mathbb{F}_2^n, +)$ .

# Perfekte Sicherheit von ONE-TIME ELEMENT

## Satz Perfekte Sicherheit von ONE-TIME ELEMENT (Krypto I)

ONE-TIME ELEMENT  $\Pi$  ist eine perfekt sichere **symmetrisches** Verschlüsselung, d.h. für alle Angreifer  $\mathcal{A}$  gilt  $W_s[\text{PrivK}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2}$ .

### Beweis:

- Klartext-, Chiffretext und Schlüsselraum sind identisch

$$\mathcal{M} = \mathcal{C} = \mathcal{K} = G.$$

- Jeder Schlüssel  $g \in G$  wird gleichverteilt mit  $W_s \frac{1}{|G|}$  gewählt.
- Für jedes  $m, c \in G$  existiert genau ein Schlüssel  $g = c \cdot m^{-1}$  mit

$$c = \text{Enc}_g(m).$$

- Nach dem Satz von Shannon ist  $\Pi$  damit perfekt sicher. D.h.

$$W_s[\text{PrivK}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2}.$$

### Anmerkungen:

- Geheimer Schlüssel  $sk = g$  muss stets neu gewählt werden.
- Idee für PK-Verfahren: Ersetze das zufällige  $g$  durch ein stets neu gewähltes “pseudozufälliges” Gruppenelement.

# ElGamal Verschlüsselungsverfahren (1984)

## Definition ElGamal Verschlüsselungsverfahren

Sei  $n$  ein Sicherheitsparameter.

- Gen** :  $(q, g) \leftarrow \mathcal{G}(1^n)$ , wobei  $g$  eine Gruppe  $G$  der Ordnung  $q$  generiert. Wähle  $x \in_R \mathbb{Z}_q$  und berechne  $h \leftarrow g^x$ .  
Schlüssel:  $pk = (q, g, h)$ ,  $sk = (q, g, x)$
- Enc** : Für eine Nachricht  $m \in G$  wähle ein  $y \in_R \mathbb{Z}_q$  und berechne  
$$c \leftarrow (g^y, h^y \cdot m).$$
- Dec** : Für einen Chiffretext  $c = (c_1, c_2)$  berechne  $m := \frac{c_2}{c_1^x}$ .

- Korrektheit**:  $\frac{c_2}{c_1^x} = \frac{h^y \cdot m}{(g^y)^x} = \frac{(g^x)^y \cdot m}{g^{xy}} = m.$
- $c_2$  ist ein Analog von *Enc* bei ONE-TIME GRUPPENELEMENT mit einem DH-Schlüssel  $g^{xy}$  als "pseudozufälligem" Gruppenelement.

## Anmerkung:

- Parameter  $q, g$  können global für alle Teilnehmer gewählt werden.

# Sicherheit von ElGamal

## Satz CPA-Sicherheit ElGamal

ElGamal  $\Pi$  ist CPA-sicher unter der DDH-Annahme.

### Beweis:

- Sei  $\mathcal{A}$  ein Angreifer auf ElGamal  $\Pi$  mit Erfolgsws

$$\epsilon(n) := \text{Ws}[PubK_{\mathcal{A}, \Pi}^{cpa}(n) = 1].$$

- Wir konstruieren mittels  $\mathcal{A}$  einen Unterscheider  $D$  zum Unterscheiden von DDH-Instanzen

$$(q, g, g^x, g^y, g') \text{ mit } g' = g^{xy} \text{ oder } g' = g^z.$$

# Unterscheider für DDH durch $\mathcal{A}$

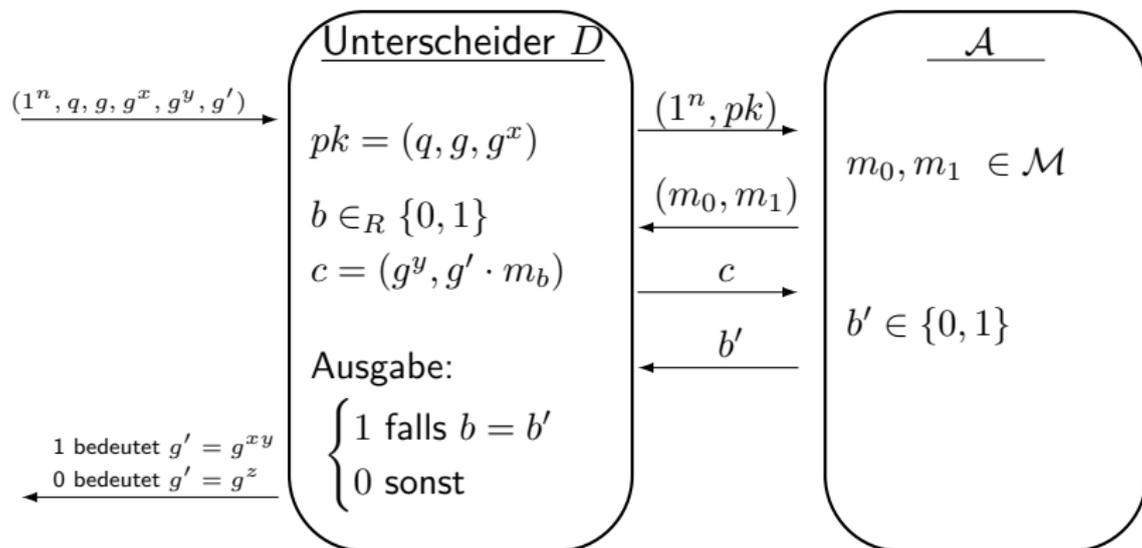
## Algorithmus DDH-Unterscheider $D$

EINGABE:  $(q, g, g^x, g^y, g')$

- 1 Setze  $pk = (G, q, g, g^x)$ .
- 2  $(m_0, m_1) \leftarrow \mathcal{A}(pk)$ .
- 3 Wähle  $b \in_R \{0, 1\}$  und berechne  $b' \leftarrow \mathcal{A}(g^y, g' \cdot m_b)$ .
- 4 Falls  $b' = b$  Ausgabe 1, sonst Ausgabe 0.

AUSGABE:  $\begin{cases} 1 & \text{wird interpretiert als } g' = g^{xy} \\ 0 & \text{wird interpretiert als } g' = g^z \end{cases}$

# DDH-Unterscheider mit Angreifer $\mathcal{A}$



## Fall DDH-Tupel

**Fall 1:** DDH-Tupel, d.h.  $g' = g^{xy}$ .

- $c = (g^y, g^{xy} \cdot m_b)$  ist identisch zu ElGamal-Chiffretexten verteilt.
- D.h.  $\text{Ws}[D(q, g, g^x, g^y, g^{xy}) = 1] = \text{Ws}[\text{PubK}_{\mathcal{A}, \Pi}(n) = 1] = \epsilon(n)$ .

**Fall 2:** kein DDH-Tupel, d.h.  $g' = g^z$  für  $z \in_R \mathbb{Z}_q$ .

- Chiffretext  $c = (c_1, c_2)$  ist von der Form  $(g^y, g^z \cdot m_b)$ .
- $c_2$  ist identisch verteilt zu ONE TIME ELEMENT.
- Die erste Komponente  $c_1$  ist unabhängig von  $c_2$ , d.h.

$$\text{Ws}[\text{PrivK}_{\mathcal{A}, \Pi}(n) = 1] = \frac{1}{2}.$$

- Es folgt  $\text{Ws}[D(G, q, g, g^x, g^y, g^z) = 1] = \text{Ws}[\text{PubK}_{\mathcal{A}, \Pi}(n) = 1] = \frac{1}{2}$ .

Aus der DDH-Annahme folgt

$$\begin{aligned} \text{negl}(n) &\geq |\text{Ws}[D(G, q, g, g^x, g^y, g^z) = 1] - \text{Ws}[D(G, q, g, g^x, g^y, g^{xy}) = 1]| \\ &= \left| \frac{1}{2} - \epsilon(n) \right|. \end{aligned}$$

Damit gilt  $\epsilon(n) \leq \frac{1}{2} + \text{negl}(n)$ .

# Parameterwahl bei ElGamal

## Einbetten von Nachrichten $m' \in \{0, 1\}^*$

- Beliebte Parameterwahl:  $\mathbb{Z}_p^*$ ,  $p = 2q + 1$  mit  $p, q$  prim.
- D.h.  $p$  ist eine sogenannte starke Primzahl.
- **Ziel:** Konstruktion einer Untergruppe  $G$  mit Ordnung  $q$ .
- Quadrieren  $\mathbb{Z}_p^* \rightarrow \mathbb{Z}_p^*$ ,  $x \mapsto x^2$  ist eine  $(2 : 1)$ -Abbildung.
- Urbilder  $x, p - x$  kollidieren, genau eines ist in  $[\frac{p-1}{2}] = [q]$ .
- Wir bezeichnen den Bildraum mit  $QR_p$ .
- $QR_p$  ist Untergruppe von  $\mathbb{Z}_p^*$  mit Ordnung  $q$ .
- Wählen  $g$  als Generator von  $QR_p$ . Sei  $|q| = n$ .
- Interpretieren  $m' \in \{0, 1\}^{n-1}$  als natürliche Zahl kleiner  $q$ .
- Es gilt  $m' + 1 \in [q]$ . Einbettung von  $m'$  ist  $m = (m' + 1)^2 \bmod p$ .
- Umkehren der Einbettung ist effizient berechenbar.

# CPA-Sicherheit ist ungenügend

## Definition CCA (informal)

CCA (=Chosen Ciphertext Attack) ist ein Angriff, bei dem der Angreifer sich Chiffretext seiner Wahl entschlüsseln lassen kann.

**Beispiele** in denen CPA-Sicherheit nicht genügt:

- Eve fängt verschlüsselte Email  $c = Enc(m)$  an Bob ab.
- Eve verschickt  $c$  selbst an Bob.
- Bob antwortet Eve und hängt dabei  $m$  an die Antwort an.
- D.h. Bob fungiert als Entschlüsselungssorakel.
  
- Alice und Eve nehmen als Bieter an einer Auktion von Bob teil.
- Alice sendet ihr Gebot  $c = Enc(m)$  verschlüsselt an Bob.
- $Enc$  soll CPA-sicher sein, d.h. Eve erhält keine Information über  $m$ .
- **Frage:** Ist es Eve möglich,  $c' = Enc(2m)$  aus  $c$  zu berechnen, ohne  $m$  zu kennen, und damit Alice zu überbieten? (Malleability)
- Es gilt: CCA-Sicherheit impliziert Non-Malleability. (ohne Beweis)

# CCA Ununterscheidbarkeit

## Spiel CCA Ununterscheidbarkeit von Chiffretexten $PubK_{\mathcal{A}, \Pi}^{cca}(n)$

Sei  $\Pi$  ein PK-Verschlüsselungsverfahren mit Angreifer  $\mathcal{A}$ .

- 1  $(pk, sk) \leftarrow Gen(1^n)$
- 2  $(m_0, m_1) \leftarrow \mathcal{A}^{Dec_{sk}(\cdot)}(pk)$ , wobei  $Dec_{sk}(\cdot)$  ein Entschlüsselungs-orakel für  $\mathcal{A}$  für beliebige Chiffretexte ist.
- 3 Wähle  $b \in_R \{0, 1\}$ . Verschlüssele  $c \leftarrow Enc_{pk}(m_b)$ .
- 4  $b' \leftarrow \mathcal{A}^{Dec_{sk}(\cdot)}(c)$ , wobei  $\mathcal{A}$  beliebige Chiffretexte  $c' \neq c$  durch das Orakel  $Dec_{sk}(\cdot)$  entschlüsseln lassen darf.
- 5  $PubK_{\mathcal{A}, \Pi}^{cca}(n) = \begin{cases} 1 & \text{für } b = b' \\ 0 & \text{sonst} \end{cases}$

### Anmerkungen:

- Zusätzlich zum Verschlüsselungs-Orakel bei CPA besitzt  $\mathcal{A}$  bei CCA ein weiteres Entschlüsselungs-Orakel  $Dec_{sk}(\cdot)$ .
- Falls  $\mathcal{A}$  in Schritt 4 auch  $c$  entschlüsseln darf, ist das Spiel trivial.

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

$(pk, sk) \leftarrow \text{Gen}(1^n)$

$m'_1 = \text{Dec}_{sk}(c'_1)$

$m'_i = \text{Dec}_{sk}(c'_i)$

$b \in_R \{0, 1\}$

$c = \text{Enc}_{pk}(m_b)$

$m'_{i+1} = \text{Dec}_{sk}(c'_{i+1})$

$m'_q = \text{Dec}_{sk}(c'_q)$

Ausgabe:

$$= \begin{cases} 1 & \text{falls } b = b' \\ 0 & \text{sonst} \end{cases}$$

$(1^n, pk) \rightarrow$

$\leftarrow c'_1$

$m'_1 \rightarrow$

$\vdots$

$\leftarrow c'_i$

$m'_i \rightarrow$

$(m_0, m_1) \leftarrow$

$c \rightarrow$

$\leftarrow c'_{i+1}$

$m'_{i+1} \rightarrow$

$\vdots$

$\leftarrow c'_q$

$m'_q \rightarrow$

$\leftarrow b'$

$\underline{A}$

$c'_1 \in \mathcal{C}$

$c'_i \in \mathcal{C}, i \leq q$

$m_0, m_1 \in \mathcal{M}$

$c'_{i+1} \in \mathcal{C} \setminus \{c\}$

$c'_q \in \mathcal{C} \setminus \{c\}$

$b' \in \{0, 1\}$

## Definition CCA-Sicherheit

Ein Verschlüsselungsverfahren  $\Pi$  heißt *CCA-sicher* (bzw. besitzt *ununterscheidbare Chiffretexte unter CCA*), falls für alle ppt Angreifer  $\mathcal{A}$  gilt  $\text{Ws}[PubK_{\mathcal{A},\Pi}^{cca}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$ .

## Anmerkungen:

- Erstes effizientes CCA-sicheres Verfahren Cramer-Shoup (1998).
- Cramer-Shoup ist CCA-sicher unter der DDH-Annahme.
- Chiffretexte sind doppelt so lang wie bei ElGamal.
- Sicherheitsbeweis von Cramer-Shoup ist nicht-trivial.
- Später in der Vorlesung: CCA-sichere Verfahren im sogenannten Random Oracle Modell.

# CCA Angriff und Malleability von Textbuch RSA

## CCA Angriff auf Textbuch RSA

- Wollen  $c = m^e \bmod N$  entschlüsseln.
- Man beachte:  $c$  darf nicht direkt angefragt werden.
- Berechne  $c' = c \cdot r^e = (mr)^e \bmod N$  für  $r \in \mathbb{Z}_N^* \setminus \{1\}$ .
- Berechne  $mr \leftarrow Dec_{sk}(c')$  mittels Entschlüsselungs-Orakel.
- Berechne  $mr \cdot r^{-1} = m \bmod N$ .

## Malleability von Textbuch RSA

- Voriger Angriff zeigt: Für  $c = m^e \bmod N$  kann die Verschlüsselung von  $mr$  berechnet werden, ohne  $m$  selbst zu kennen.
- D.h. Textbuch RSA ist malleable.

# CCA Angriff und Malleability von ElGamal

## Praktischer CCA-Angriff auf Padded RSA Variante PKCS #1 v1.5

- Bleichenbacher Angriff: Sende adaptiv Chiffretexte an Server.
- Falls die Entschlüsselung nicht das korrekte Format besitzt, sendet der Server eine Fehlermeldung zurück.
- Genügt, um einen beliebigen Chiffretext  $c$  zu entschlüsseln.

## CCA Angriff auf ElGamal

- Ziel: Entschlüssele  $c = (g^y, g^{xy} \cdot m)$ .
- Lasse  $c' = (g^y, g^{xy} \cdot m \cdot r)$  für  $r \in G \setminus \{1\}$  entschlüsseln.
- Berechne  $mr \cdot r^{-1} = m$ .
- ElGamal ist malleable, da  $c'$  korrekte Verschlüsselung von  $mr$ .

# Einwegfunktionen

**Ziel:** CPA-sichere Verschlüsselung aus Trapdoor-Einwegpermutation

**Später:** CCA-sichere Verschlüsselung aus Trapdoor-Einwegperm.

## Spiel Invertieren $Invert_{\mathcal{A},f}(n)$

Sei  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  effizient berechenbar,  $\mathcal{A}$  ein Invertierer für  $f$ .

1 Wähle  $x \in_R \{0, 1\}^n$ . Berechne  $y \leftarrow f(x)$ .

2  $x' \leftarrow \mathcal{A}(1^n, y)$

3  $Invert_{\mathcal{A},f}(n) = \begin{cases} 1 & \text{falls } f(x') = y \\ 0 & \text{sonst} \end{cases}$ .

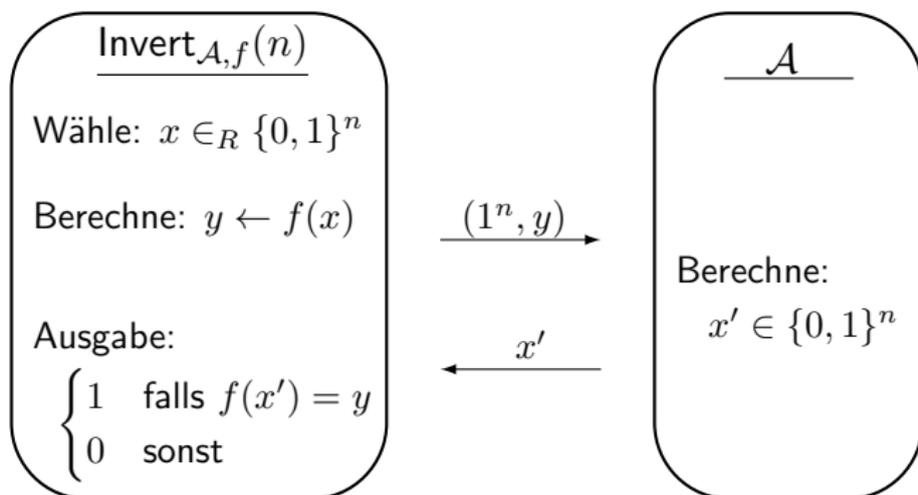
## Definition Einwegfunktion

Eine Funktion  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  heißt *Einwegfunktion*, falls

1 Es existiert ein deterministischer pt Alg  $\mathcal{B}$  mit  $f(x) \leftarrow \mathcal{B}(x)$ .

2 Für alle ppt Algorithmen  $\mathcal{A}$  gilt  $\text{Ws}[Invert_{\mathcal{A},f}(n) = 1] \leq \text{negl}(n)$ .

# Spiel Invertieren



# Die Faktorisierungsannahme

- **Problem:** Existenz von Einwegfunktionen ist ein offenes Problem.
- Konstruktion unter Komplexitätsannahme (z.B. Faktorisierung)
- Verwenden dazu  $(N, p, q) \leftarrow \text{GenModulus}(1^n)$  von RSA.

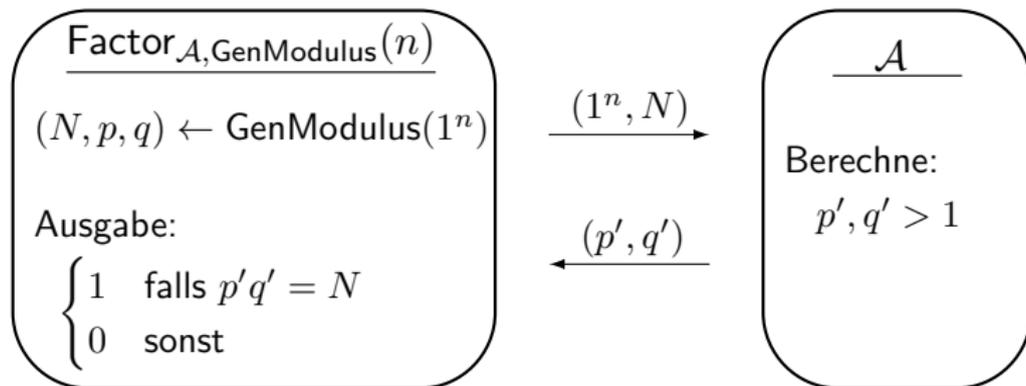
## Spiel Faktorisierungsspiel $\text{Factor}_{\mathcal{A}, \text{GenModulus}}(n)$

1  $(N, p, q) \leftarrow \text{GenModulus}(1^n)$

2  $(p', q') \leftarrow \mathcal{A}(N)$  mit  $p', q' > 1$ .

