Authentisierung und Verschlüsselung [WIP]

Sommersemester 2023

mail@nilslambertz.de
nilslambertz

Alle Angaben ohne Gewähr. Keine Garantie auf Vollständigkeit oder Richtigkeit.

mail@nilslambertz.de
nilslambertz

mail@nilslambertz.de
nilslambertz

1 Einführung

1.1 Ziel von Digitalen Signaturen

Digitale Signaturen sollen **Authentizität** (Dokument wurde von einer bestimmten Person signiert) und **Integrität** (Dokument wurde nicht verändert) sicherstellen.

1.2 Informelle Definition von Signaturen

- **asymmetrische** Verfahren
- Schlüsselpaar (pk, sk)
- Nachricht m wird mit sk signiert und erzeugt Signatur σ
- Mit pk kann überprüft werden, ob eine Signatur σ gültig für eine Nachricht m ist

1.3 Digitale Signaturen

1.3.1 Definition

Ein digitales Signaturverfahren für einen Nachrichtenraum $\mathcal{M}$ ist ein Tupel $\Sigma = (Gen, Sign, Vfy)$ von probabilistischen Polyzeit (PPT) Algorithmen:

- $Gen(1^k) \rightarrow (pk, sk)$
- $Sign(sk, m) \rightarrow \sigma,\ m \in \mathcal{M}$
- $Vfy(pk, m, \sigma) \in \{0, 1\}$

1.3.2 Correctness

Correctness ("Das Verfahren funktioniert"): $\forall (pk, sk) \leftarrow Gen(1^k) \forall m \in \mathcal{M} : Vfy(pk, m, Sign(sk, m)) = 1$

1.4 Sicherheitsdefinitionen

Sicherheit besteht aus einem **Angreifermodell** (was kann der Angreifer tun, welche Angriffsmöglichkeiten stehen zur Verfügung) und einem **Angreiferziel** (was muss der Angreifer tun, um das Verfahren zu brechen).

1.4.1 Angreifermodelle

1. no-message attack (NMA)
   - Angreifer erhält nur pk
2. **non-adaptive chosen-message attack (naCMA)**
   - Angreifer wählt $m_1, \ldots, m_q$
   - Angreifer erhält **danach** pk und Signaturen $\sigma_1, \ldots, \sigma_q$
3. **(adaptive) chosen-message attack (CMA)**
   - Angreifer erhält pk
   - Angreifer wählt dann (adaptiv) $m_1, \ldots, m_q$ und erhält Signaturen $\sigma_1, \ldots, \sigma_q$
   - Adaptiv: Angreifer darf Wahl von m_i abhängig von vorherigen σ_j $(j < i)$ und pk machen

1.4.2 Angreiferziele

1. Universal Unforgeability (UUF)
   - Nachricht m wird zufällig gewählt
   - Angreifer muss m signieren

2. **Existential Unforgeablility (EUF)**
   - Angreifer kann Nachricht m beliebig wählen und diese signieren

In den **Sicherheitsdefinitionen** werden **Angreiferziel** und **Angreifermodell** kombiniert, z.B.

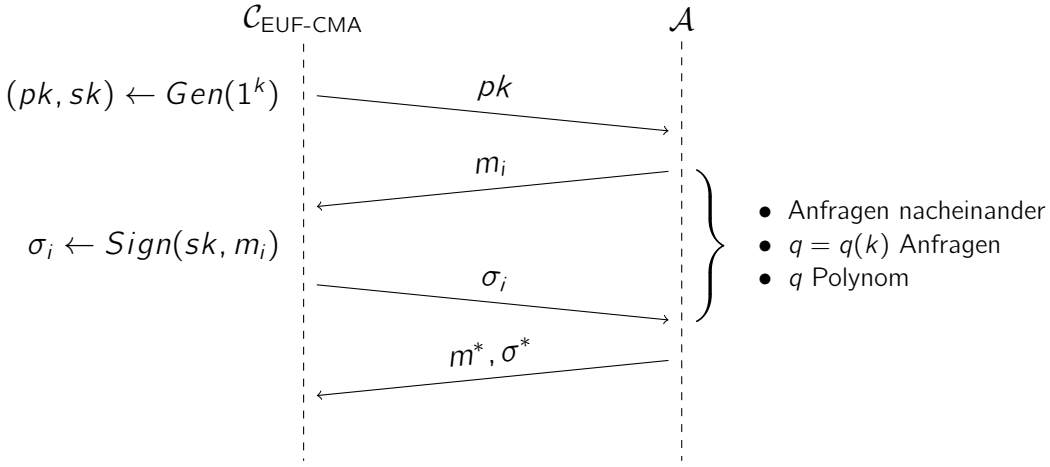
- EUF-CMA
- EUF-naCMA

1.5 EUF-CMA-Sicherheitsexperiment

Bei Sicherheitsexperimenten spielt ein Angreifer $\mathcal{A}$ gegen einen Challenger $\mathcal{C}$. $\mathcal{A}$ gewinnt, falls er die Sicherheit des Verfahrens bricht.

$\mathcal{A}$ muss dabei mit einer nicht vernachlässigbaren Wahrscheinlichkeit eine gültige Signatur erzeugen können, ohne den Schlüssel sk zu kennen.