3 
$$\text{Factor}_{\mathcal{A}, \text{GenModulus}}(n) = \begin{cases} 1 & \text{falls } p'q' = N \\ 0 & \text{sonst} \end{cases} .$$

# Spiel Faktorisieren



## Definition Faktorisierungsannahme

Faktorisieren ist hart bezüglich *GenModulus* falls für alle ppt Algorithmen  $\mathcal{A}$  gilt  $\text{Ws}[Factor_{\mathcal{A}, GenModulus}(n) = 1] \leq \text{negl}(n)$ .

**Faktorisierungsannahme:** Faktorisieren ist hart bezüglich *GenModulus*.

# Konstruktion aus Faktorisierungsannahme

- Sei  $p(n)$  ein Polynom, so dass  $GenModulus(1^n)$  höchstens  $p(n)$  Zufallsbits verwendet.
- OBdA sei  $p(n) : \mathbb{N} \rightarrow \mathbb{N}$  monoton wachsend.

## Algorithmus FACTOR-ONEWAY $f_{FO}$

**Eingabe:**  $x \in \{0, 1\}^*$

- 1 Berechne  $n$  mit  $p(n) \leq |x| < p(n+1)$ .
- 2  $(N, p, q) := GenModulus(1^n, x)$ , wobei  $GenModulus$  die Eingabe  $x$  als internen Zufallsstring verwendet.

**Ausgabe:**  $N$

**Bemerkung:**

- $GenModulus(1^n, x)$  ist deterministisch. (Derandomisierung)

# Existenz von Einwegfunktionen

## Satz Einweg-Eigenschaft von $f_{FO}$

Unter der Faktorisierungsannahme ist  $f_{FO}$  eine Einwegfunktion.

### Beweis:

- Sei  $\mathcal{A}$  ein Invertierer für  $f_{FO}$  mit Erfolgsws  $\epsilon(n)$ .
- Konstruieren mit  $\mathcal{A}$  Faktorisierer  $\mathcal{A}'$  im Spiel  $Factor_{\mathcal{A}', GenModulus}(n)$ .

## Algorithmus Faktorisierer $\mathcal{A}'$

EINGABE:  $1^n, N$

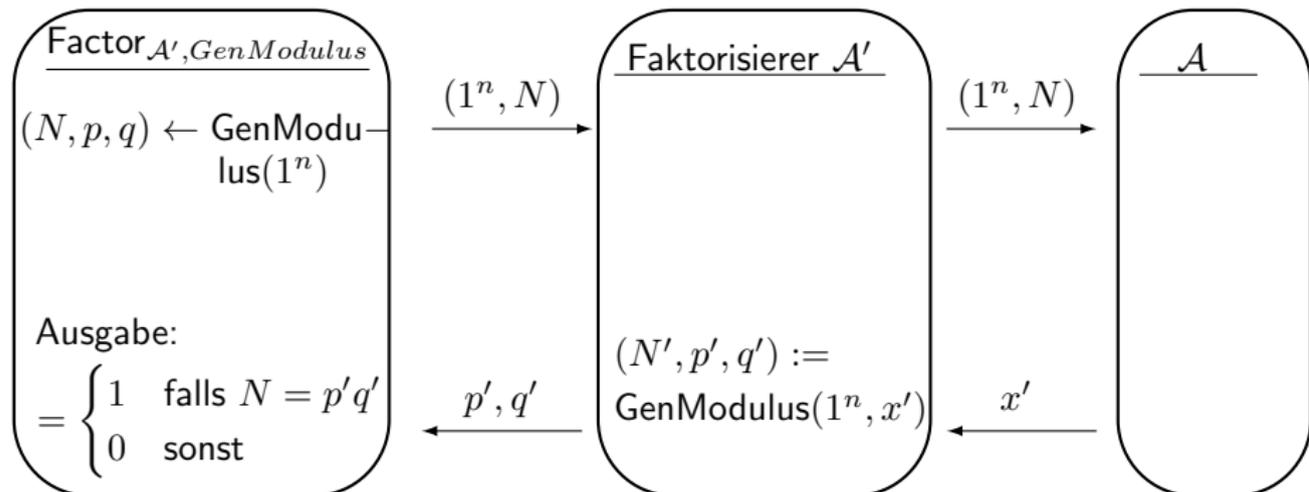
- 1  $x' \leftarrow \mathcal{A}(1^n, N)$ .
- 2  $(N', p', q') \leftarrow GenModulus(1^n, x')$ .

AUSGABE:  $p', q'$

Unter der Faktorisierungsannahme gilt

$$\text{negl}(n) \geq \text{Ws}[Factor_{\mathcal{A}', GenModulus}(n) = 1] = \text{Ws}[Invert_{\mathcal{A}, f_{FO}}(n) = 1] = \epsilon(n).$$

# Faktorisieren mit Invertierer für $f_{FO}$



# Trapdoor-Permutationsfamilie

## Definition Permutationsfamilie

Eine *Permutationsfamilie*  $\Pi_f = (Gen, Samp, f)$  besteht aus 3 ppt Alg:

- 1  $I \leftarrow Gen(1^n)$ , wobei  $I$  eine Urbildmenge  $D$  für  $f$  definiert.
- 2  $x \leftarrow Samp(I)$ , wobei  $x \in_R D$ .
- 3  $y := f_I(x)$  und  $f : D \rightarrow D$  ist bijektiv.

## Definition Trapdoor-Permutationsfamilie

*Trapdoor-Permutationsfamilie*  $\Pi_f = (Gen, Samp, f, Inv)$  besteht aus

- 1  $(I, td) \leftarrow Gen(1^n)$  mit  $td$  als Trapdoor-Information.
- 2  $x \leftarrow Samp(I)$  wie zuvor.
- 3  $y := f_I(x)$  wie zuvor.
- 4  $x \leftarrow Inv_{td}(y)$  mit  $Inv_{td}(f_I(x)) = x$  für alle  $x \in D$ .

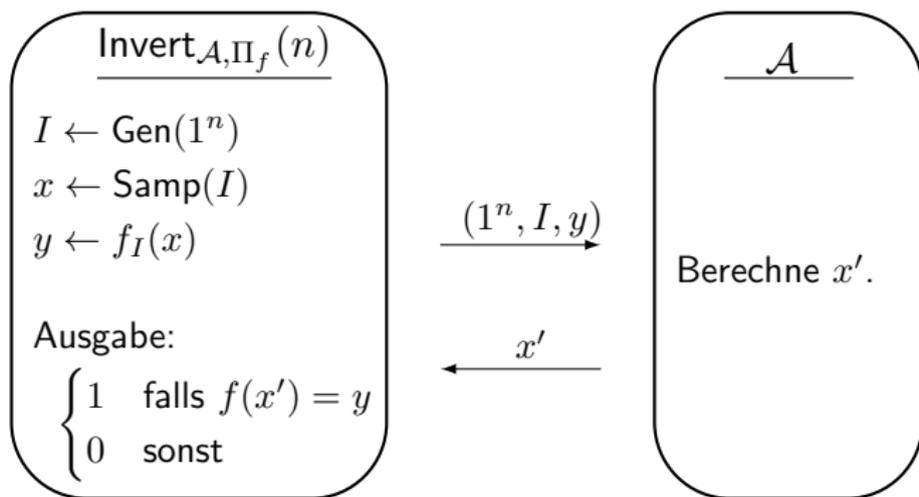
## Spiel Invertieren einer Permutation $Invert_{\mathcal{A}, \Pi_f}(n)$

Sei  $\mathcal{A}$  ein Invertierer für die Familie  $\Pi_f$ .

1  $I \leftarrow Gen(1^n)$ ,  $x \leftarrow Samp(I)$  und  $y \leftarrow f(I, x)$ .

2  $x' \leftarrow \mathcal{A}(I, y)$ .

3  $Invert_{\mathcal{A}, \Pi_f}(n) = \begin{cases} 1 & \text{falls } f(x') = y \\ 0 & \text{sonst} \end{cases}$ .



# Konstruktion einer Trapdoor-Einwegpermutation

## Definition Einweg-Permutation

Eine (Trapdoor-)Permutationsfamilie heißt *(Td-)Einwegpermutation* falls für alle ppt Algorithmen  $\mathcal{A}$  gilt  $\text{Ws}[Invert_{\mathcal{A}, \Pi_f}(n) = 1] \leq \text{negl}(n)$ .

## Bsp: Trapdoor-Einwegpermutation unter RSA-Annahme

- **Gen( $1^n$ ):**  
 $(N, e, d) \leftarrow \text{GenRSA}(1^n)$ , Ausgabe  $I = (N, e)$  und  $td = (N, d)$ .
- **Samp( $I$ ):**  
Wähle  $x \in_R \mathbb{Z}_N$ .
- **$f_I(x)$ :**  
Berechne  $y := x^e \bmod N$ .
- **$Inv_{td}(y)$ :**  
Berechne  $x := y^d \bmod N$ .

# Hardcore-Prädikat

**Ziel:** Destilliere Komplexität des Invertierens auf ein Bit.

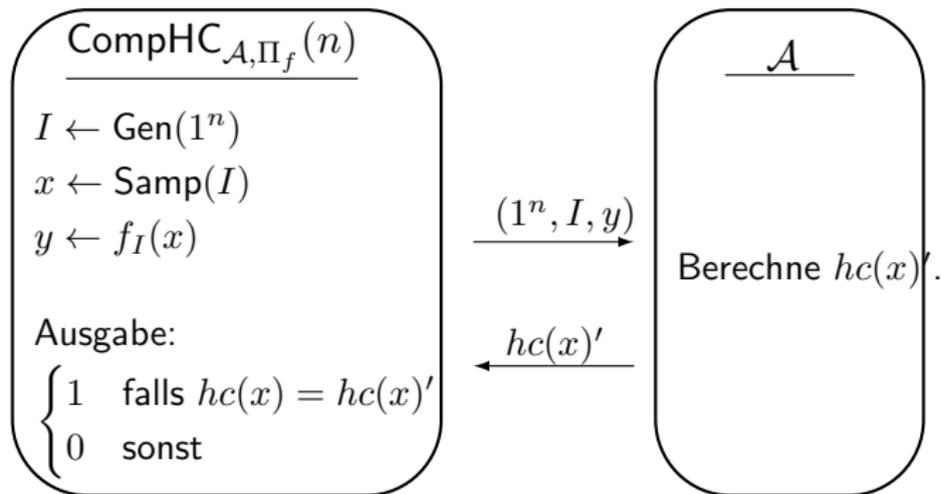
## Definition Hardcore-Prädikat

Sei  $\Pi_f$  eine Einwegpermutation. Sei  $hc$  ein deterministischer pt Alg mit Ausgabe eines Bits  $hc(x)$  bei Eingabe  $x \in D$ .  $hc$  heißt *Hardcore-Prädikat* für  $f$  falls für alle ppt Algorithmen  $\mathcal{A}$  gilt:

$$\text{Ws}[\mathcal{A}(1^n, f(x)) = hc(x)] \leq \frac{1}{2} + \text{negl}(n).$$

**Intuition:** Bild  $f(x)$  hilft nicht beim Berechnen von  $hc(x)$ .

# Spiel zum Berechnen des Hardcore-Prädikats



Falls  $hc$  ein Hardcoreprädikat ist, so gilt für alle ppt  $\mathcal{A}$

$$\mathbb{W}_s[\text{CompHC}_{\mathcal{A}, \Pi_f}(n) = 1] = \mathbb{W}_s[\mathcal{A}(1^n, I, f(x)) = hc(x)] \leq \frac{1}{2} + \text{negl}(n).$$

# Goldreich-Levin Hardcore-Prädikat

## Satz von Goldreich-Levin

Sei  $\Pi_f$  eine Einwegpermutation. Dann existiert eine Einwegpermutation  $\Pi_g$  mit Hardcoreprädikat  $hc$ .

**Konstruktion:** (ohne Beweis)

- Sei  $f$  eine Einwegpermutation mit Definitionsbereich  $\{0, 1\}^n$ .
- Sei  $x = x_1 \dots x_n \in \{0, 1\}^n$ . Konstruiere

$$g(x, r) := (f(x), r) \text{ mit } r \in_R \{0, 1\}^n.$$

- Offenbar ist  $g$  ebenfalls eine Einwegpermutation.
- Wir konstruieren ein Hardcore-Prädikat  $hc$  für  $g$  mittels

$$hc(x, r) = \langle x, r \rangle = \sum_{i=1}^n x_i r_i \bmod 2.$$

- Beweis der Hardcore-Eigenschaft ist nicht-trivial.

# Verschlüsselung aus Trapdoor-Einwegpermutation

## Algorithmus $\Pi_{cpa}$

Sei  $\Pi_f$  eine Td-Einwegpermutation mit Hardcore-Prädikat  $hc$ .

- 1 **Gen:**  $(I, td) \leftarrow Gen(1^n)$ . Ausgabe  $pk = I$  und  $sk = td$ .
- 2 **Enc:** Für  $m \in \{0, 1\}$  setze  $x \leftarrow Sample(I)$  und berechne
$$c \leftarrow (f(x), hc(x) \oplus m).$$
- 3 **Dec:** Für Chiffretext  $c = (c_1, c_2)$  berechne  $x := Inv_{td}(c_1)$  und
$$m := c_2 \oplus hc(x).$$

## Intuition:

- $hc(x)$  ist “pseudozufällig” gegeben  $f(x)$ .
- D.h.  $hc(x) \oplus m$  ist ununterscheidbar von 1-Bit One-Time Pad.

# Bsp: Verschlüsselung mit RSA-Td-Einwegpermutation

## Algorithmus $\Pi_{cpa}^{rsa}$

Sei  $\Pi_{rsa}$  die RSA Td-Einwegpermutation mit Hardcore-Prädikat  $hc$ .

- 1 **Gen:**  $(N, e, d) \leftarrow GenRSA(1^n)$ . Ausgabe  $pk = (N, e)$  und  $sk = (N, d)$ .
- 2 **Enc:** Für  $m \in \{0, 1\}$  wähle  $r \in_R \mathbb{Z}_N^*$  und berechne  
$$c \leftarrow (r^e \bmod N, hc(r) \oplus m).$$
- 3 **Dec:** Für Chiffretext  $c = (c_1, c_2)$  berechne  $r := c_1^d \bmod N$  und  
$$m \leftarrow c_2 \oplus hc(r).$$

# CPA-Sicherheit unserer Konstruktion

## Satz CPA-Sicherheit von $\Pi_{cpa}$

Sei  $\Pi_f$  eine Trapdoor-Einwegpermutation mit Hardcore-Prädikat  $hc$ .  
Dann ist  $\Pi_{cpa}$  CPA-sicher.

### Beweis:

- Sei  $\mathcal{A}$  ein Angreifer mit Erfolgswhs  $\epsilon(n) = W_S[\text{PubK}_{\mathcal{A}, \Pi_f}^{cpa}(n) = 1]$ .
- OBdA  $(m_0, m_1) \leftarrow \mathcal{A}(pk)$  mit  $\{m_0, m_1\} = \{0, 1\}$ . (Warum?)
- Verwenden  $\mathcal{A}$ , um  $\mathcal{A}'$  im Spiel  $\text{CompHC}_{\mathcal{A}', \Pi_f}(n)$  zu konstruieren.

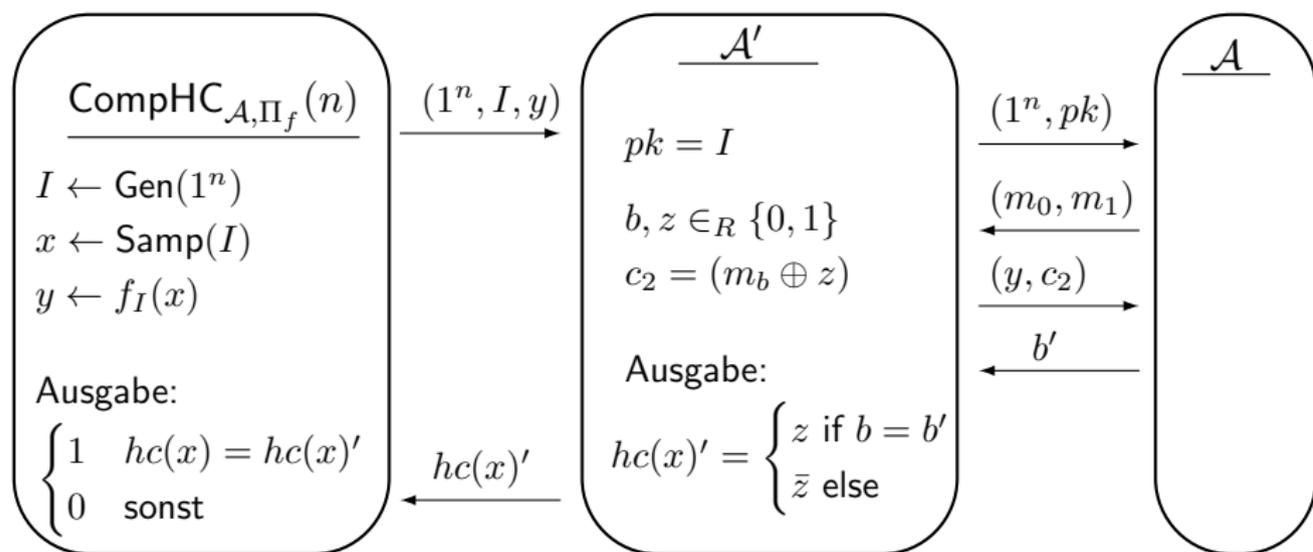
## Algorithmus Angreifer $\mathcal{A}'$

**Eingabe:**  $1^n, l, y = f(x) \in D$

- 1 Setze  $pk \leftarrow l$  und berechne  $(m_0, m_1) \leftarrow \mathcal{A}(pk)$ .
- 2 Wähle  $b, z \in_R \{0, 1\}$ . Setze  $c_2 \leftarrow m_b \oplus z$ .
- 3  $b' \leftarrow \mathcal{A}(y, c_2)$

**Ausgabe:**  $hc(x)' = \begin{cases} z & \text{falls } b = b' \\ \bar{z} & \text{sonst} \end{cases}$ .

# Angreifer $\mathcal{A}'$ für das Hardcore-Prädikat



# CPA-Sicherheit von $\Pi_{cpa}$

## Beweis: Fortsetzung

- Sei  $x = f^{-1}(y)$ .  $\mathcal{A}_{hc}$  rät  $z = hc(x)$ .
- Es gilt  $\text{Ws}[\mathcal{A}_{hc}(f(x)) = hc(x)] = \frac{1}{2} \cdot \text{Ws}[b = b' \mid z = hc(x)] + \frac{1}{2} \cdot \text{Ws}[b \neq b' \mid z \neq hc(x)]$ .
- **1. Fall**  $z = hc(x)$ :  $(y, c_2)$  ist korrekte Verschlüsselung von  $m_b$ , d.h.  
 $\text{Ws}[b = b' \mid z = hc(x)] = \epsilon(n)$ .

- **2. Fall**  $z \neq hc(x)$ : Es gilt

$$\begin{aligned}(y, c_2) &= (f(x), z \oplus m_b) \\ &= (f(x), z \oplus 1 \oplus m_b \oplus 1) = (f(x), hc(x) \oplus m_b \oplus 1).\end{aligned}$$

D.h.  $(y, c_2)$  ist korrekte Verschlüsselung von  $m_b \oplus 1 = m_{1-b}$ .

$$\text{Ws}[b \neq b' \mid z \neq hc(x)] = \text{Ws}[1 - b = b' \mid z \oplus 1 = hc(x)] = \epsilon(n).$$

Da  $hc$  ein Hardcore-Prädikat ist, folgt

$$\frac{1}{2} + \text{negl}(n) \geq \text{Ws}[\mathcal{A}_{hc}(f(x)) = hc(x)] = \epsilon(n).$$

# Random Oracle

## Definition Random Oracle

Sei  $\mathcal{F}^{n,\ell}$  die Menge aller Funktionen  $\{0, 1\}^n \rightarrow \{0, 1\}^{\ell(n)}$ . Ein *Random Oracle* ist eine zufällige Funktion  $H \in_R \mathcal{F}^{n,\ell}$ . Wir besitzen keine Beschreibung von  $H$ . Bei Anfrage  $x$  liefert das Random Oracle  $H(x)$ .

### Anmerkung: Bildliche Darstellung

- Ein Random Oracle  $H$  ist eine Funktion in einer schwarzen Box.
- $H$  ist beobachtbar über das Eingabe/Ausgabe-Verhalten der Box.

### Alternative Beschreibung eines Random Oracles

- Oracle erhält Anfragen  $x_1, \dots, x_q$ .
- Falls  $x_i \neq x_j$  für alle  $j < i$ , gib  $y_i \in_R \{0, 1\}^{\ell(n)}$  aus.
- Falls  $x_i = x_j$  für ein  $j < i$ , gib  $y_j$  aus.
- D.h. wir können uns vorstellen, dass das Orakel die Antworten auf Anfragen bei Bedarf erzeugt und konsistent beantwortet.
- Damit können Random Oracle durch ppt Alg. simuliert werden.

# Random Oracles liefern Einwegfunktionen

## Satz

Für polynomielles  $\ell(n)$  sind Random Oracles  $H$  Einwegfunktionen.

## Beweis:

- Sei  $x \in_R \{0, 1\}^n$  und  $y = H(x)$ . Wollen ein Urbild von  $y$  ermitteln.
- Jeder Angreifer  $\mathcal{A}$  stellt oBdA verschiedene Anfragen  $x_1, \dots, x_q$ . (Warum sollte jeder Angreifer so verfahren?)

- $\mathcal{A}$  gewinnt offenbar falls  $x_i = x$  für ein  $i \in [q]$ . Es gilt

$$\text{Ws}[x_i = x \text{ für ein } i] = \sum_{i=1}^q \text{Ws}[x_i = x] = \frac{q}{2^n}.$$

- $\mathcal{A}$  gewinnt ebenfalls für  $H(x_i) = y$  für mindestens ein  $i \in [q]$ , d.h.

$$\text{Ws}[H(x_i) = y \text{ für mindestens ein } i] \leq \sum_{i=1}^q \text{Ws}[H(x_i) = y] = \frac{q}{2^{\ell(n)}}.$$

- Damit gilt  $\text{Ws}[Invert_{\mathcal{A}, H}(n) = 1] \leq \frac{q}{2^n} + \frac{q}{2^{\ell(n)}}$ .
- Für polynomielles  $q, \ell(n)$  ist dies vernachlässigbar in  $n$ .

## Satz

Für polynomielles  $\ell(n)$  sind Random Oracles kollisionsresistent.

## Beweis:

- Jeder Angreifer  $\mathcal{A}$  stellt oBdA verschiedene Anfragen  $x_1, \dots, x_q$ .
- $\mathcal{A}$  gewinnt mit

$$\begin{aligned} \text{Ws}[H(x_i) = H(x_j) \text{ für ein } i \neq j] &\leq \sum_{i \neq j} \text{Ws}[H(x_i) = H(x_j)] \\ &= \binom{q}{2} \leq \frac{q^2}{2^{\ell(n)}}. \end{aligned}$$

- Dies ist vernachlässigbar für polynomielles  $q, \ell(n)$ .

# Random Oracle Methode

## Definition Random Oracle Modell / Methode

Das *Random Oracle Modell (ROM)* nimmt die Existenz von Random Oracles an. Die *Random Oracle Methode* besteht aus 2 Schritten:

- 1 Konstruiere ein Verfahren  $\Pi$  mit Hilfe eines Random Oracles  $H$  und beweise die Sicherheit von  $\Pi$  im ROM.
- 2 Instantiiere  $\Pi$  mit einer kryptographischen Hashfunktion  $H'$  anstelle von  $H$ , z.B. mit SHA-1.

### Negativ:

- Beschreibung von  $H'$  spezifiziert  $H'(x)$  für alle  $x$ .
- Es existieren künstliche Kryptosysteme, die sicher im Random Oracle Modell aber unsicher für *jede* Instantiierung von  $H'$  sind.

### Positiv:

- Ein Beweis im ROM ist besser als kein Beweis.
- Erfolgreicher Angriff muss die Instantiierung von  $H'$  attackieren.
- $H'$  kann leicht durch eine andere Hashfunktion ersetzt werden.



## Verschlüsselung ROM-RSA

Sei  $H : \mathbb{Z}_N^* \rightarrow \{0, 1\}^{\ell(n)}$  ein Random Oracle.

1 **Gen:**  $(N, e, d) \leftarrow \text{GenRSA}(1^n)$  mit  $pk = (N, e)$ ,  $sk = (N, d)$ .

2 **Enc:** Für  $m \in \{0, 1\}^{\ell(n)}$ , wähle  $r \in_R \mathbb{Z}_N^*$ . Berechne

$$c \leftarrow (r^e \bmod N, H(r) \oplus m).$$

3 **Dec:** Für  $c = (c_1, c_2)$  berechne

$$r := c_1^d \bmod N \text{ und } m := H(r) \oplus c_2.$$

# Sicherheit von RSA im Random Oracle Modell

## Satz CPA-Sicherheit von ROM-RSA

Unter der RSA-Annahme und für ein Random Oracle  $H$  ist ROM-RSA  $\Pi$  CPA-sicher.

### Beweis:

- Sei  $\mathcal{A}$  ein Angreifer mit Erfolgsws  $\epsilon(n) = \text{Ws}[PubK_{\mathcal{A},\Pi}^{cpa}(n) = 1]$ .
- $\mathcal{A}$  darf Orakelanfragen an  $H$  stellen, sowohl vor Ausgabe von  $(m_0, m_1)$  als auch nach Erhalt von  $Enc(m_b)$ .
- Definiere  $Query$  : Ereignis  $\mathcal{A}$  stellt Anfrage  $r = c_1^d \bmod N$  an  $H$ .

$$\begin{aligned}\epsilon(n) &= \text{Ws}[\overline{Query}] \cdot \text{Ws}[PubK_{\mathcal{A},\Pi}^{cpa}(n) = 1 \mid \overline{Query}] + \\ &\quad \text{Ws}[Query] \cdot \text{Ws}[PubK_{\mathcal{A},\Pi}^{cpa}(n) = 1 \mid Query] \\ &\leq \text{Ws}[PubK_{\mathcal{A},\Pi}^{cpa}(n) = 1 \mid \overline{Query}] + \text{Ws}[Query].\end{aligned}$$

- Zeigen

$$\text{Ws}[PubK_{\mathcal{A},\Pi}^{cpa}(n) = 1 \mid \overline{Query}] = \frac{1}{2} \text{ und } \text{Ws}[Query] \leq \text{negl}(n).$$

- Daraus folgt  $\epsilon(n) \leq \frac{1}{2} + \text{negl}(n)$ .

# Beweis der CPA-Sicherheit von ROM-RSA (1/3)

## Beweis:

- Falls  $r$  nicht an  $H$  angefragt wird, ist  $H(r) \oplus m$  nach Eigenschaft des Random Oracles ein perfektes One-Time Pad für  $m$ .
- Daraus folgt  $W_s[PubK_{\mathcal{A}, \Pi}^{cpa}(n) = 1 \mid \overline{Query}] = \frac{1}{2}$ .

## Beweis der CPA-Sicherheit von ROM-RSA (2/3)

**Beweis:**  $Ws[Query] \leq \text{negl}(n)$

- Idee: Verwende Anfragen von  $\mathcal{A}$ , um  $e$ -te Wurzeln zu berechnen.

### Algorithmus RSA-Invertierer $\mathcal{A}'$

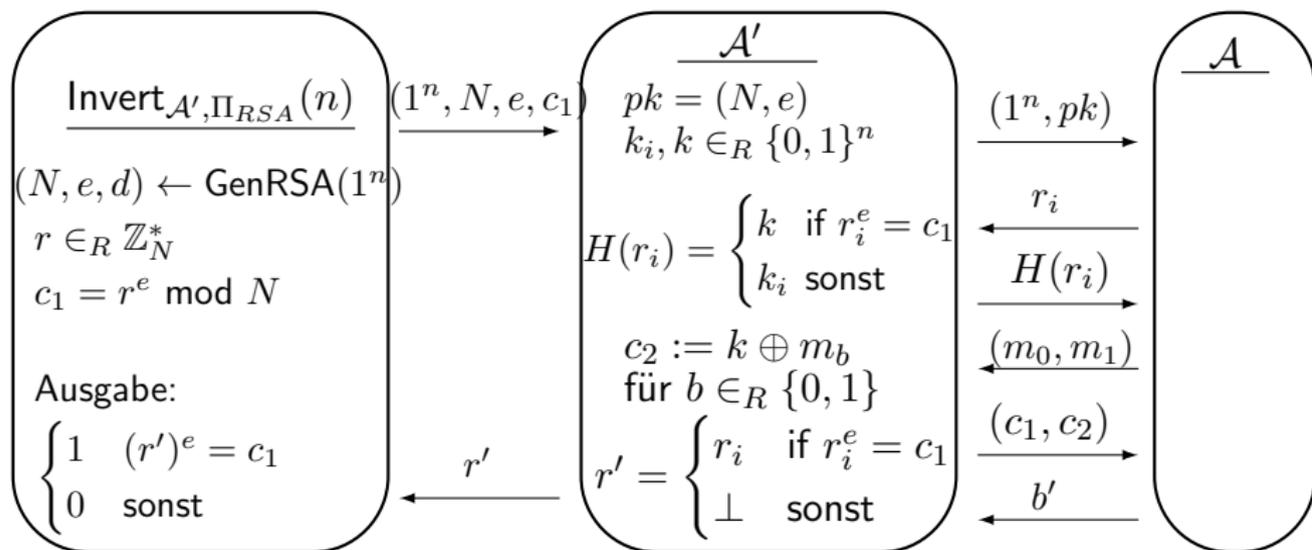
EINGABE:  $N, e, c_1 = r^e \bmod N$

- 1 Wähle  $k \in_R \{0, 1\}^{\ell(n)}$ . (Wir setzen  $H(r) = k$ , ohne  $r$  zu kennen.)
- 2  $(m_0, m_1) \leftarrow \mathcal{A}(N, e)$ , beantworte Orakelanfragen  $r_i$  an  $H(\cdot)$   
konsistent mit  $\begin{cases} k_i = k & \text{für } r_i^e = c_1 \bmod N \\ k_i \in_R \{0, 1\}^{\ell(n)} & \text{sonst} \end{cases}$ .
- 3 Berechne  $c \leftarrow (c_1, k \oplus m_b)$  für ein  $b \in_R \{0, 1\}$ .
- 4  $b' \leftarrow \mathcal{A}(c)$ , beantworte Anfragen von  $\mathcal{A}$  an  $H(\cdot)$  wie zuvor.
- 5 Falls  $r_i^e = c_1 \bmod N$  für eine der Orakelanfragen, setze  $r \leftarrow r_i$ .

AUSGABE:  $r$

- Es gilt  $Ws[Query] = Ws[\mathcal{A}'(N, e, r^e) = r] \leq \text{negl}(n)$ .

# Beweis der CPA-Sicherheit von ROM-RSA (3/3)



# Sicherheit gegenüber CCA

## Idee:

- Ersetze One-Time Pad durch CCA-sicheres Secret Key Verfahren.
- *Erinnerung Krypto I*: Konstruktion von CPA-sicherem Secret Key Verfahren mittels Pseudozufallsfunktion (und MAC) möglich.

## Verschlüsselung ROM-RSA-2

Sei  $H : \mathbb{Z}_N^* \rightarrow \{0, 1\}^{\ell(n)}$  ein Random Oracle,  $\Pi' = (Gen', Enc', Dec')$  ein CPA-sicheres Secret Key Verschlüsselungsverfahren.

- 1 **Gen:**  $(N, e, d) \leftarrow GenRSA(1^n)$  mit  $pk = (N, e)$ ,  $sk = (N, d)$ .
- 2 **Enc:** Für  $m \in \{0, 1\}^{\ell(n)}$ , wähle  $r \in_R \mathbb{Z}_N^*$ . Berechne  $k = H(r)$  und
$$c \leftarrow (r^e \bmod N, Enc'_k(m)).$$
- 3 **Dec:** Für  $c = (c_1, c_2)$  berechne
$$r \leftarrow c_1^d \bmod N, k \leftarrow H(r) \text{ und } m \leftarrow Dec'_k(c_2).$$

# Sicherheit von ROM-RSA-2

## Satz Sicherheit von ROM-RSA-2

Unter der RSA-Annahme, für ein Random Oracle  $H$  und ein CPA-sicheres  $\Pi'$  liefert ROM-RSA-2 CCA-sichere Verschlüsselung.

### Anmerkungen:

- Wir werden den Satz hier nicht formal beweisen.
- Der Beweis verläuft größtenteils analog zum vorigen Beweis.
- Problem: Müssen Orakel  $Dec_{sk}(\cdot)$  simulieren, ohne  $sk$  zu kennen.
- Verwende dazu geschicktes Simulieren des Random Oracles  $H(\cdot)$ .
- Bsp. für geschicktes Simulieren: s. Beweis zu RSA-FDH (später).

# Jacobi-Symbol

**Erinnerung Jacobi-Symbol:** Beweise siehe Diskrete Mathematik II

## Definition Quadratischer Rest

Sei  $N \in \mathbb{N}$ . Ein Element  $a \in \mathbb{Z}_N$  heißt *quadratischer Rest* in  $\mathbb{Z}_N$ , falls es ein  $b \in \mathbb{Z}_N$  gibt mit  $b^2 = a \pmod N$ . Wir definieren

$$QR_N = \{a \in \mathbb{Z}_N^* \mid a \text{ ist quadratischer Rest}\} \text{ und } QNR_N = \mathbb{Z}_N^* \setminus QR_N.$$

## Lemma Anzahl quadratischer Reste in primen Restklassen

Sei  $p > 2$  prim. Dann gilt  $|QR_p| = \frac{|\mathbb{Z}_p^*|}{2} = \frac{p-1}{2}$ .

### Beweisidee:

- Quadrieren auf  $\mathbb{Z}_p^*$ ,  $x \mapsto x^2$ , ist eine 2:1-Abbildung.
- Die verschiedenen Werte  $x, (-x)$  werden beide auf  $x^2$  abgebildet.

# Legendre-Symbol

## Definition Legendre Symbol

Sei  $p > 2$  prim und  $a \in \mathbb{N}$ . Das *Legendre Symbol* ist definiert als

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{falls } p|a \\ 1 & \text{falls } (a \bmod p) \in QR_p \\ -1 & \text{falls } (a \bmod p) \in QNR_p \end{cases} .$$

## Satz

$$\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \bmod p$$

## Eigenschaften Quadratischer Reste

- 1 Multiplikativität:  $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right) \left(\frac{b}{p}\right)$
- 2  $(QR_p, \cdot)$  ist eine multiplikative Gruppe.

# Das Jacobi Symbol

## Definition Jacobi Symbol

Sei  $N = p_1^{e_1} \cdot \dots \cdot p_k^{e_k} \in \mathbb{N}$  ungerade und  $a \in \mathbb{N}$ . Dann ist das *Jacobi Symbol* definiert als

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)^{e_1} \cdot \dots \cdot \left(\frac{a}{p_k}\right)^{e_k}.$$

- **Warnung:**  $\left(\frac{a}{N}\right) = 1$  impliziert nicht, dass  $a \in QR_N$  ist.
- Bsp:  $\left(\frac{2}{15}\right) = \left(\frac{2}{3}\right) \cdot \left(\frac{2}{5}\right) = (-1)(-1) = 1$ .
- D.h.  $2 \in QNR_3$  und  $2 \in QNR_5$ . Damit besitzt  $x^2 = 2$  weder Lösungen modulo 3 noch modulo 5.
- Nach CRT besitzt  $x^2 = 2 \pmod{15}$  ebenfalls keine Lösung.

# Pseudoquadrate

**Berechnung des Jacobi-Symbols:** Sei  $a \in \mathbb{Z}_N$ .

- Berechnung von  $\left(\frac{a}{N}\right)$  ist in Zeit  $\log^2(\max\{N, a\})$  möglich, **ohne** die Faktorisierung von  $N$  zu kennen.
- Algorithmus ist ähnlich zum Euklidischen Algorithmus, verwendet das Gaußsche Reziprozitätsgesetz.

## Definition Pseudoquadrat

Sei  $N \in \mathbb{N}$ . Die Menge der *Pseudoquadrate* ist definiert als

$$QNR_N^{+1} = \{a \in \mathbb{Z}_N^* \mid \left(\frac{a}{N}\right) = 1 \text{ und } a \notin QR_N\}.$$

# Multiplikation von Resten/Nichtresten

## Lemma Multiplikation von Resten/Nichtresten

Sei  $N = pq$  ein RSA-Modul. Seien  $x, x' \in QR_N$  und  $y, y' \in QNR_N^{+1}$ .

- 1  $xx' \in QR_N$
- 2  $yy' \in QR_N$
- 3  $xy \in QNR_N^{+1}$

**Beweis:** für 3 (1+2 folgen analog)

- Nach Chinesischem Restsatz gilt

$$QR_N \simeq QR_p \times QR_q \text{ und } QNR_N^{+1} \simeq QNR_p \times QNR_q.$$

- Aus der Multiplikativität des Legendre-Symbols folgt

$$\left(\frac{xy}{N}\right) = \left(\frac{xy}{p}\right) \left(\frac{xy}{q}\right) = \left(\frac{x}{p}\right) \left(\frac{x}{q}\right) \left(\frac{y}{p}\right) \left(\frac{y}{q}\right) = 1 \cdot 1 \cdot (-1) \cdot (-1) = 1.$$

- Analog gilt

$$\left(\frac{xy}{p}\right) = \left(\frac{x}{p}\right) \left(\frac{y}{p}\right) = (-1).$$

- Daraus folgt  $xy \in QNR_N^{+1}$ .

# Quadratische Residuositätsannahme

## Definition Quadratische Residuosität

Das Unterscheiden quadratischer Reste ist hart bezüglich  $\text{GenModulus}(1^n)$  falls für alle ppt  $\mathcal{D}$  gilt

$$|\text{Ws}[\mathcal{D}(1^n, N, qr) = 1] - \text{Ws}[\mathcal{D}(1^n, N, qnr) = 1]| \leq \text{negl}(n),$$

wobei  $qr \in_R QR_N$  und  $qnr \in_R QNR_N^{+1}$ .

**QR-Annahme:** Unterscheiden quadratischer Reste ist hart.

## Idee des Goldwasser-Micali Kryptosystems

- $pk = N, sk = (p, q)$
- Verschlüsselung von 0 ist zufälliges  $x' \in_R QR_N$ .
- Wähle  $x \in_R \mathbb{Z}_N^*$  und berechne  $x' \leftarrow x^2 \bmod N$ .
- Verschlüsselung von 1 ist zufälliges  $y \in_R QNR_N^{+1}$ .
- **Problem:** Wie wählt man  $y$  ohne  $p, q$  zu kennen?
- Abhilfe: Public-Key enthält  $z \in_R QNR_N^{+1}$ .
- Sender wählt  $x \in_R \mathbb{Z}_N^*$  und berechnet  $y \leftarrow z \cdot x^2 \bmod N \in QNR_N^{+1}$ .



# GOLDWASSER-MICALI Verschlüsselung (1984)

## Definition GOLDWASSER-MICALI Verschlüsselung

Sei  $n$  ein Sicherheitsparameter.

- 1 **Gen:**  $(N, p, q) \leftarrow \text{GenModulus}(1^n)$ . Wähle  $z \in_R \text{QNR}_N^{+1}$ . (Wie?)  
Schlüssel:  $pk = (N, z)$  und  $sk = (p, q)$
- 2 **Enc:** Für  $m \in \{0, 1\}$  wähle  $x \in_R \{0, 1\}$  und berechne  
$$c \leftarrow z^m \cdot x^2 \pmod{N}.$$
- 3 **Dec:** Berechne  $m = \begin{cases} 0 & \text{falls } \left(\frac{c}{p}\right) = 1 \\ 1 & \text{sonst} \end{cases}.$

## Korrektheit:

- Für  $m = 0$  ist  $c \in \text{QR}_N \simeq \text{QR}_p \times \text{QR}_q$ , d.h.  $\left(\frac{c}{p}\right) = 1$ .
- Für  $m = 1$  ist  $c \in \text{QNR}_N^{+1} \simeq \text{QNR}_p \times \text{QNR}_q$ , d.h.  $\left(\frac{c}{p}\right) = (-1)$ .

# Sicherheit von GOLDWASSER-MICALI Verschlüsselung

## Satz Sicherheit von GOLDWASSER-MICALI

Unter der QR-Annahme ist GOLDWASSER-MICALI  $\Pi$  CPA-sicher.

### Beweis:

- Sei  $\mathcal{A}$  ein Angreifer für  $\Pi$  mit  $\epsilon(n) = \text{Ws}[PubK_{\mathcal{A}, \Pi}^{cpa}(n) = 1]$ .
- Konstruieren Unterscheider  $\mathcal{D}$  für Quadratische Residuosität.

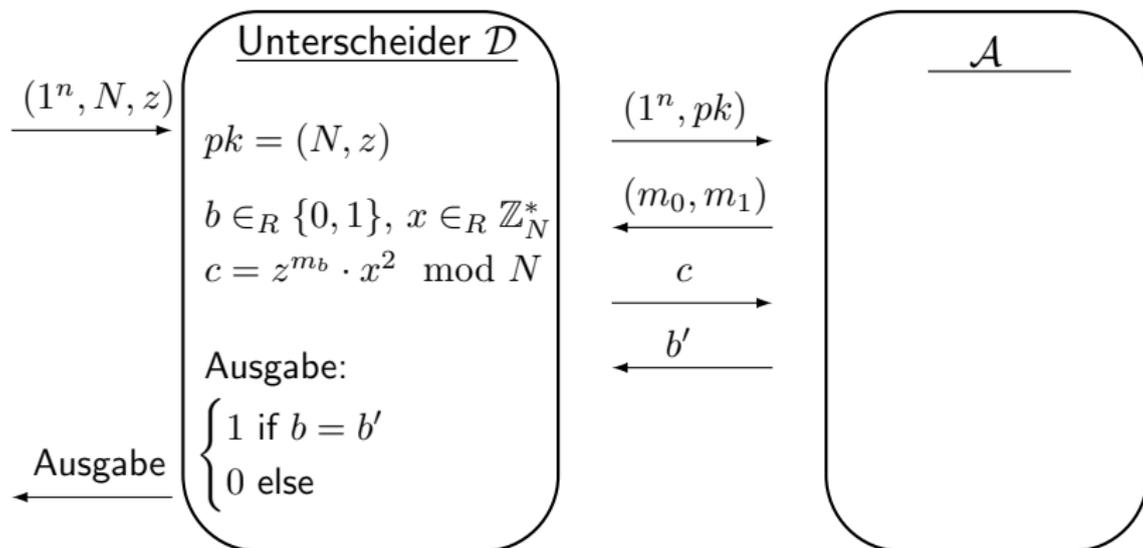
### Algorithmus QR-Unterscheider $\mathcal{D}$

EINGABE:  $(N, z)$  mit  $\left(\frac{z}{N}\right) = 1$

- 1 Setze  $pk = (N, z)$  und berechne  $(m_0, m_1) \leftarrow \mathcal{A}(pk)$ .  
OBdA gilt  $\{m_0, m_1\} = \{0, 1\}$ .
- 2 Wähle  $b \in_R \{0, 1\}$  und  $x \in_R \mathbb{Z}_N^*$ . Berechne  $c \leftarrow z^{m_b} \cdot x^2 \pmod N$ .
- 3  $b' \leftarrow \mathcal{A}(c)$

AUSGABE:  $\begin{cases} 1 & \text{falls } b = b', \text{ Interpretation } z \in QNR_N^{+1} \\ 0 & \text{sonst, Interpretation } z \in QR_N \end{cases}$

# Algorithmus QR-Unterscheider



# Sicherheit von GOLDWASSER-MICALI Verschlüsselung

**Fall 1:**  $z \in QNR_N^{+1}$

- Verteilung von  $c$  ist identisch zu GOLDWASSER-MICALI.
- D.h.  $\text{Ws}[D(1^n, N, qnr) = 1] = \epsilon(n)$ .

**Fall 2:**  $z \in QR_N$

- Falls 0 verschlüsselt wird, gilt  $c = x^2 \in_R QR_N$ .
- Falls 1 verschlüsselt wird, gilt  $c = z \cdot x^2 \in_R QR_N$ .
- D.h. die Verteilung von  $c$  ist unabhängig von der Wahl von  $b$ .
- Sei  $\Pi'$  GOLDWASSER-MICALI Verschlüsselung mit  $z \in QR_N$ .
- Dann gilt  $\text{Ws}[D(1^n, N, qr) = 1] = \text{Ws}[PubK_{\mathcal{A}, \Pi'}^{cpa}(n) = 1] = \frac{1}{2}$ .

Unter der Quadratischen Residuositäts-Annahme folgt

$$\text{negl}(n) \geq |\text{Ws}[D(N, qr) = 1] - \text{Ws}[D(N, qnr) = 1]| = \left| \frac{1}{2} - \epsilon(n) \right|.$$

Damit gilt  $\epsilon(n) \leq \frac{1}{2} + \text{negl}(n)$ .

# Rabin Verschlüsselung 1979

## Idee: Rabin Verschlüsselung

- Beobachtung: Berechnen von Wurzeln in  $\mathbb{Z}_p$  ist effizient möglich.
- Ziehen von Quadratwurzeln in  $\mathbb{Z}_N$  ist äquivalent zum Faktorisieren.

**Vorteil:** CPA-Sicherheit beruht nur auf Faktorisierungsannahme.

Für unsere bisherigen Verfahren verwendeten wir *stärkere* Annahmen.

- RSA: Berechnen von  $e$ -ten Wurzeln in  $\mathbb{Z}_n$ .
- Goldwasser-Micali: Unterscheiden von  $QR_N$  und  $QNR_N^{+1}$ .

# Berechnen von Quadratwurzeln in $\mathbb{Z}_p$

## Satz Ziehen von Wurzeln in $\mathbb{Z}_p$

Sei  $p$  prim mit  $p = 3 \pmod 4$  und  $a \in QR_p$ . Dann gilt für  $b^2 = a \pmod p$ , dass  $b = \pm a^{\frac{p+1}{4}} \pmod p$ . Ferner ist  $a^{\frac{p+1}{4}} \in QR_p$  und  $-a^{\frac{p+1}{4}} \in QNR_p$

### Beweis:

- Es gilt  $(\pm a^{\frac{p+1}{4}})^2 = a^{\frac{p+1}{2}} = a^{\frac{p-1}{2}} \cdot a = \left(\frac{a}{p}\right) \cdot a = a \pmod p$ .

- Wir schreiben  $p = 4k + 3$ . Dann gilt

$$\left(\frac{-a^{\frac{p+1}{4}}}{p}\right) = \left(\frac{-1}{p}\right) \cdot \left(\frac{a^{\frac{p+1}{4}}}{p}\right) = (-1)^{\frac{p-1}{2}} \cdot \left(\frac{a}{p}\right)^{\frac{p+1}{4}} = (-1)^{2k+1} = (-1).$$

- Damit folgt  $-a^{\frac{p+1}{4}} \in QNR_p$  und  $a^{\frac{p+1}{4}} \in QR_p$ .

# Quadratwurzel bei bekannter Faktorisierung

## Definition Blum-Zahl

Sei  $N = pq$  ein RSA-Modul.  $N$  heißt *Blum-Zahl* falls  $p = q = 3 \pmod{4}$ .

## Satz Quadratwurzeln in $\mathbb{Z}_N$

Sei  $N = pq$  eine Blum-Zahl mit bekannten  $p, q$ . Dann können die vier Quadratwurzeln von  $a \in QR_N$  in Zeit  $\mathcal{O}(\log^3 N)$  berechnet werden.

**Beweis:**

## Algorithmus QUADRATWURZEL

EINGABE:  $N, p, q, a \in QR_N$

① Berechne  $x_p \leftarrow a^{\frac{p+1}{4}} \pmod{p}$ ,  $x_q \leftarrow a^{\frac{q+1}{4}} \pmod{q}$ .

② Berechne mittels Chinesischem Restsatz die Lösungen von

$$\left| \begin{array}{l} b_1 = x_p \pmod{p} \\ b_1 = x_q \pmod{q} \end{array} \right|, \left| \begin{array}{l} b_2 = -x_p \pmod{p} \\ b_2 = x_q \pmod{q} \end{array} \right|, \left| \begin{array}{l} b_3 = x_p \pmod{p} \\ b_3 = -x_q \pmod{q} \end{array} \right|, \left| \begin{array}{l} b_4 = -x_p \pmod{p} \\ b_4 = -x_q \pmod{q} \end{array} \right|$$

AUSGABE:  $b_1, \dots, b_4$  mit  $b_i^2 = a \pmod{N}$

## Satz Existenz einer Hauptwurzel

Sei  $N = pq$  eine Blumzahl. Dann besitzt jedes  $a \in QR_N$  genau ein  $b \in QR_N$  mit  $b^2 = a \pmod N$ . Wir bezeichnen  $b$  als *Hauptwurzel*.

### Beweis:

- Algorithmus QUADRATWURZEL liefert

$$b_1 \simeq (x_p, x_q) \in QR_p \times QR_q.$$

- Damit ist  $b_1 \in QR_N$  eine Hauptwurzel.
- Alle anderen Wurzel  $b_2, b_3, b_4$  sind keine Hauptwurzeln.
- Z.B. gilt  $b_2 = (-x_p, x_q) \in QNR_p \times QR_q$  und damit  $b_2 \in QNR_N$ .

# Quadratwurzeln ohne Faktorisierung

## Spiel Wurzelziehen $SQR_{\mathcal{A}, GenModulus}(n)$

1  $(N, p, q) \leftarrow GenModulus(1^n)$

2 Wähle  $z \in_R QR_N$ .

3  $y \leftarrow \mathcal{A}(1^n, N, z)$

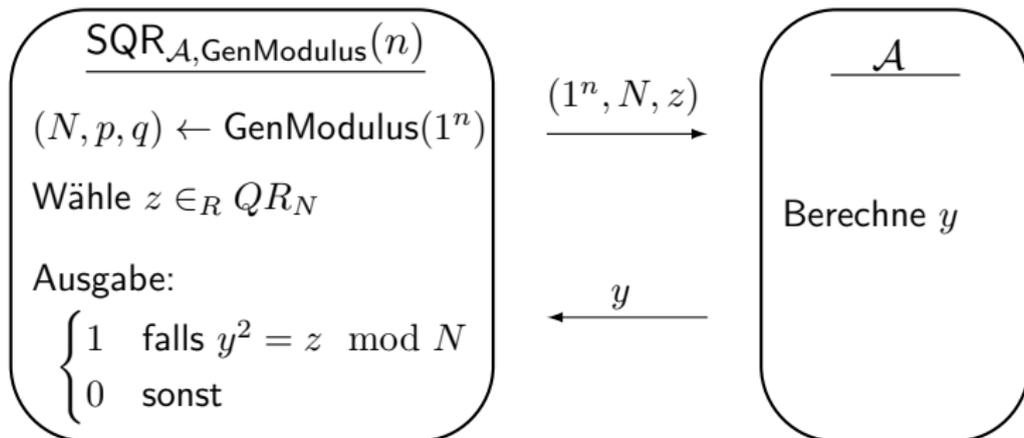
4 
$$SQR_{\mathcal{A}, GenModulus}(n) = \begin{cases} 1 & \text{falls } y^2 = z \pmod{N} \\ 0 & \text{sonst} \end{cases} .$$

## Definition Quadratwurzelannahme

Das Berechnen von Quadratwurzeln ist hart bezüglich  $GenModulus$ , falls für alle ppt  $\mathcal{A}$  gilt  $Ws[SRQ_{\mathcal{A}, GenModulus}(n) = 1] \leq \text{negl}(n)$ .

*Quadratwurzelannahme:* Berechnen von Quadratwurzeln ist hart.

# Spiel Wurzelziehen



# Nicht-triviale Quadratwurzeln

## Satz Faktorisieren mit Wurzeln

Sei  $N = pq$  ein RSA-Modul. Seien  $x, y \in \mathbb{Z}_N^*$  mit  $x^2 = y^2 \pmod{N}$  und  $x \not\equiv \pm y \pmod{N}$ . Dann können  $p, q$  in Zeit  $\mathcal{O}(\log^2 N)$  berechnet werden.

### Beweis:

- Mittels CRT erhalten wir  $x \simeq (x_p, x_q) \in \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ .
- Es gilt  $y \simeq (x_p, -x_q)$  oder  $y \simeq (-x_p, x_q)$ .
- Wir betrachten den Fall  $y \simeq (x_p, -x_q)$ . Der zweite Fall ist analog.
- Es gilt  $x + y \simeq (2x_p, 0)$  bzw.  $x - y \simeq (0, 2x_q)$ .
- Damit folgt  $\text{ggT}(N, x + y) = q$  bzw.  $\text{ggT}(N, x - y) = p$  wegen  $2x_p \in \mathbb{Z}_p^*$  und  $2x_q \in \mathbb{Z}_q^*$ .
- Die ggT-Berechnung benötigt Laufzeit  $\mathcal{O}(\log^2 N)$ .

# Quadratwurzeln implizieren Faktorisierung

## Satz Quadratwurzeln implizieren Faktorisierung

Quadratwurzel- und Faktorisierungsannahme sind äquivalent.

### Beweis:

- Bereits gezeigt: Faktorisierung impliziert Quadratwurzeln.
- Sei  $\mathcal{A}$  Angreifer im Spiel  $SQR_{\mathcal{A}, GenModulus}(n)$  mit Erfolgsws  $\epsilon(n)$ .
- Konstruieren einen Angreifer  $\mathcal{A}'$  im Spiel  $Factor_{\mathcal{A}', GenModulus}(n)$ .

### Algorithmus $\mathcal{A}'$

EINGABE:  $N$

- 1 Wähle  $x \in \mathbb{Z}_N^*$  und berechne  $z \leftarrow x^2 \bmod N$ .
- 2  $y \leftarrow \mathcal{A}(1^n, N, z)$
- 3 Falls  $x = \pm y \bmod N$  oder  $x^2 \neq y^2 \bmod N$ , Abbruch.

AUSGABE:  $p, q = \{\text{ggT}(N, x + y), \text{ggT}(N, x - y)\}$

# Faktorisieren mit Quadratwurzeln

- Unter der Faktorisierungsannahme gilt

$$\begin{aligned} \text{negl}(n) &\geq \text{Ws}[\text{Factor}_{\mathcal{A}', \text{GenModulus}}(n) = 1] \\ &= \text{Ws}[x \neq \pm y \bmod N \wedge x^2 = y^2 \bmod N] \\ &= \text{Ws}[x \neq \pm y \bmod N \mid x^2 = y^2 \bmod N] \cdot \text{Ws}[x^2 = y^2 \bmod N] \\ &= \text{Ws}[x \neq \pm y \bmod N \mid x^2 = y^2 \bmod N] \cdot \epsilon(n) \\ &= \frac{1}{2} \cdot \epsilon(n) \end{aligned}$$

- Die letzte Gleichung folgt, da  $x^2$  exakt vier Wurzeln in  $\mathbb{Z}_N^*$  besitzt.

# Einwegfunktion unter Quadratwurzelannahme

## Definition Einwegfunktion QUADRAT

Definieren *Einwegfunktionfamilie* QUADRAT = (Gen, Samp, f) als

- 1 **Gen**( $1^n$ ) :  $(N, p, q) \leftarrow \text{GenModulus}(1^n)$ , Ausgabe  $I = N$ .  
Definiert  $f : \mathbb{Z}_N^* \rightarrow QR_N$ .
- 2 **Samp**( $I$ ) : Wähle  $x \in_R \mathbb{Z}_N^*$  zufällig.
- 3 **f<sub>I</sub>**( $x$ ) : Berechne  $f(x) := x^2 \pmod N$ .

## Anmerkungen:

- QUADRAT ist Einwegfunktion unter der Quadratwurzelannahme.
- D.h. QUADRAT ist Einwegfunktion unter Faktorisierungsannahme.
- **Ziel:** Konstruktion einer Trapdoor-Einwegpermutation.

# RABIN Trapdoor-Einwegpermutation

## Definition RABIN Trapdoor-Einwegpermutation

Trapdoor-Einwegpermutationsfamilie  $RABIN = (Gen, Samp, f)$  mit

- 1 **Gen**( $1^n$ ) :  $(N, p, q) \leftarrow GenModulus(1^n)$  mit Blumzahl  $N$ .  
Ausgabe  $I = N$ ,  $td = (p, q)$ . Definiert  $f : QR_N \rightarrow QR_N$ .
- 2 **Samp**( $I$ ) : Wähle  $r \in_R \mathbb{Z}_N^*$  zufällig. Berechne  $x \leftarrow r^2 \bmod N$ .
- 3 **f<sub>I</sub>**( $x$ ) : Berechne  $y = f(x) := x^2 \bmod N$ .
- 4 **Inv<sub>td</sub>**( $y$ ) : Bestimme Hauptwurzel  $x \in QR_N$  von  $y = x^2$ .

## Anmerkungen:

- RABIN ist einweg unter der Faktorisierungsannahme. Wir wissen:  
Trapdoor-Einwegpermutation + Hardcore-Prädikat  
= CPA-sichere Verschlüsselung.
- Benötigen ein Hardcore-Prädikat  $hc : QR_N \rightarrow \{0, 1\}$  für  $f$ .

# Berechnen des niederwertigsten Bits

## Satz Hardcore-Prädikat $lsb(x)$

Sei  $f$  die RABIN Trapdoor-Einwegpermutation und  $N$  eine Blumzahl. Für  $x \in QR_N$  bezeichne  $lsb(x)$  das niederwertigste Bit von  $x$ . Dann ist  $hc(x) := lsb(x)$  ein Hardcore-Prädikat für  $f$ .

- ohne Beweis (nicht-trivial)
- Für alle ppt  $\mathcal{A}$  gilt damit  $W_S[\mathcal{A}(1^n, N, x^2) = lsb(x)] \leq \frac{1}{2} + \text{negl}(n)$ .

# RABIN Kryptosystem (1979)

## Algorithmus RABIN Verschlüsselung

- 1 **Gen:**  $(N, p, q) \leftarrow \text{GenModulus}(1^n)$ , wobei  $N$  eine Blumzahl ist.  
Ausgabe  $pk = N, sk = (p, q)$ .
- 2 **Enc:** Für  $m \in \{0, 1\}$  wähle  $r \in \mathbb{Z}_N^*$ , berechne  $x \leftarrow r^2 \bmod N$  und  
 $c \leftarrow (x^2 \bmod N, \text{lsb}(x) \oplus m)$ .
- 3 **Dec:** Für  $c = (c_1, c_2)$  berechne Hauptwurzel  $x$  von  $c_1$  und  
 $m := \text{lsb}(x) \oplus c_2$ .

## Satz CPA-Sicherheit von RABIN

RABIN Verschlüsselung ist CPA-sicher unter der Faktorisierungsannahme.

### Beweis:

- Folgt aus dem Satz zur CPA-Sicherheit von  $\text{VERSCHLÜSSELUNG}_{\Pi}$ .

# Diskussion RSA versus RABIN

## Vergleich: RSA und RABIN

- Quadrieren und Exponentieren mit  $e$  erscheinen ähnlich.
- Aber: RABIN ist kein Spezialfall von RSA, da  $e = 2 \notin \mathbb{Z}_{\phi(N)}^*$ .
- RABIN-Einwegpermutation beruht auf Faktorisierungsannahme.
- Die Faktorisierungsannahme ist möglicherweise schwächer als die RSA-Annahme. Offen: Invertieren von RSA  $\Rightarrow$  Faktorisierung?
- RABIN ist nicht ineffizienter als RSA.
  
- Historische Variante: Textbook RABIN mit  $c := m^2 \bmod N$ .
- Es existiert ein CCA-Angriff auf Textbook RABIN, der  $p, q$  liefert. (Wie?)

# Die Gruppe $\mathbb{Z}_{N^2}^*$

## Lemma Teilerfremdheit von $N$ und $\phi(N)$

Sei  $N = pq$  ein RSA-Modulus mit  $p, q$  gleicher Bitlänge. Dann gilt  $\text{ggT}(N, \phi(N)) = 1$ .

### Beweis:

- OBdA  $p > q$ . Dann kann  $p$  weder  $(p - 1)$  noch  $(q - 1)$  teilen.
- Annahme:  $q$  teilt  $p - 1$ . Dann ist  $\frac{p-1}{q} \geq 2$ .
- Widerspruch:  $\frac{p}{q} < 2$ , da  $p, q$  gleiche Bitlänge besitzen.

## Lemma Ordnung von $(1 + N) \bmod N^2$

Sei  $N$  ein RSA-Modul. Dann besitzt  $(1 + N)$  in  $\mathbb{Z}_{N^2}^*$  Ordnung  $N$ .

### Beweis:

- Es gilt  $(1 + N)^a = \sum_{i=0}^a \binom{a}{i} N^i = 1 + aN \bmod N^2$ .
- D.h.  $(1 + N)^a \neq 1 \bmod N^2$  für  $1 \leq a < N$  und  $(1 + N)^N = 1 \bmod N^2$ .

# Die Struktur von $\mathbb{Z}_{N^2}^*$

## Satz Isomorphismus $\mathbb{Z}_N \times \mathbb{Z}_N^* \simeq \mathbb{Z}_{N^2}^*$

Die Abbildung  $f : \mathbb{Z}_N \times \mathbb{Z}_N^* \rightarrow \mathbb{Z}_{N^2}^*$  mit  $f(a, b) = (1 + N)^a \cdot b^N \pmod{N^2}$  ist ein Isomorphismus, d.h.

- 1  $f$  ist bijektiv.
- 2  $f(a_1, b_1) \cdot f(a_2, b_2) = f(a_1 + a_2, b_1 b_2) \quad \forall a_1, a_2 \in \mathbb{Z}_N, b_1, b_2 \in \mathbb{Z}_N^*$ .

### Beweis: Bijektivität

- Zeigen, dass  $|\mathbb{Z}_N \times \mathbb{Z}_N^*| = |\mathbb{Z}_{N^2}^*|$  und dass  $f$  injektiv ist.
- $|\mathbb{Z}_{N^2}^*| = \phi(N^2) = (p^2 - p)(q^2 - q) = pq(p - 1)(q - 1) = |\mathbb{Z}_N| \cdot |\mathbb{Z}_N^*|$
- **Annahme:**  $\exists (a_1, b_1) \neq (a_2, b_2)$  mit  $f(a_1, b_1) = f(a_2, b_2)$ .
- Dann folgt  $(1 + N)^{a_1} b_1^N = (1 + N)^{a_2} b_2^N \pmod{N^2}$ .
- Wegen  $|\mathbb{Z}_{N^2}^*| = N \cdot \phi(N)$  liefert Potenzieren mit  $\phi(N)$   
$$(1 + N)^{(a_1 - a_2)\phi(N)} = 1 \pmod{N^2}.$$
- Es gilt  $\text{ord}(1 + N) = N$  und daher  $N \mid (a_1 - a_2)\phi(N)$ .
- Wegen  $\text{ggT}(N, \phi(N)) = 1$  folgt  $N \mid a_1 - a_2$ , d.h.  $a_1 = a_2 \pmod{N}$ .

## Beweis: Fortsetzung Bijektivität

- $a_1 = a_2$  liefert  $b_1^N = b_2^N \pmod{N^2}$  und damit  $b_1^N = b_2^N \pmod{N}$ .
- Wegen  $\text{ggT}(N, \phi(N))$  ist die Exponentiation mit  $N$  bijektiv.
- Daraus folgt  $b_1 = b_2 \pmod{N}$ . (Widerspruch:  $(a_1, b_1) \neq (a_2, b_2)$ )

## Beweis: Homomorphismus-Eigenschaft

- Es gilt  $f(a_1, b_1) \cdot f(a_2, b_2) = (1 + N)^{a_1+a_2} \cdot (b_1 b_2)^N \pmod{N^2}$ .
- Wegen  $\text{ord}(1 + N) = N$  entspricht dies  $(1 + N)^{a_1+a_2 \pmod{N}} \cdot (b_1 b_2)^N$ .
- Es gilt  
$$f(a_1 + a_2, b_1 b_2) = (1 + N)^{a_1+a_2 \pmod{N}} \cdot (b_1 b_2 \pmod{N})^N \pmod{N^2}.$$
- Sei  $r = b_1 b_2 \pmod{N}$ . D.h.  $b_1 b_2 = r + kN$ .
- Dann gilt  $(b_1 b_2)^N = (r + kN)^N = r^N = (b_1 b_2 \pmod{N})^N \pmod{N^2}$ .  $\square$

# N-te Reste

## Definition N-te Reste

Sei  $N$  ein RSA-Modul. Wir bezeichnen die Elemente der Menge  $\text{Res}(N^2) := \{y \in \mathbb{Z}_{N^2}^* \mid \exists x \in \mathbb{Z}_{N^2}^* \text{ mit } x^N = y\}$  als *N-te Reste* in  $\mathbb{Z}_{N^2}^*$ .

## Lemma Eigenschaften N-ter Reste

- 1 Exponentiation mit  $N$  ist eine  $(N : 1)$ -Abbildung in  $\mathbb{Z}_{N^2}^*$ .
- 2  $\text{Res}(N^2) \simeq \{(0, b) \mid b \in \mathbb{Z}_N^*\}$

## Beweis:

- Sei  $x \in \mathbb{Z}_{N^2}^*$  mit  $x \simeq (a, b)$ . Dann gilt
$$x^N \bmod N^2 \simeq (a, b)^N = (N \cdot a \bmod N, b^N \bmod N) = (0, b^N).$$
- Für die  $N$  Elemente  $(a, b)$ ,  $a \in \mathbb{Z}_N$ , gilt  $(a, b)^N = (0, b^N)$ .
- Damit ist jeder  $N$ -te Rest von der Form  $(0, b^N)$ .
- Bleibt zu zeigen, dass jedes Element  $y \simeq (0, b)$  ein  $N$ -ter Rest ist.
- Falls  $y \simeq (0, b)$  ist, so gilt  $y = (1 + N)^0 \cdot b^N = b^N \bmod N^2$ .
- Damit ist  $y$  ein  $N$ -ter Rest.

# DCR Annahme

## Satz Decisional Composite Residuosity (DCR)

Das *Decisional Composite Residuosity* Problem ist hart bezüglich GenModulus falls für alle ppt  $\mathcal{A}$  und  $r \in_R \mathbb{Z}_{N^2}^*$  gilt

$$|\text{Ws}[\mathcal{A}(1^n, N, r^N \bmod N^2) = 1] - \text{Ws}[\mathcal{A}(1^n, N, r) = 1]| \leq \text{negl}(n).$$

*DCR Annahme*: DCR ist hart bezüglich GenModulus.

- DCR Annahme: Unterscheiden von  $(0, r)$  und  $(r', r)$  ist schwer.

**Idee:** zur Konstruktion einer Verschlüsselungsfunktion

- Sei  $m \in \mathbb{Z}_N$ . Wähle einen zufälligen  $N$ -ten Rest  $(0, r)$  und setze
$$c \leftarrow (m, 1) \cdot (0, r) = (m, r).$$
- Da  $(0, r)$  ununterscheidbar von  $(r', r)$ , ist  $c$  ununterscheidbar von
$$c' \leftarrow (m, 1) \cdot (r', r) = (m + r', r).$$
- $c' = (m + r', r)$  ist für  $r' \in_R \mathbb{Z}_N$  ein zufälliges Element in  $\mathbb{Z}_N \times \mathbb{Z}_N^*$ .
- Insbesondere ist  $c'$  unabhängig von  $m$ .

# Verschlüsselung

## Algorithmus Verschlüsselung

EINGABE:  $m \in \mathbb{Z}_N$

- 1 Wähle  $r \in_R \mathbb{Z}_N^*$ .
- 2 Berechne  $c \leftarrow f(m, r) = (1 + N)^m \cdot r^N \bmod N^2$ .

AUSGABE:  $c \in \mathbb{Z}_{N^2}^*$

### Anmerkungen:

- Wir berechnen das Bild von  $(m, r)$  unter unserem Isomorphismus.
- Faktor der Nachrichtenexpansion beträgt 2.

# Entschlüsselung

## Algorithmus Entschlüsselung

EINGABE:  $c \simeq (m, r) \in \mathbb{Z}_{N^2}^*$

- 1 Berechne  $c' := c^{\phi(N)} \bmod N^2$ .
- 2 Berechne  $m' := \frac{c'-1}{N}$  über  $\mathbb{N}$ .
- 3 Berechne  $m := m' \cdot \phi(N)^{-1} \bmod N$ .

AUSGABE:  $m \in \mathbb{Z}_N$

### Korrektheit:

- Es gilt  $c' \simeq (m, r)^{\phi(N)} = (m\phi(N), r^{\phi(N)}) = (m\phi(N), 1)$ .
- Damit gilt
$$c' = (1 + N)^{m\phi(N)} \bmod N \quad 1^N = 1 + (m\phi(N) \bmod N) \cdot N \bmod N^2.$$
- Da  $1 + (m\phi(N) \bmod N)N < N^2$  gilt die Gleichung über  $\mathbb{N}$ .
- Daraus folgt  $m' = m\phi(N) \bmod N$ . Multiplikation mit  $\phi(N)^{-1}$  liefert

$$m = m' \cdot \phi(N)^{-1} \bmod N.$$

# Paillier Kryptosystem (1999)

## Algorithmus Paillier Verschlüsselung

- 1 **Gen:**  $(N, p, q) \leftarrow \text{GenModulus}(1^n)$ . Ausgabe  $pk = N, sk = \phi(N)$ .
- 2 **Enc:** Für eine Nachricht  $m \in \mathbb{Z}_N$ , wähle ein  $r \in_R \mathbb{Z}_N^*$  und berechne
$$c \leftarrow (1 + N)^m \cdot r^N \bmod N^2.$$
- 3 **Dec:** Für einen Chiffretext  $c \in \mathbb{Z}_{N^2}^*$  berechne

$$m' := \frac{(c^{\phi(N) \bmod N^2}) - 1}{N} \text{ über } \mathbb{N} \quad \text{und} \quad m := m' \cdot \phi(N)^{-1} \bmod N.$$

# Sicherheit von Paillier Verschlüsselung

## Satz Sicherheit von Paillier Verschlüsselung

Unter der DCR Annahme ist Paillier Verschlüsselung  $\Pi_P$  CPA-sicher.

### Beweis:

- Sei  $\mathcal{A}$  ein Angreifer mit Erfolgsws  $\epsilon(n) = \text{Ws}[\text{PubK}_{\mathcal{A}, \Pi_P}^{\text{cpa}}(n) = 1]$ .
- Konstruieren Algorithmus  $\mathcal{A}_{\text{dcr}}$  für das DCR Problem.

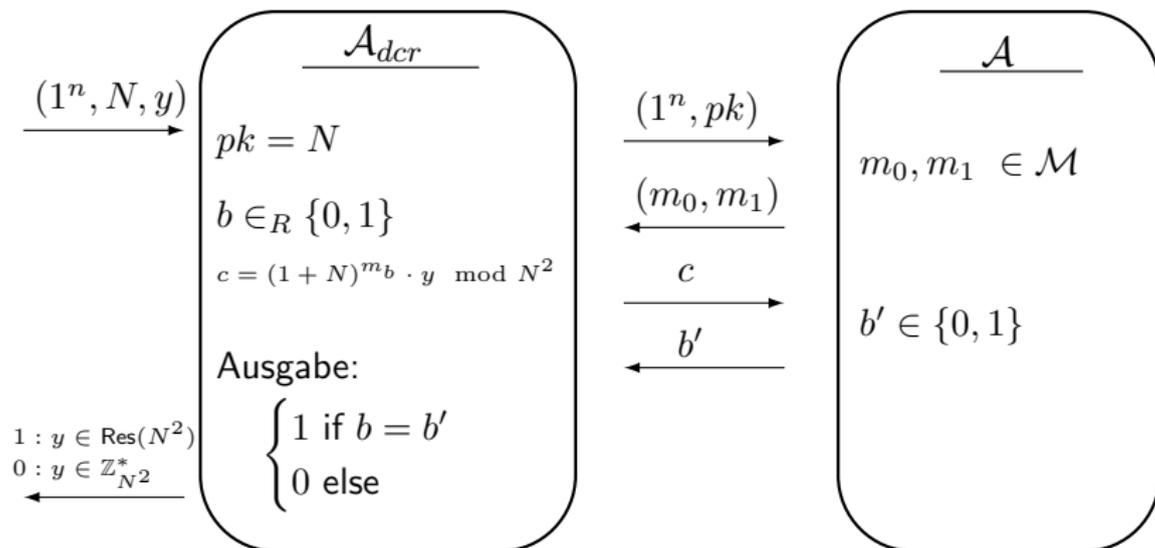
## Algorithmus DCR Unterscheider $\mathcal{A}_{\text{dcr}}$

EINGABE:  $1^n, N, y$

- 1 Setze  $pk = N$  und berechne  $(m_0, m_1) \leftarrow \mathcal{A}(1^n, pk)$ .
- 2 Wähle  $b \in \{0, 1\}$  und berechne  $c \leftarrow (1 + N)^{m_b} \cdot y \bmod N^2$ .
- 3  $b' \leftarrow \mathcal{A}(c)$ .

AUSGABE:  $= \begin{cases} 1 & \text{falls } b = b', \\ 0 & \text{sonst,} \end{cases} \begin{array}{l} \text{Interpretation } y \in \text{Res}(N^2) \\ \text{Interpretation } y \in \mathbb{Z}_{N^2}^* \end{array} .$

# Algorithmus DRC Unterscheider



# Sicherheit von Paillier Verschlüsselung

**Fall 1:**  $y \in_R \text{Res}(N^2)$ , d.h.  $y = r^N$  für  $r \in_R \mathbb{Z}_{N^2}$ .

- Verteilung von  $c$  ist identisch zum Paillier Verfahren.
- D.h.  $\text{Ws}[\mathcal{A}_{dcr}(1^n, N, r^N) = 1] = \epsilon(n)$ .

**Fall 2:**  $y \in_R \mathbb{Z}_{N^2}^*$ , d.h.  $y = r \in_R \mathbb{Z}_{N^2}^*$ .

- Dann ist  $c = (1 + N)^{m_b} \cdot y \bmod N^2$  zufällig in  $\mathbb{Z}_{N^2}^*$ .
- Insbesondere ist die Verteilung von  $c$  unabhängig von  $b$ .
- Daraus folgt  $\text{Ws}[\mathcal{A}_{dcr}(1^n, N, r) = 1] = \frac{1}{2}$ .

Unter der DCR-Annahme folgt

$$\begin{aligned} \text{negl}(n) &\geq \left| \text{Ws}[\mathcal{A}_{dcr}(1^n, N, r^N \bmod N^2) = 1] - \text{Ws}[\mathcal{A}_{dcr}(1^n, N, r) = 1] \right| \\ &= \left| \epsilon(n) - \frac{1}{2} \right|. \end{aligned}$$

Daraus folgt  $\epsilon(n) \leq \frac{1}{2} + \text{negl}(n)$ .

# Homomorphe Verschlüsselung

## Definition Homomorphe Verschlüsselung

Sei  $\Pi$  ein Verschlüsselungsverfahren mit  $Enc : G \rightarrow G'$  für Gruppen  $G, G'$ .  $\Pi$  heißt *homomorph*, falls  $Enc(m_1) \circ_{G'} Enc(m_2)$  eine gültige Verschlüsselung von  $m_1 \circ_G m_2$  für alle  $m_1, m_2 \in G$  ist.

### Bsp:

- **Textbook-RSA** mit  $Enc : (\mathbb{Z}_N^*, \cdot) \rightarrow (\mathbb{Z}_N^*, \cdot)$  und

$$m_1^e \cdot m_2^e = (m_1 \cdot m_2)^e \text{ mod } N.$$

- **ElGamal** mit  $Enc : (\mathbb{Z}_p^*, \cdot) \rightarrow (\mathbb{Z}_p^*, \cdot) \times (\mathbb{Z}_p^*, \cdot)$  und

$$(g^{y_1}, h^{y_1} m_1) \cdot (g^{y_2}, h^{y_2} m_2) = (g^{y_1+y_2}, h^{y_1+y_2} m_1 m_2).$$

Hier ist  $\circ_{G'}$  die komponentenweise Multiplikation in  $\mathbb{Z}_p^* \times \mathbb{Z}_p^*$ .

- **Goldwasser-Micali** mit  $Enc : (\mathbb{Z}_2, +) \rightarrow (\mathbb{Z}_N^*, \cdot)$  und

$$z^{m_1} x_1^2 \cdot z^{m_2} x_2^2 = z^{m_1+m_2 \text{ mod } 2} (x_1 x_2)^2 \text{ mod } N.$$

# E-voting mit Paillier

- **Paillier** mit  $Enc : (\mathbb{Z}_N, +) \rightarrow (\mathbb{Z}_{N^2}^*, \cdot)$  und
$$(1 + N)^{m_1} r_1^N \cdot (1 + N)^{m_2} r_2^N = (1 + N)^{m_1 + m_2 \bmod N} (r_1 r_2)^N \bmod N^2.$$
Vorteil:  $G = (\mathbb{Z}_N, +)$  ist additiv und groß.

## Algorithmus E-voting mit Paillier

- Wahlleiter generiert öffentlichen RSA-Modul  $N = pq$ .
- Wähler  $i \in [n]$  mit  $n < N$  wählt  $v_i = 0$  für NEIN,  $v_i = 1$  für JA und sendet an alle anderen Wähler  $c_i = (1 + N)^{v_i} r_i^N \bmod N^2$ .
- Wähler aggregieren  $c := \prod_{i=1}^n c_i \bmod N^2$ .
- Wahlleiter erhält  $c$  und veröffentlicht  $Dec(c) = \sum_{i=1}^n v_i$ .

**Eigenschaften:** (falls alle Parteien sich an das Protokoll halten)

- Wahlleiter erhält  $c$ , ohne die einzelnen  $c_i$  kennenzulernen.
- Kein Wähler erhält Informationen über die  $v_i$  anderer Wähler.
- Berechnung von  $c$  ist öffentlich verifizierbar.

# Voll homomorphe Verschlüsselung

## Definition Voll homomorphe Verschlüsselung

Sei  $\Pi$  ein Verschlüsselungsverfahren mit  $Enc : R \rightarrow R'$  für Ringe  $R, R'$ .  $\Pi$  heißt *voll homomorph*, falls

- 1  $Enc(m_1) + Enc(m_2)$  eine gültige Verschlüsselung von  $m_1 + m_2$
  - 2  $Enc(m_1) \cdot Enc(m_2)$  eine gültige Verschlüsselung von  $m_1 \cdot m_2$
- für alle  $m_1, m_2 \in R$  ist.

## Anwendung: Cloud Computing

- Sende verschlüsselt Algorithmus  $\mathcal{A}$ , Eingabe  $x$  an einen Server  $S$ .
- $S$  berechnet daraus die verschlüsselte Ausgabe  $Enc(\mathcal{A}(x))$ .
- Erlaubt Auslagern von Berechnungen an  $S$ .
- $S$  lernt nichts über das Programm  $\mathcal{A}$  oder die Eingabe  $x$ .

## Erste voll homomorphe Verschlüsselung:

Gentry Verfahren (2009), basierend auf Problemen der Gittertheorie.

# Digitale Signaturen

**Funktionsweise** von digitalen Signaturen:

- Schlüsselgenerierung erzeugt  $pk, sk$ .
- Signieren ist Funktion von  $sk$ .
- Verifikation ist Funktion von  $pk$ .

**Idee:** Es soll unmöglich sein, ein gültiges Paar von Nachricht  $m$  mit zugehöriger Signatur  $\sigma$  zu erzeugen, ohne  $sk$  zu kennen.

**Eigenschaften** digitaler Signaturen.

- **Integrität:**  $m$  kann nicht verändert werden, da man keine gültige Signatur zu einem  $m' \neq m$  erstellen kann.
- **Authentizität:** Falls  $\sigma$  ein gültige Signatur zu  $m$  ist, so kommt die Signatur vom Besitzer des  $sk$ .
- **Transferierbarkeit:** Jeder kann die Gültigkeit von  $(m, \sigma)$  überprüfen. Insbesondere kann  $(m, \sigma)$  weitergereicht werden.
- **Nicht-Abstreitbarkeit:** Signierer kann nicht behaupten, dass eine andere Person eine gültige Signatur erzeugt hat.

# Definition Signaturverfahren

## Definition Signaturverfahren

Ein *Signaturverfahren* ist ein 3-Tupel  $(Gen, Sign, Vrfy)$  von ppt Alg mit

- 1 **Gen:**  $(pk, sk) \leftarrow Gen(1^n)$ .
- 2 **Sign:** Für eine Nachricht  $m \in \{0, 1\}^*$  berechne

$$\sigma := Sign_{sk}(m).$$

- 3 **Vrfy:** Für ein Tupel  $(m, \sigma)$  berechne

$$Vrfy_{pk}(m, \sigma) = \begin{cases} 1 & \text{falls } \sigma \text{ gültig ist für } m. \\ 0 & \text{sonst} \end{cases}.$$

Für alle  $m \in \{0, 1\}^*$  gilt  $Vrfy_{pk}(m, \sigma) = 1$  gdw  $\sigma = Sign_{sk}(m)$ .

# Unfälschbarkeit von Signaturen

## Spiel CMA-Spiel $Forge_{\mathcal{A},\Pi}(n)$

Sei  $\Pi$  ein Signaturverfahren mit Angreifer  $\mathcal{A}$ .

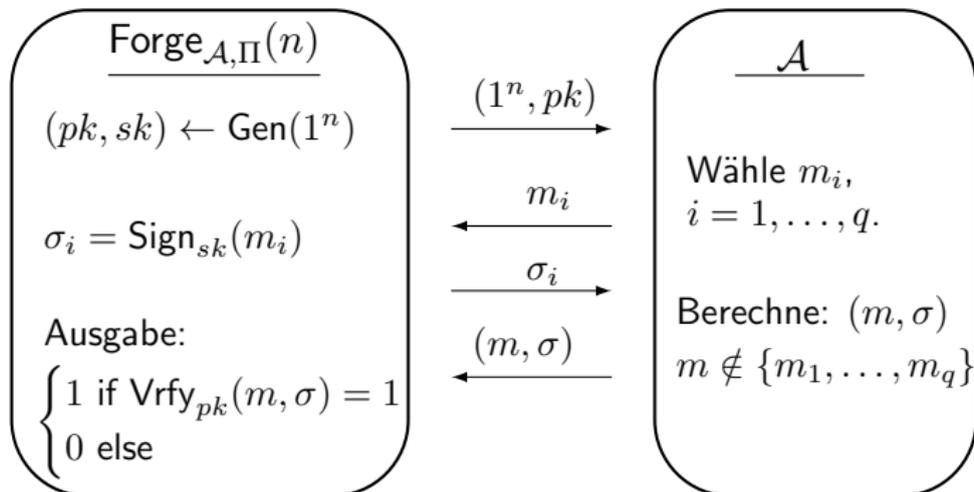
- 1  $(pk, sk) \leftarrow Gen(1^n)$
- 2  $(m, \sigma) \leftarrow \mathcal{A}^{Sign_{sk}(\cdot)}(pk)$ , wobei  $Sign_{sk}(\cdot)$  ein Signierorakel für beliebige Nachrichten  $m' \neq m$  ist.
- 3  $Forge_{\mathcal{A},\Pi}(n) = \begin{cases} 1 & \text{falls } Vrfy_{pk}(m, \sigma) = 1, Sign_{sk}(m) \text{ nicht angefragt} \\ 0 & \text{sonst} \end{cases}$

## Definition CMA-Sicherheit

Sei  $\Pi$  ein Signaturverfahren.  $\Pi$  heißt *existentiell unfälschbar* unter *Chosen Message Angriffen* (oder kurz *CMA-sicher*), falls für alle ppt Angreifer  $\mathcal{A}$  gilt

$$\text{Ws}[Forge_{\mathcal{A},\Pi}(n) = 1] \leq \text{negl}(n).$$

# CMA Spiel Forge



# Unsicherheit von Textbook RSA Signaturen

## Algorithmus Textbook RSA Signaturen

- 1 **Gen:**  $(N, e, d) \leftarrow \text{GenRSA}(1^n)$ . Setze  $pk = (N, e)$ ,  $sk = (N, d)$ .
- 2 **Sign:** Für  $m \in \mathbb{Z}_N$  berechne  $\sigma = m^d \bmod N$ .
- 3 **Vrfy:** Für  $(m, \sigma) \in \mathbb{Z}_N \times \mathbb{Z}_N$  Ausgabe  $\begin{cases} 1 & \text{falls } \sigma^e \stackrel{?}{=} m \bmod N \\ 0 & \text{sonst} \end{cases}$ .

### Unsicherheit: gegenüber CMA-Angriffen

- Wähle beliebiges  $\sigma \in \mathbb{Z}_N$ . Berechne  $m := \sigma^e \bmod N$ .
- Offenbar ist  $\sigma$  eine gültige Signatur für  $m$ .
- Angreifer besitzt keine Kontrolle über  $m$  (existentielle Fälschung).

### Fälschen einer Signatur für ein gewähltes $m \in \mathbb{Z}_N$ :

- Wähle  $m_1 \in_R \mathbb{Z}_N^* \setminus \{1, m\}$ . Berechne  $m_2 = \frac{m}{m_1} \bmod N$ .
- Lasse  $m_1, m_2$  vom Orakel  $\text{Sign}_{sk}(\cdot)$  unterschreiben.
- Seien  $\sigma_1, \sigma_2$  die Signaturen. Dann ist  $\sigma = \sigma_1 \cdot \sigma_2 = m_1^d \cdot m_2^d = (m_1 m_2)^d = m^d \bmod N$  gültig für  $m$ .

# Hashfunktionen und Kollisionen

## Definition Hashfunktion

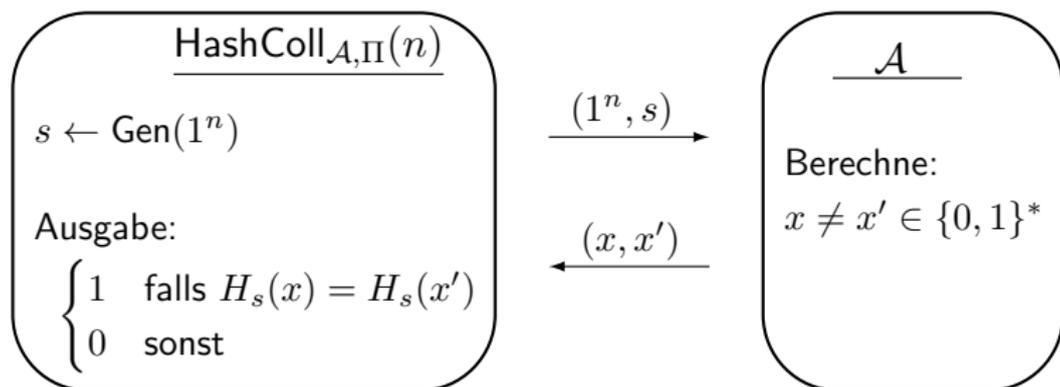
Eine *Hashfunktion* ist ein Paar  $(Gen, H)$  von pt Algorithmen mit

- 1 **Gen:**  $s \leftarrow Gen(1^n)$ .  $Gen$  ist probabilistisch.
- 2 **H:** Für einen Index  $s$  und ein Argument  $x \in \{0, 1\}^*$  berechne  $H_s(x)$ , wobei  $H_s : \{0, 1\}^* \rightarrow \{0, 1\}^n, x \mapsto H_s(x)$ .

## Spiel $HashColl_{\mathcal{A}, \Pi}(n)$

- 1  $s \leftarrow Gen(1^n)$
- 2  $(x, x') \leftarrow \mathcal{A}(s)$
- 3  $HashColl_{\mathcal{A}, \Pi} = \begin{cases} 1 & \text{falls } H_s(x) = H_s(x') \text{ und } x \neq x' \\ 0 & \text{sonst} \end{cases}$ .

# Kollisionsresistente Hashfunktionen



## Definition Kollisionsresistenz

Eine Hashfunktion  $\Pi$  heißt *kollisionsresistent*, falls für alle ppt  $\mathcal{A}$  gilt  $\mathbb{W}_s[\text{HashColl}_{\mathcal{A}, \Pi}(n) = 1] \leq \text{negl}(n)$ .

# Hashed RSA

## Algorithmus Hashed RSA

- 1 Gen:**  $(N, e, d, H) \leftarrow \text{GenHashRSA}(1^n)$  mit  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_N$ .  
Ausgabe  $pk = (N, e, H)$ ,  $sk = (N, d, H)$ .
- 2 Sign:** Für  $m \in \{0, 1\}^*$  berechne  $\sigma = H(m)^d \bmod N$ .
- 3 Vrfy:** Für  $(m, \sigma) \in \mathbb{Z}_N^2$  Ausgabe 1 gdw  $\sigma^e \stackrel{?}{=} H(m) \bmod N$ .

### Einfacher Angriff:

- Sei  $m_1 \neq m_2$  eine Kollision für  $H$  ist, d.h.  $H(m_1) = H(m_2)$ .
- Frage  $(m_1, \sigma)$  an. Dann ist  $(m_2, \sigma)$  eine gültige Fälschung.
- D.h. wir benötigen für  $H$  Kollisionsresistenz.

### Anmerkung: Sicherheit gegen unsere Angriffe für Textbook RSA

- 1** Wähle  $\sigma \in \mathbb{Z}_N$ , berechne  $\sigma^e$ . Müssen  $m \in H^{-1}(\sigma^e)$  bestimmen.  
Übung: Urbildbestimmung ist schwer für kollisionsresistentes  $H$ .
- 2** Für ein  $m \in \mathbb{Z}_N^*$  benötigen wir  $m_1, m_2$  mit  $H(m) = H(m_1) \cdot H(m_2)$  in  $\mathbb{Z}_N$ . Scheint Invertierbarkeit von  $H$  zu erfordern.

# RSA Full Domain Hash (RSA-FDH) Signaturen

## Signatur RSA-FDH

Sei  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_N^*$  ein Random-Oracle.

① **Gen:**  $(N, e, d) \leftarrow \text{GenRSA}(1^n)$  mit  $pk = (N, e)$  und  $sk = (N, d)$ .

② **Sign:** Für eine Nachricht  $m \in \{0, 1\}^*$  berechne

$$\sigma \leftarrow H(m)^d \bmod N.$$

③ **Vrfy:** Für  $(m, \sigma)$  überprüfe

$$\sigma^e \stackrel{?}{=} H(m) \bmod N.$$

## Anmerkung:

- RSA-FDH entspricht Hashed-RSA mit einem Random Oracle als Hashfunktion.

# CMA-Sicherheit von RSA-FDH

## Satz CMA-Sicherheit von RSA-FDH

Unter der RSA-Annahme und für ein Random-Oracle  $H$  ist RSA-FDH ein CMA-sicheres Signaturverfahren.

### Beweisskizze:

- Sei  $\Pi = \text{RSA-FDH}$  und  $\epsilon = \text{Ws}[\text{Forge}_{\mathcal{A},\Pi}(n) = 1]$ .
- OBdA gelten folgende Annahmen für die Orakelanfragen von  $\mathcal{A}$ :
  - 1  $\mathcal{A}$  fragt verschiedene  $x_1, \dots, x_q$  an  $H(\cdot)$ .
  - 2 Bevor  $\mathcal{A}$  Anfrage  $m$  an  $\text{Sign}_{sk}(\cdot)$  stellt, fragt er  $H(m)$  an.
  - 3 Für eine Fälschung  $(m, \sigma)$  hat  $\mathcal{A}$  zuvor Anfrage  $H(m)$  gestellt.
- Konstruieren RSA-Invertierer  $\mathcal{A}'$  mittels  $\mathcal{A}$ .

# Beweis der CMA-Sicherheit von RSA-FDH

## Algorithmus RSA-Invertierer $\mathcal{A}'$

EINGABE:  $N, e, y = x^e \bmod N$

1 Wähle  $j \in_R \{1, \dots, q\}$ .

2  $(m, \sigma) \leftarrow \mathcal{A}^{\text{Sign}_{sk}(\cdot)}(N, e)$ .

▶ Beantworte Orakelanfragen  $m_i$  an  $H(\cdot)$  konsistent mit

$$H(m_i) = \begin{cases} \sigma_i^e \bmod N & \text{für ein selbst gewähltes } \sigma \in_R \mathbb{Z}_N^* \text{ für } i \neq j \\ y & \text{sonst} \end{cases}$$

▶ Beantworte Orakelanfragen  $m_i$  an  $\text{Sign}_{sk}(\cdot)$  mit

$$\text{Sign}_{sk}(H(m_i)) = \begin{cases} \sigma_i & \text{für } i \neq j \\ \text{Abbruch} & \text{sonst} \end{cases}$$

3 Falls  $m = m_j$  und  $\sigma^e = y \bmod N$ , setze  $x \leftarrow \sigma$ .

AUSGABE:  $x$

• Unter der RSA-Annahme gilt  $\text{negl}(n) \geq \text{Ws}[\mathcal{A}'(N, e, x^e) = x]$   
 $= \text{Ws}[m = m_j] \cdot \text{Ws}[\text{Forge}_{\mathcal{A}, \Pi}(n) = 1] = \frac{\epsilon(n)}{q}$ .

• Damit ist  $\epsilon(n) \leq q \cdot \text{negl}(n)$  vernachlässigbar für polynomielles  $q$ .

# Hash-and-Sign Paradigma

**Ziel:** Signaturen für Nachrichten beliebiger Länge

- Starten mit Signaturverfahren  $\Pi$  für  $m \in \{0, 1\}^n$ .
- Verwenden Hashfunktion  $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ .
- Unterschreiben Hashwerte statt der Nachrichten.

## Definition Hash-and-Sign Paradigma

Sei  $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$  und  $\Pi_H = (\text{Gen}_H, H)$  eine Hashfunktion.

- 1 **Gen'**:  $(pk, sk) \leftarrow \text{Gen}(1^n)$ ,  $s \leftarrow \text{Gen}_H(1^n)$ .  
Ausgabe  $pk' = (pk, s)$  und  $sk' = (sk, s)$ .
- 2 **Sign'**: Für eine Nachricht  $m \in \{0, 1\}^*$  berechne  
$$\sigma \leftarrow \text{Sign}_{sk}(H_s(m)).$$
- 3 **Vrfy'**: Für eine Nachricht  $m \in \{0, 1\}^*$  mit Signatur  $\sigma$  prüfe  
$$\text{Vrfy}_{pk}(H_s(m), \sigma) \stackrel{?}{=} 1.$$

**Intuition:** Fälschung impliziert Fälschung in  $\Pi$  oder Kollision in  $H$ .

# Sicherheit von Hash-and-Sign

## Satz Sicherheit des Hash-and-Sign Paradigmas

Sei  $\Pi$  CMA-sicher und  $\Pi_H$  kollisionsresistent. Dann ist das Hash-and-Sign Signaturverfahren  $\Pi'$  CMA-sicher.

### Beweis:

- Sei  $\mathcal{A}$  ein Angreifer für Hash-and-Sign  $\Pi'$  mit Ausgabe  $(m, \sigma)$ .
- Sei  $Q = \{m_1, \dots, m_q\}$  die Menge der von  $\mathcal{A}$  an das Signierorakel  $Sign_{sk}(\cdot)$  gestellten Anfragen. Es gilt  $m \notin Q$ .
- Sei  $coll$  das Ereignis, dass  $m_i \in Q$  mit  $H_s(m_i) = H_s(m)$ .
- Dann gilt 
$$\begin{aligned} \mathbb{W}_s[Forge_{\mathcal{A}, \Pi'}(n) = 1] &= \mathbb{W}_s[Forge_{\mathcal{A}, \Pi'}(n) = 1 \wedge coll] + \mathbb{W}_s[Forge_{\mathcal{A}, \Pi'}(n) = 1 \wedge \overline{coll}] \\ &\leq \mathbb{W}_s[coll] + \mathbb{W}_s[Forge_{\mathcal{A}, \Pi'}(n) = 1 \wedge \overline{coll}] \end{aligned}$$
- Wir zeigen nun, dass beide Summanden vernachlässigbar sind.

# Algorithmus für Kollisionen

**Beweis:**  $Ws[coll] \leq \text{negl}(n)$

- Konstruieren mittels  $\mathcal{A}$  einen Algorithmus  $\mathcal{C}$  für Kollisionen.

## Algorithmus $\mathcal{C}$

EINGABE:  $s$

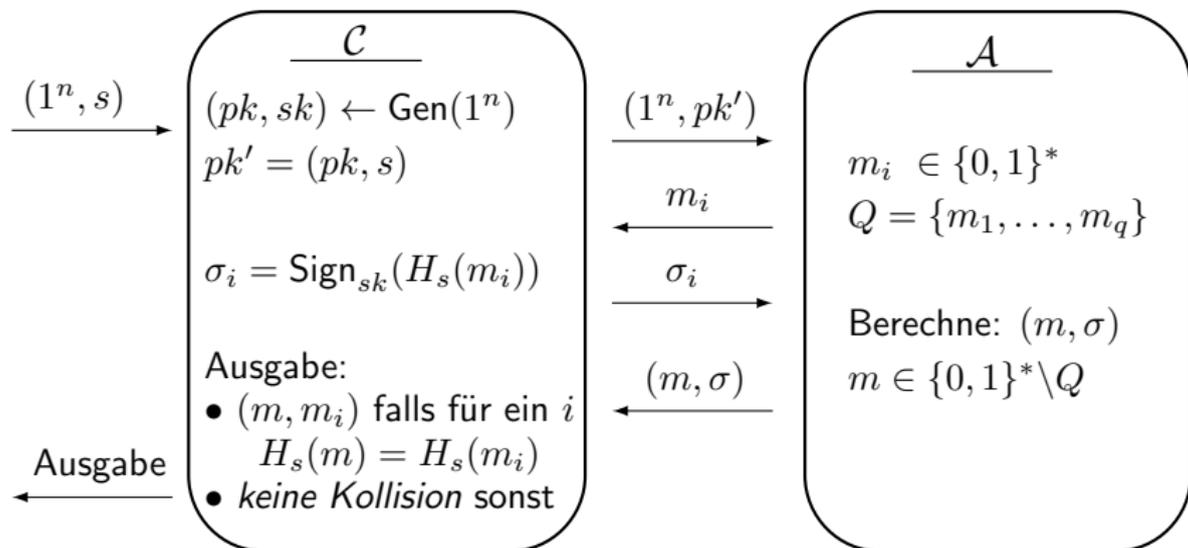
- 1 Berechne  $(pk, sk) \leftarrow \text{Gen}(1^n)$ . Setze  $pk' \leftarrow (pk, s)$ .
- 2  $(m, \sigma) \leftarrow \mathcal{A}(pk')$ . Auf Orakelanfrage  $m_i \in \{0, 1\}^*$ , antworte mit  $\sigma_i \leftarrow \text{Sign}_{sk}(H_s(m_i))$ .

AUSGABE:  $\begin{cases} (m, m_i) & \text{falls } H_s(m) = H_s(m_i) \text{ für ein } m_i \\ \text{keine Kollision} & \text{sonst} \end{cases}$ .

- Es gilt  $Ws[coll] = Ws[\text{HashColl}_{\mathcal{C}, \Pi_H}(n) = 1]$ .
- Aus der Kollisionsresistenz von  $H$  folgt

$$Ws[\text{HashColl}_{\mathcal{C}, \Pi_H}(n) = 1] \leq \text{negl}(n).$$

# Algorithmus $\mathcal{C}$ für Kollisionen



# Fälschen von Signaturen in $\Pi$

**Beweis:**  $Ws[Forge_{\mathcal{A},\Pi'}(n) = 1 \wedge \overline{coll}] \leq \text{negl}(n)$

- Konstruieren mittels  $\mathcal{A}$  einen Angreifer  $\mathcal{A}'$  für  $\Pi$ .

## Algorithmus $\mathcal{A}'$

EINGABE:  $pk$ , Zugriff auf Signierorakel  $Sign_{sk}(\cdot)$

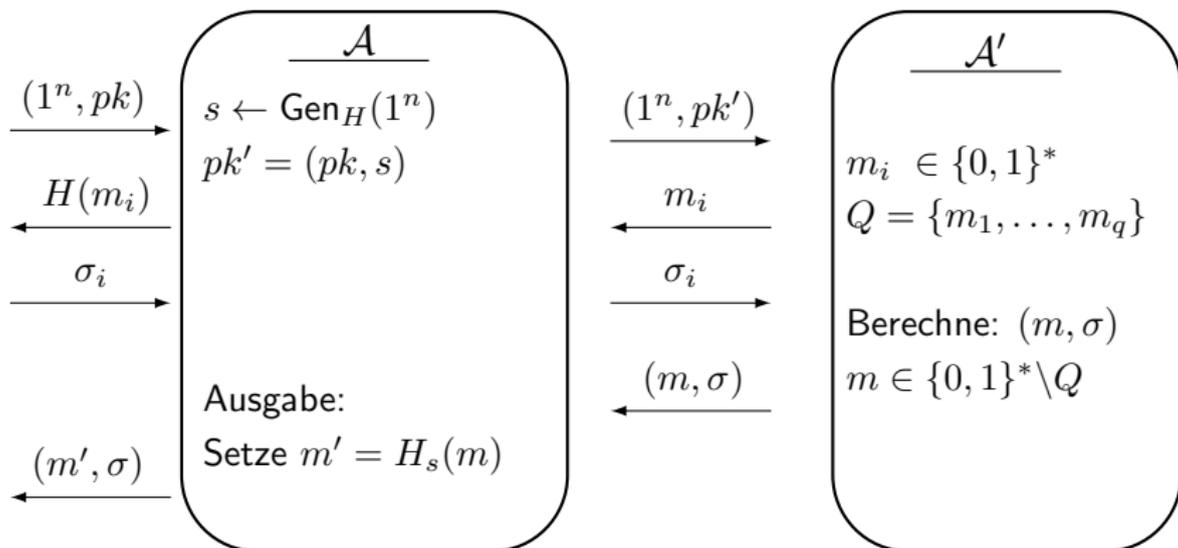
- 1 Berechne  $s \leftarrow Gen_H(1^n)$ . Setze  $pk' = (pk, s)$ .
- 2  $(m, \sigma) \leftarrow \mathcal{A}(pk')$ . Beantworte Orakelanfrage  $m_i \in \{0, 1\}^*$  mit Ausgabe  $\sigma_i \leftarrow Sign_{sk}(H_s(m_i))$  des Signierorakels.
- 3 Setze  $m' \leftarrow H_s(m)$ .

AUSGABE:  $(m', \sigma)$

- Falls  $(m, \sigma)$  gültig ist für  $\Pi'$ , so ist  $(m', \sigma) = (H_s(m), \sigma)$  gültig für  $\Pi$ .
- Ereignis  $\overline{coll}$  bedeutet, dass  $m' \neq H_s(m_i)$  für alle Anfragen  $H_s(m_i)$ .
- Damit gilt  $Ws[Forge_{\mathcal{A},\Pi'}(n) \wedge \overline{coll}] = Ws[Forge_{\mathcal{A}',\Pi}(n) = 1]$ .
- Aus der CMA-Sicherheit von  $\Pi$  folgt

$$Ws[Forge_{\mathcal{A}',\Pi}(n) = 1] \leq \text{negl}(n).$$

# Algorithmus $\mathcal{A}$ für Fälschungen



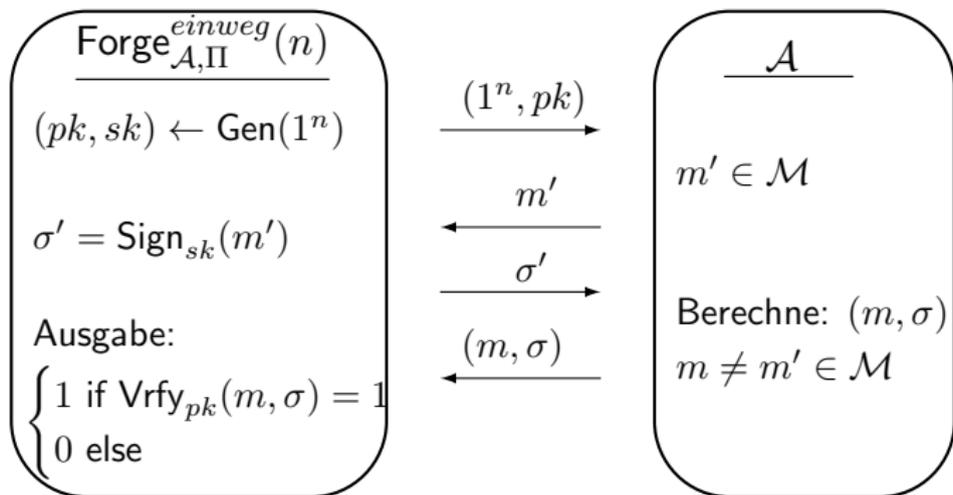
# Einwegsignaturen

## Ziel: Einwegsignaturen

- Konstruieren Verfahren zum sicheren Signieren *einer* Nachricht.
- Konstruktion mittels kollisionsresistenter Hashfunktionen.

### Spiel $Forge_{\mathcal{A}, \Pi}^{\text{einweg}}(n)$

- 1  $(pk, sk) \leftarrow Gen(1^n)$
- 2  $(m, \sigma) \leftarrow \mathcal{A}^{Sign_{sk}(\cdot)}(pk)$ , wobei  $\mathcal{A}$  **eine** Nachricht  $m' \neq m$  an  $Sign_{sk}(\cdot)$  anfragen darf.
- 3  $Forge_{\mathcal{A}, \Pi}^{\text{einweg}}(n) = \begin{cases} 1 & \text{falls } Vrfy_{pk}(m, \sigma) = 1 \\ 0 & \text{sonst} \end{cases}$ .



## Definition CMA-sichere Einwegsignaturen

Ein Signaturverfahren  $\Pi$  heißt *CMA-sichere Einwegsignatur*, falls für alle ppt  $\mathcal{A}$  gilt  $\Pr[\text{Forge}_{\mathcal{A}, \Pi}^{\text{einweg}}(n) = 1] \leq \text{negl}(n)$ .

# Beispiel von Lamports Einwegsignaturen

## Illustration: Signieren einer 3-Bit Nachricht

- Verwende Einwegfunktion  $f : D \rightarrow R$ .
- Wähle als geheimen Schlüssel 6 Element  $x_{i,j}$  zufällig aus  $D$ . Setze

$$sk = \begin{pmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{pmatrix}.$$

- Für alle  $x_{i,j}$  berechne  $y_{i,j} = f(x_{i,j})$ . Dies liefert

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{pmatrix}.$$

- Unterschreibe  $m = m_1 m_2 m_3 \in \{0, 1\}^3$  mit  $\sigma = x_{1,m_1} x_{2,m_2} x_{3,m_3}$ .
- Verifikation von  $(m, \sigma)$ : Überprüfe  $f(\sigma_i) = y_{i,m_i}$  für  $i = 1, 2, 3$ .

# Lamports Einwegsignaturen

## Definition Lamport Einwegsignaturen

Sei  $f$  eine Einwegfunktion. Konstruieren Signaturen für  $m \in \{0, 1\}^\ell$ .

- **Gen:** Bei Eingabe  $1^n$ :

Wähle  $x_{i,j} \in_R \{0, 1\}^n$ , berechne  $y_{i,j} := f(x_{i,j})$  für  $i \in [\ell], j \in \{0, 1\}$ .

Setze  $sk = \begin{pmatrix} x_{1,0} & \dots & x_{\ell,0} \\ x_{1,1} & \dots & x_{\ell,1} \end{pmatrix}$  und  $pk = \begin{pmatrix} y_{1,0} & \dots & y_{\ell,0} \\ y_{1,1} & \dots & y_{\ell,1} \end{pmatrix}$ .

- **Sign:** Für  $m_1 \dots m_\ell \in \{0, 1\}^\ell$ , Ausgabe  $(m, \sigma)$  mit

$$\sigma = (x_{1,m_1}, \dots, x_{\ell,m_\ell}).$$

- **Vrfy:** Für  $(m, \sigma)$  überprüfe  $f(\sigma_i) \stackrel{?}{=} y_{i,m_i}$  für  $i \in [\ell]$ .

# Sicherheit von Lamport Einwegsignaturen

## Satz CMA-Sicherheit von Lamport

Unter der Annahme, dass  $f$  eine Einwegfunktion ist sind Lamport Einwegsignaturen CMA-sicher für Nachrichtenlänge  $\ell$  polynomiell in  $n$ .

**Beweis:** Sei  $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$  das Lamport Signaturverfahren.

- Sei  $\mathcal{A}$  ein Angreifer mit  $\epsilon(n) := \text{Ws}[\text{Forge}_{\mathcal{A}, \Pi}^{\text{einweg}}(n) = 1]$ .
- Wir konstruieren einen Invertierer  $\mathcal{A}'$  für  $f$  mittels  $\mathcal{A}$ .

## Algorithmus Invertierer $\mathcal{A}'$

EINGABE:  $y$

- 1 Wähle  $i \in_R \{0, 1\}^\ell$  und  $b \in_R \{0, 1\}$ .
- 2 Berechne  $(pk, sk) \leftarrow \text{Gen}(1^n)$ . Setze  $y_{i,b} \leftarrow y$  in  $pk$ .
- 3  $(m, \sigma) \leftarrow \mathcal{A}$ . Bei Signaturanfrage für  $m'$  antworte mit  $\sigma' = (\sigma_{1,m'_1}, \dots, \sigma_{\ell,m'_\ell})$  falls  $m'_i \neq b$ . Sonst Abbruch.

AUSGABE:  $= \begin{cases} x_i & \text{falls } m_i = b \\ \text{Abbruch} & \text{sonst} \end{cases}$ .

# Sicherheit von Lamport Einwegsignaturen

## Beweis: Fortsetzung

- Sei  $(m, \sigma)$  eine gültige Signatur mit  $m_i = b$ .
- Dann ist  $\sigma_i$  ein Urbild von  $y$ , d.h.  $f(\sigma_i) = y$ .
- Wahl von  $y$  im Invertier-Spiel erfolgt durch  $x \in_R D$  und  $y := f(x)$ .
- D.h.  $pk$  ist identisch verteilt zum Lamport Signaturverfahren.
  
- Benötigen  $m'_i \neq b$  und  $m_i = b \neq m_i$ . Es gilt  $\text{Ws}[m'_i \neq b] = \frac{1}{2}$ .
- Wegen  $m \neq m'$  folgt  $\text{Ws}[m'_i \neq m_i] \geq \frac{1}{\ell}$ .
- Wir erhalten insgesamt  $\text{Ws}[Invert_{\mathcal{A},f}(n) = 1]$ 
  - $= \text{Ws}[Forge_{\mathcal{A},\Pi}^{einweg}(n) \wedge (m'_i \neq b) \wedge (m'_i \neq m_i)]$
  - $= \text{Ws}[Forge_{\mathcal{A},\Pi}^{einweg}(n)] \cdot \text{Ws}[(m'_i \neq b)] \cdot \text{Ws}[(m'_i \neq m_i)] \geq \epsilon(n) \cdot \frac{1}{2\ell}$
- Aufgrund der Einweg-Eigenschaft von  $f$  gilt
$$\text{negl}(n) \geq \text{Ws}[Invert_{\mathcal{A},f}(n) = 1].$$
- Daraus folgt  $\epsilon(n) \leq 2\ell \cdot \text{negl}(n)$ .
- Dies ist vernachlässigbar für polynomielles  $\ell$ .

# Einwegsignaturen für Nachrichten beliebiger Länge

## Satz Einwegsignaturen für Nachrichten beliebiger Länge

Unter der Annahme kollisionsresistenter Hashfunktionen existiert ein CMA-sicheres Einwegsignatur-Verfahren für Nachrichten beliebiger Länge.

### Beweis:

- Aus der Existenz von kollisionsresistenten Hashfunktionen folgt die Existenz von Einwegfunktionen. (Übung, siehe auch Krypto I)
- Nutzen Einwegfunktion  $f$  zur Konstruktion von CMA-sicheren Lamport-Signaturen der Nachrichtenlänge  $\ell$ .
- Nutzen Hash-and-Sign Paradigma für beliebige Nachrichtenlänge. Hier verwenden wir erneut kollisionsresistente Hashfunktionen.

# Einfache $k$ -weg Signaturen

## $k$ -weg Signaturen

- Definiere mittels  $k$ -maliger Anwendung von  $Gen_{\text{Lamport}}(1^n)$   
 $pk = (pk_1, \dots, pk_k)$  und  $sk = (sk_1, \dots, sk_k)$ .
- Unterzeichnen die  $i$ -te Nachricht  $m$  mittels  $sk_i$  als  $(m, \sigma)$ .
- Man bezeichnet  $i$  auch als *Zustand* im Signaturverfahren.
- $(m, \sigma)$  gilt als gültig, falls  $(m, \sigma)$  für ein  $pk_i$  gültig ist.

## Nachteile:

- Anzahl  $k$  muss bei Schlüsselgenerierung feststehen.
- Länge von  $pk$  und  $sk$  hängen beide von  $k$  ab.

## Idee:

- Konstruiere neues Schlüsselpaar nur bei Bedarf.
- Validiere  $(pk_{i+1}, sk_{i+1})$  mittels  $(pk_i, sk_i)$ .

## Zwei Signaturen mit einem öffentlichen Schlüssel

- Sei  $\Pi = (Gen, Sign, Vrfy)$  ein Einwegsignaturverfahren.
- Konstruieren  $\Pi' = (Gen', Sign', Vrfy')$  für zwei Signaturen.

### Gen':

- Erzeuge  $(pk_1, sk_1) \leftarrow Gen(1^n)$ .

### Sign' und Vrfy' von $m_1$ :

- Erzeuge  $(pk_2, sk_2) \leftarrow Gen(1^n)$ . Berechne  $\sigma'_1 \leftarrow Sign_{sk_1}(m_1 || pk_2)$ .
- Ausgabe der Signatur  $\sigma_1 = (pk_2, \sigma'_1)$ . Merke  $(m_1, \sigma_1, sk_2)$ .
- Verifikation von  $(m_1, \sigma_1) = (m_1, pk_2, Sign_{sk_1}(m_1 || pk_2))$ :

Überprüfe  $Vrfy_{pk_1}(m_1 || pk_2, \sigma'_1) \stackrel{?}{=} 1$ .

### Sign' und Vrfy' von $m_2$ :

- Erzeuge  $(pk_3, sk_3) \leftarrow Gen(1^n)$ . Berechne  $\sigma'_2 \leftarrow Sign_{sk_2}(m_2 || pk_3)$ .
- Ausgabe  $\sigma_2 = (m_1, \sigma_1, pk_3, \sigma'_2)$ . Merke  $(m_2, \sigma_2, sk_3)$
- Verifikation von  $(m_2, \sigma_2) = (m_2, m_1, pk_2, \sigma'_1, pk_3, \sigma'_2)$ :

Überprüfe  $Vrfy_{pk_1}(m_1 || pk_2, \sigma'_1) \stackrel{?}{=} 1$  **und**  $Vrfy_{pk_2}(m_2 || pk_3, \sigma'_2) \stackrel{?}{=} 1$ .

# Beliebige Anzahl von Signaturen

## Algorithmus Signaturketten

Sei  $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$  ein Einwegsignaturverfahren.

- 1 **Gen'**:  $(pk_1, sk_1) \leftarrow \text{Gen}(1^n)$
- 2 **Sign'**: Signieren der Nachricht  $m_i$ .
  - ▶ Verwende gemerkten Zustand  $(m_{i-1}, \sigma_{i-1}, sk_i)$ .
  - ▶  $(pk_{i+1}, sk_{i+1}) \leftarrow \text{Gen}(1^n)$ . Berechne  $\sigma'_i = \text{Sign}_{sk_i}(m_i || pk_{i+1})$ .
  - ▶ Ausgabe  $\sigma_i = (m_{i-1}, \sigma_{i-1}, pk_{i+1}, \sigma'_i)$ . Merke  $(m_i, \sigma_i, sk_{i+1})$ .
- 3 **Vrfy'**: Verifikation von  $(m_i, \sigma_i) \stackrel{\text{Sortieren}}{=} (m_j, pk_{j+1}, \sigma'_j)_{j=1}^i$ :  
Überprüfe  $\text{Vrfy}_{pk_j}(m_j || pk_{j+1}, \sigma'_j) \stackrel{?}{=} 1$  für  $j = 1, \dots, i$ .

**Vorteil:** Ein öffentlicher Schlüssel  $pk_1$ , beliebige Signaturanzahl.

**Nachteile:**

- Signaturlänge, Zustandsgröße und Verifikationszeit sind linear in der Anzahl der signierten Dokumente.
- Signaturen geben alle zuvor signierten Dokumente preis.

# Merkle-Baum

## Idee: Konstruktion von Merkle-Bäumen

- Ersetze Signaturkette durch Baum (sogenannter Merkle-Baum).
- Verwenden Baum der Tiefe  $n$  für Nachrichten der Länge  $n$ .
- Die Wurzel erhält Label  $\epsilon$ .
- Die Kinder eines Knotens mit Label  $w$  erhalten Label  $w0$  und  $w1$ .
- Blätter besitzen Nachrichten-Label  $m \in \{0, 1\}^n$ .
- Knoten mit Label  $w$  speichern Schlüsselpaar  $pk_w, sk_w$ .
- Wurzelschlüssel  $pk_\epsilon$  ist der öffentliche Schlüssel.

## Ziel: Zertifiziere Pfad von Wurzel zu $m$ mittels Signaturen.

- Signieren Public-Keys auf Pfad inklusive der Nachbarknoten.

# Signieren und Verifizieren von $m = 001$

## Signieren von $m = 001$

- Pfad von Wurzel zu  $m$ :  $\epsilon, 0, 00, 001$  mit Nachbarknoten  $1, 01, 000$ .
- Signiere  $(pk_0 || pk_1)$  mittels  $sk_\epsilon$ . Sei dies  $\sigma_\epsilon$ .
- Signiere  $(pk_{00} || pk_{01})$  mittels  $sk_0$ . Sei dies  $\sigma_0$ .
- Signiere  $(pk_{000} || pk_{001})$  mittels  $sk_{00}$ . Sei dies  $\sigma_{00}$ .
- Signiere  $m$  mittels  $sk_{001}$ . Sei dies  $\sigma_{001}$ .
- Signatur ist  $\sigma = (pk_0 || pk_1, \sigma_\epsilon, pk_{00} || pk_{01}, \sigma_0, pk_{000} || pk_{001}, \sigma_{00}, \sigma_{001})$

## Verifizieren von $(m, \sigma)$ :

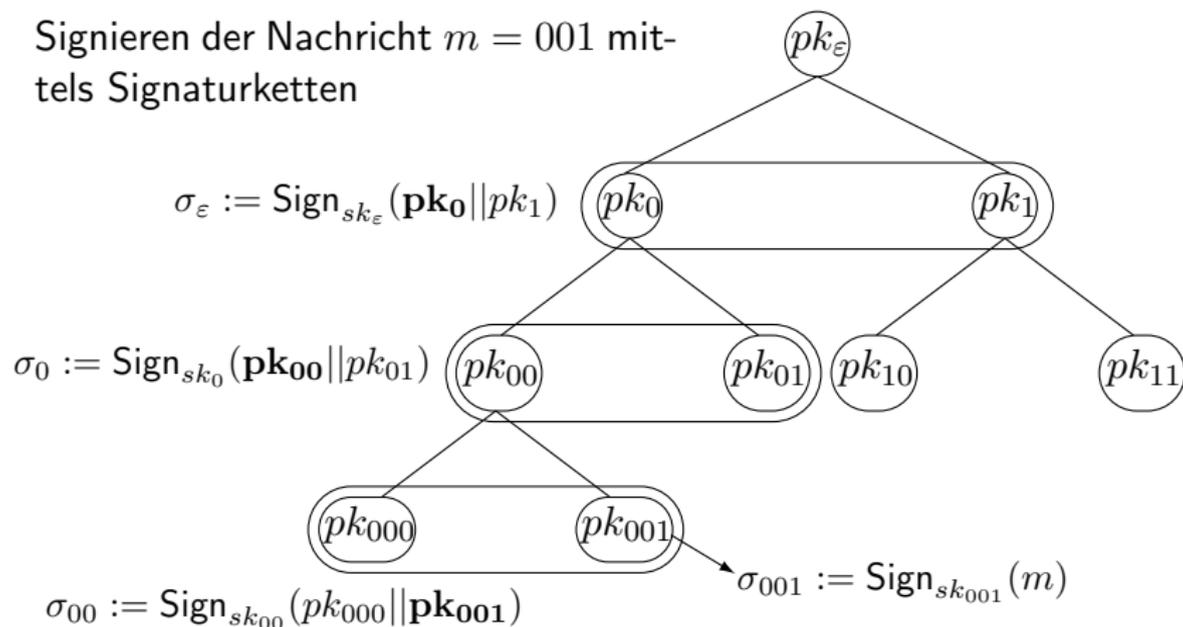
- Verifiziere  $(pk_0 || pk_1, \sigma_\epsilon)$  mittels  $pk_\epsilon$ .
- Verifiziere  $(pk_{00} || pk_{01}, \sigma_0)$  mittels  $pk_0$ .
- Verifiziere  $(pk_{000} || pk_{001}, \sigma_{00})$  mittels  $pk_{00}$ .
- Verifiziere  $(001, \sigma_{001})$  mittels  $pk_{001}$ .

## Notation:

- Sei  $m \in \{0, 1\}^n$ . Wir definieren  $m|_i := m_1 \dots m_i$  für  $i = 0, \dots, n$ .
- D.h.  $m|_i$  ist der  $i$ -Zeichen Präfix von  $m$  und  $m|_i = \epsilon$ .

# Signaturpfad der Nachricht $m = 001$

Signieren der Nachricht  $m = 001$  mittels Signaturketten



# Merkle Signaturen

## Algorithmus Merkle Signatur

Sei  $\Pi' = (Gen', Vrfy', Sign')$  ein Einwegsignaturverfahren.

- 1 **Gen:**  $(pk_\epsilon, sk_\epsilon) \leftarrow Gen'(1^n)$
- 2 **Sign:** Für Nachricht  $m \in \{0, 1\}^n$ : FOR  $i := 0$  to  $n - 1$ 
  - ▶ Falls  $pk_{m|i,0}, pk_{m|i,1}$  noch nicht erzeugt, erzeuge und speichere  $(pk_{m|i,0}, sk_{m|i,0}) \leftarrow Gen'(1^n), (pk_{m|i,1}, sk_{m|i,1}) \leftarrow Gen'(1^n)$ .
  - ▶ Erzeuge  $\sigma_{m|i} \leftarrow Sign'_{sk_{m|i}}(pk_{m|i,0}, pk_{m|i,1})$ .

Berechne  $\sigma_m \leftarrow Sign'_{sk_m}(m)$

Ausgabe von  $\sigma = ((pk_{m|i,0} || pk_{m|i,1}, \sigma_{m|i})_{i=0}^{n-1}, \sigma_m)$ .

- 3 **Vrfy':** Für  $(m, \sigma)$  überprüfe  $Vrfy'_{pk_{m|i}}(pk_{m|i,0} || pk_{m|i,1}, \sigma_{m|i}) \stackrel{?}{=} 1$  für  $i = 0, \dots, n - 1$  und  $Vrfy'_{pk_m}(m, \sigma_m) \stackrel{?}{=} 1$ .

# Eigenschaften von Merkle Signaturen

## Eigenschaften:

- Erlaubt das Signieren aller möglichen  $2^n$  Nachrichten.
- Schlüsselpaare werden nur bei Bedarf erzeugt.

## Vorteile: gegenüber Signaturketten

- Signaturlänge/Verifikationszeit sind linear in der Nachrichtenlänge aber unabhängig von der Anzahl Signaturen.
- Keine Preisgabe der zuvor signierten Nachrichten.

## Satz Sicherheit von Merkle Signaturen

Sei  $\Pi'$  eine CMA-sichere Einwegsignatur. Dann sind Merkle Signaturen  $\Pi$  ein CMA-sicheres Signaturverfahren.

## Beweis:

- Sei  $\mathcal{A}$  ein Angreifer mit  $\epsilon(n) := Ws[Forge_{\mathcal{A},\Pi}(n) = 1]$ .
- $\mathcal{A}'$  frage höchstens  $q$  Signaturen an. Setze  $\ell := 2nq + 1$  als obere Schranke für die Anzahl der benötigten Schlüssel in  $\Pi$ .
- Wir konstruieren mittels  $\mathcal{A}$  einen Angreifer  $\mathcal{A}'$  für  $\Pi'$ .

# Sicherheit von Merkle Signaturen

## Algorithmus Angreifer $\mathcal{A}$ für die Einwegsignatur

EINGABE:  $pk$ , Zugriff auf eine Anfrage an Orakel  $Sign_{sk}(\cdot)$

- 1 Berechne  $(pk^{(i)}, sk^{(i)}) \leftarrow Gen(1^n)$  für  $i = 1, \dots, \ell$ .  
Wähle  $i' \in_R [\ell]$ . Ersetze  $pk^{(i')}$  durch  $pk$ .  $j \leftarrow 2$
- 2  $(m, \sigma') \leftarrow \mathcal{A}'(pk^{(1)})$ . Signaturanfragen für  $m$ : For  $i \leftarrow 1$  to  $n - 1$ 
  - ▶ Falls  $pk_{m|i,0}, pk_{m|i,1}$  undefiniert,  $(pk_{m|i,0}, pk_{m|i,1}) \leftarrow (pk^{(j)}, pk^{(j+1)})$ .  
 $j \leftarrow j + 2$
  - ▶ Berechne  $\sigma_{m|i}, \sigma_m$  und  $\sigma$  analog zu Merkle-Signaturen.  
Falls  $sk = sk^{(i')}$  benötigt, verwende das Signierorakel  $Sign_{sk}(\cdot)$ .
- 3 Sei  $\sigma' = ((pk'_{m|i,0} || pk'_{m|i,1}, \sigma'_{m|i})_{i=0}^{n-1}, \sigma'_m)$ 
  - ▶ **Fall 1:**  $\exists 0 \leq i < n$  mit  $(pk'_{m|i,0} || pk'_{m|i,1}) \neq (pk_{m|i,0} || pk_{m|i,1})$ .  
Sei  $i$  minimal. Dann gilt  $pk'_{m|i} = pk_{m|i} = pk^{(k)}$  für ein  $k \in [\ell]$ .  
Falls  $k = i'$ , Ausgabe  $(pk'_{m|i,0} || pk'_{m|i,1}, \sigma'_{m|i})$ .
  - ▶ **Fall 2:** Es gilt  $pk'_m = pk_m = pk^{(k)}$  für ein  $k \in [\ell]$ .  
Falls  $k = i'$ , Ausgabe  $(m, \sigma'_m)$ .

# Sicherheit von Merkle Signaturen

## Beweis: Fortsetzung

- Verteilung der Nachrichten für  $\mathcal{A}'$  ist identisch zum Forge-Spiel.
- D.h.  $\mathcal{A}$  liefert eine gültige Signatur  $(m', \sigma')$  mit  $\text{Ws } \epsilon(n)$ .
- Sowohl für Fall 1 als auch für Fall 2 gilt  $\text{Ws}[k = i'] = \frac{1}{\ell}$ .
- Wir nehmen im folgenden an, dass  $k = i'$ .
- **Fall 1:**  $\exists$  neuer Public-Key in Geschwisterknotenpaar.
- $\mathcal{A}$  stellte eventuell Orakelanfrage für Nachricht  $(pk_{m|i,0} || pk_{m|i,1})$ .
- Wegen  $(pk'_{m|i,0} || pk'_{m|i,1}) \neq (pk_{m|i,0} || pk_{m|i,1})$  ist  $\sigma'_{m|i}$  bezüglich  $pk$  eine gültige Signatur für eine neue Nachricht.
- **Fall 2:**  $pk'_m$  existiert bereits.
- $\mathcal{A}'$  kann nicht Orakelanfrage  $m$  gestellt haben, da er  $m$  ausgibt.
- Damit ist  $\sigma'_m$  eine gültige neue Signatur für  $m$  bezüglich  $pk$ .
- **Insgesamt:**  $\text{negl}(n) \geq \text{Ws}[Forge_{\mathcal{A}, \Pi}^{\text{einweg}}(n) = 1] = \frac{\epsilon(n)}{\ell}$ .
- Da  $\ell$  polynomiell ist, folgt  $\epsilon(n) \leq \text{negl}(n)$ .

# Existenz CMA-sicherer Signatur

## Korollar Signatursatz

Falls kollisionsresistente Hashfunktionen existieren, so existiert ein CMA-sicheres Signaturverfahren.

### Beweis:

- Unter der Annahme kollisionsresistenter Hashfunktionen existieren CMA-sichere Einwegsignaturen.
- Aus CMA-sicheren Einwegsignaturen können CMA-sichere Signaturen konstruiert werden.

### Anmerkung:

- Man kann sogar zeigen, dass ein CMA-sicheres Signaturverfahren existiert unter der Annahme der Existenz von Einwegfunktionen.

# Digital Signature Standard – Schnorr Signaturen

## Systemparameter:

- Primzahlen  $p, q$ , wobei  $q$  Bitlänge  $n$  besitzt und  $q|p-1$ ,  $q^2 \nmid p-1$ .
- Generator  $g$  einer Untergruppe von  $\mathbb{Z}_p^*$  mit Ordnung  $q$ .

## Algorithmus Digital Signature Standard

- 1 Gen:**  $(p, q, g, H) \leftarrow \text{Gen}(1^n)$  mit Hashfunktion  $H: \{0, 1\}^* \rightarrow \mathbb{Z}_q$ .  
Wähle  $x \in_R \mathbb{Z}_q$ , berechne  $y := g^x \bmod p$ .  
Setze  $pk = (p, q, g, H, y)$ ,  $sk = (p, q, g, H, x)$ .
- 2 Sign:** Für  $m \in \{0, 1\}^*$ , wähle  $k \in_R \mathbb{Z}_q^*$  und berechne  
 $r := (g^k \bmod p) \bmod q$  und  $s := (H(m) + xr) \cdot k^{-1} \bmod q$ .  
Signatur  $\sigma = (r, s)$ .
- 3 Vrfy:** Für  $(m, \sigma) = (m, r, s)$  überprüfe  
 $r \stackrel{?}{=} (g^{H(m)} \cdot s^{-1} \bmod q) \cdot y^{r \cdot s^{-1} \bmod q} \bmod p \bmod q$ .

# Eigenschaften des DSS

## Korrektheit:

$$\begin{aligned}g^{H(m) \cdot s^{-1}} y^{r \cdot s^{-1}} &= g^{H(m)(H(m)+xr)^{-1}k} g^{xr(H(m)+xr)^{-1}k} \pmod p \\ &= g^{(H(m)+xr) \cdot (H(m)+xr)^{-1}k} = g^k \pmod p.\end{aligned}$$

## Parameterwahl:

- Bitlänge von  $p$ : 1024, Bitlänge  $n$  von  $q$ : 160.
- Die Signaturlänge von  $(r, s) \in \mathbb{Z}_q^2$  ist damit nur 320 Bit.
- Dlog in  $\mathbb{Z}_p^*$ : subexponentieller Index-Calculus Algorithmus
- Dlog in  $\langle g \rangle$ : Pollard-Rho mit Komplexität  $\mathcal{O}(2^{\frac{n}{2}})$ .

## Sicherheit:

- Keine größeren Schwächen bekannt.
- Aber: DSS besitzt **keinen** Sicherheitsbeweis.

# Viele Public-Keys mittels eines Public Keys

## Zertifizierung

- Zertifizierungsstelle CA (Certificate Authority) veröffentlicht  $pk_{CA}$ .
- CA zertifiziert Schlüssel  $pk_A$  eines Nutzers Alice mit Zertifikat

$$cert_{CA \rightarrow A} \leftarrow \text{Sign}_{sk_{CA}}(pk_A).$$

- Alice kann  $(pk_A, cert_{CA \rightarrow A})$  über unsicheren Kanal verschicken.
- CMA-Sicherheit des Signaturverfahrens verhindert erfolgreiches Fälschen eines Zertifikat für einen anderen Schlüssel  $pk'_A$ .
- D.h. mit nur einem öffentlichen Schlüssel kann eine CA beliebig viele weitere öffentliche Schlüssel zertifizieren.
- Liefert sogenannte Public-Key Infrastruktur.

# Übersicht Krypto II - Verschlüsselung

Abkürzung: TD-OWP = Trapdoor-Einwegpermutation

<b>Funktionalität</b>	<b>Konstrukt</b>	<b>Annahme</b>
Schlüsselaustausch <i>Diffie-Hellman</i>	sicher $g^{xy}$	Unterscheidungsannahme <i>DDH</i> : $g^{xy} \leftrightarrow g^z$
PK Verschlüsselung <i>RSA</i> <i>Rabin</i>	CPA $(r^e, hc(r) \oplus m)$ $(x^2, lsb(x) \oplus m)$	TD-OWP <i>RSA-Annahme</i> <i>Faktorisierungsannahme</i>
PK Verschlüsselung <i>ROM-RSA</i>	CPA (ROM) $(r^e, H(r) \oplus m)$	TD-OWP <i>RSA-Annahme</i>
PK Verschlüsselung <i>ROM-RSA2</i>	CCA (ROM) $(r^e, Enc'_{H(r)}(m))$	TD-OWP + PRF <i>RSA-Annahme</i>
PK Verschlüsselung <i>EIGamal</i> <i>Goldwasser-Micali</i> <i>Paillier</i>	CPA $(g^y, g^{xy} \cdot m)$ $z^m \cdot x^2$ $(1 + N)^m \cdot r^N$	Unterscheidungsannahme <i>DDH</i> : $g^{xy} \leftrightarrow g^z$ <i>QR</i> : $QR_N \leftrightarrow QNR_N^{+1}$ <i>DCR</i> : $r^N \bmod N^2 \leftrightarrow r$

# Übersicht Krypto II - Signatur

Abkürzungen:

- OWF = Einwegfunktion
- CRH = kollisionsresistente Hashfunktion

<b>Funktionalität</b>	<b>Sicherheit</b>	<b>Annahme</b>
Signatur <i>RSA-FDH</i>	CMA (ROM) $\sigma = H(m)^d$	TD-OWP <i>RSA-Annahme</i>
Einwegsignatur fester Länge <i>Lamport</i>	CMA $y_{i,j} = f(x_{i,j}), \sigma_i = x_{i,m_i}$	OWF
Einwegsignatur beliebiger Länge <i>Lamport + Hash&amp;Sign</i>	CMA <i>Signiere <math>H(m)</math>.</i>	CRH
Signatur <i>Merkle</i>	CMA <i>Zertifiziere Baumpfad.</i>	CRH

# Konstruktion aus Einwegpermutation

Wir wollen folgende Konstruktionen kurz skizzieren:

- 1 Einwegpermutation  
⇒ Einwegpermutation + Hardcore-Prädikat.
- 2 Einwegpermutation + Hardcore-Prädikat  
⇒ Pseudozufallsgenerator mit 1 Bit Expansion.
- 3 Pseudozufallsgenerator mit 1 Bit Expansion  
⇒ Pseudozufallsgenerator mit polynomieller Expansion.
- 4 Pseudozufallsgenerator mit polynomieller Expansion  
⇒ Pseudozufallsfunktion.

# Einwegperm. $\Rightarrow$ Einwegperm. + Hardcore-Prädikat

## Goldreich-Levin Hardcore-Prädikat

Sei  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  eine Einwegpermutation und

$$g : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}, (x, r) \mapsto (f(x), r).$$

Dann ist  $hc(x, r) := \langle x, r \rangle = \sum_{i=1}^n x_i r_i$  ein Hardcore-Prädikat für  $g$ .

### Beweis:

- Offenbar ist  $g$  ebenfalls eine Einwegpermutation.
- Sei  $\mathcal{A}$  ein polynomieller Angreifer für das Hardcore-Prädikat  $hc$ .
- Vereinfachende Annahme:  $\mathcal{A}$  besitze Erfolgsws

$$W_{S_{x,r \in_R \{0,1\}^n}}[\mathcal{A}(g(x), r) = hc(x, r)] = 1.$$

(Der Beweis für Erfolgsws  $\frac{1}{2} + \frac{1}{poly(n)}$  ist deutlich komplexer.)

- Konstruieren einen Angreifer  $\mathcal{A}'$  zum Invertieren von  $f$ .

## Algorithmus $\mathcal{A}'$

EINGABE:  $1^n, y = f(x) \in \{0, 1\}^n$

- For  $i = 1$  to  $n$ : Setze  $x_i := \mathcal{A}(y, e_i)$  für den  $i$ -ten Einheitsvektor  $e_i$ .

AUSGABE:  $x = x_1 \dots x_n \in \{0, 1\}^n$

## Korrektheit und Laufzeit:

- Es gilt  $hc(y, e_i) = hc(f(x), e_i) = \langle x, e_i \rangle = x_i$ .
- Die Laufzeit von  $\mathcal{A}'$  ist  $n$ -mal die Laufzeit von  $\mathcal{A}$ .

# Konstruktion eines pseudozufälligen Bits

**Idee:** Gegeben sei  $f(x)$ . Dann ist  $hc(x)$  ein pseudozufälliges Bit.

## Satz

Sei  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  eine Einwegpermutation mit Hardcore-Prädikat  $hc$ . Dann ist

$$G : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}, x \mapsto (f(x), hc(x))$$

ein Pseudozufallsgenerator mit Expansionsfaktor  $\ell(n) = n + 1$ .

## Beweis:

- Sei  $\mathcal{A}$  ein Unterscheider für  $G$  mit Erfolgsws

$$\epsilon(n) = \text{Ws}_{x \in_R \{0,1\}^n}[\mathcal{A}(G(x) = 1)] - \text{Ws}_{r \in_R \{0,1\}^{n+1}}[\mathcal{A}(r) = 1].$$

- Konstruieren daraus Angreifer  $\mathcal{A}'$  auf das Hardcore-Prädikat.

# Beweis Pseudozufallsgenerator(1/3)

## Algorithmus $\mathcal{A}'$

EINGABE:  $1^n, y = f(x)$

① Wähle  $r \in_R \{0, 1\}$ . Berechne  $b \leftarrow \mathcal{A}(y \parallel r)$ .

② Setze  $h(x) = \begin{cases} r & \text{für } b = 1 \\ 1 - r & \text{für } b = 0 \end{cases}$ .

AUSGABE:  $h(x)$

**Fall 1:**  $r = hc(x)$

- Dann ist die Verteilung identisch zum Generator  $G$ , d.h.

$$Ws[\mathcal{A}'(f(x)) = hc(x) \mid r = hc(x)] = Ws[\mathcal{A}(G(x)) = 1].$$

## Beweis Pseudozufallsgenerator (2/3)

**Fall 2:**  $r \neq hc(x)$

- Dann gilt

$$\begin{aligned} \text{Ws}[\mathcal{A}'(f(x)) = hc(x) \mid r \neq hc(x)] &= \text{Ws}[\mathcal{A}(f(x) \parallel \overline{hc(x)}) = 0] \\ &= 1 - \text{Ws}[\mathcal{A}(f(x) \parallel \overline{hc(x)}) = 1]. \end{aligned}$$

Unter der Annahme, dass  $hc$  ein Hardcore-Prädikat ist, gilt

$$\begin{aligned} &\frac{1}{2} + \text{negl}(n) \\ \geq &\text{Ws}[\mathcal{A}'(f(x)) = hc(x)] \\ = &\frac{1}{2} \cdot (\text{Ws}[\mathcal{A}'(f(x)) = hc(x) \mid r = hc(x)] + \text{Ws}[\mathcal{A}'(f(x)) = hc(x) \mid r \neq hc(x)]) \\ = &\frac{1}{2} \cdot (\text{Ws}[\mathcal{A}(G(x)) = 1] + 1 - \text{Ws}[\mathcal{A}(f(x) \parallel \overline{hc(x)}) = 1]) \\ = &\frac{1}{2} + \underbrace{\frac{1}{2} \cdot (\text{Ws}[\mathcal{A}(G(x)) = 1] - \text{Ws}[\mathcal{A}(f(x) \parallel \overline{hc(x)}) = 1])}_{\text{zu zeigen: } \epsilon(n)} \end{aligned}$$

Daraus folgt  $\epsilon(n) \leq \text{negl}(n)$  wie gewünscht.

# Beweis Pseudozufallsgenerator (3/3)

## Lemma

$$\epsilon(n) = \frac{1}{2} \cdot \left( \text{Ws}[\mathcal{A}(G(x)) = 1] - \text{Ws}[\mathcal{A}(f(x) || \overline{hc(x)}) = 1] \right)$$

### Beweis:

- Es gilt  $\epsilon(n) = \text{Ws}_{x \in_R \{0,1\}^n}[\mathcal{A}(G(x)) = 1] - \text{Ws}_{r \in_R \{0,1\}^{n+1}}[\mathcal{A}(r) = 1]$ .
- Wir schreiben

$$\begin{aligned} & \text{Ws}_{r \in_R \{0,1\}^{n+1}}[\mathcal{A}(r) = 1] \\ &= \text{Ws}_{r \in_R \{0,1\}^n, r' \in \{0,1\}}[\mathcal{A}(r || r') = 1] \\ &= \text{Ws}_{x \in_R \{0,1\}^n, r' \in \{0,1\}}[\mathcal{A}(f(x) || r') = 1] \quad // f \text{ ist Permutation.} \\ &= \frac{1}{2} \left( \text{Ws}_{x \in_R \{0,1\}^n}[\mathcal{A}(f(x) || hc(x)) = 1] + \text{Ws}_{x \in_R \{0,1\}^n}[\mathcal{A}(f(x) || \overline{hc(x)}) = 1] \right) \\ &= \frac{1}{2} \cdot \text{Ws}_{x \in_R \{0,1\}}[\mathcal{A}(G(x)) = 1] + \frac{1}{2} \cdot \text{Ws}_{x \in_R \{0,1\}^n}[\mathcal{A}(f(x) || \overline{hc(x)}) = 1]. \end{aligned}$$

- Daraus folgt die Behauptung des Lemmas.