1.5.1 Visualisierung: EUF-CMA-Sicherheitsexperiment



$$Vfy(pk, m^*, \sigma^*) = 1?$$
$$\wedge$$
$$m^* \notin \{m_1, \ldots, m_q\}?$$

$\mathcal{A}$ gewinnt, falls $Vfy(pk, m^*, \sigma^*) = 1$ **und** $m^* \notin \{m_1, \ldots, m_q\}$

1.5.2 Definition: Vernachlässigbarkeit

Eine Funktion $negl : \mathbb{N} \to [0, 1]$ ist *vernachlässigbar*, wenn

$$\forall c \in \mathbb{N} \exists k_0 \in \mathbb{N} \forall k \geq k_0 : negl(k) < \frac{1}{k^c}$$

1.5.3 Definition: EUF-CMA

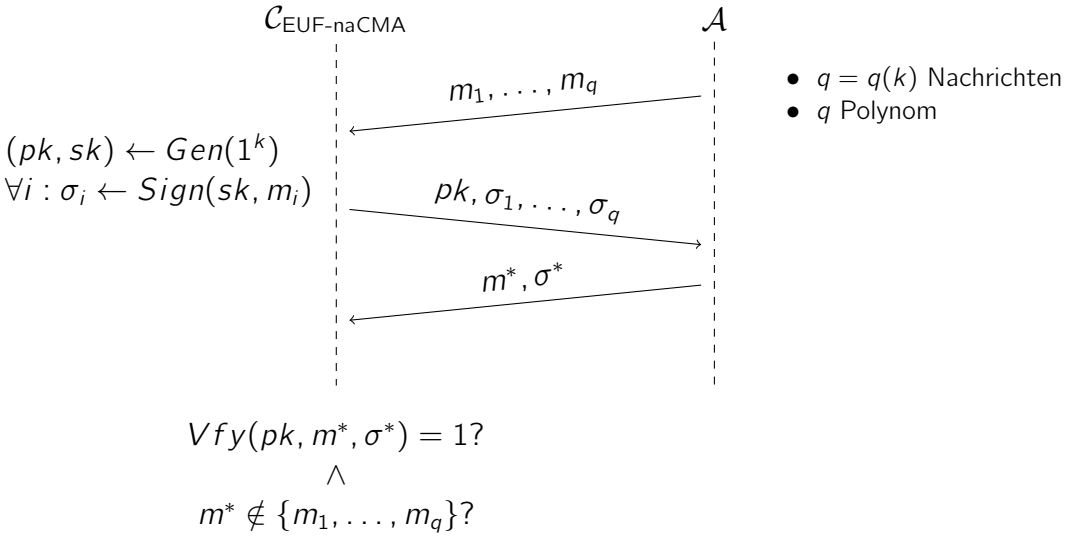
Ein digitales Signaturverfahren $\Sigma = (Gen, Sign, Vfy)$ ist *EUF-CMA-sicher*, wenn für alle PPT $\mathcal{A}$ gilt, dass

$$\begin{aligned}
&\Pr[\mathcal{A} \text{ gewinnt EUF-CMA-Experiment}] \\
&= \Pr[\mathcal{A}^{\mathcal{C}_{\text{EUF-CMA}}}(pk) = (m^*, \sigma^*) : Vfy(pk, m^*, \sigma^*) = 1 \wedge m^* \notin \{m_1, \ldots, m_q\}] \\
&\leq negl(k)
\end{aligned}$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

1.6 EUF-naCMA-Sicherheitsexperiment

1.6.1 Visualisierung: EUF-naCMA-Sicherheitsexperiment



$\mathcal{A}$ gewinnt, falls $Vfy(pk, m^*, \sigma^*) = 1$ **und** $m^* \notin \{m_1, \ldots, m_q\}$

1.6.2 Definition: EUF-naCMA

Ein digitales Signaturverfahren $\Sigma = (Gen, Sign, Vfy)$ ist *EUF-naCMA-sicher*, wenn für alle PPT $\mathcal{A}$ gilt, dass

$$\begin{aligned}&\Pr[\mathcal{A} \text{ gewinnt EUF-naCMA-Experiment}]\\ &= \Pr[\mathcal{A}^{\mathcal{C}_{\text{EUF-naCMA}}} = (m^*, \sigma^*) : Vfy(pk, m^*, \sigma^*) = 1 \wedge m^* \notin \{m_1, \ldots, m_q\}]\\ &\leq negl(k)\end{aligned}$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

1.7 Einmalsignaturen

- Ziel: Signaturen, die viele Nachrichten signieren können
- Vorstufe: Signaturen, die nur **eine** Nachricht **sicher** signieren können (**Einmalsignaturen**)
- für jeden *public key* sollte nur eine einzige Signatur ausgestellt werden, sonst evtl. unsicher

1.7.1 Sicherheitsbegriffe für Einmalsignaturen

Analog zum vorherigen Kapitel definieren wir **EUF-1-CMA** und **EUF-1-naCMA** für Einmalsignaturen.

1.7.2 Beziehungen zwischen Sicherheitsdefinitionen

$$\begin{array}{ccc}\text{EUF-naCMA} & \Leftarrow & \text{EUF-CMA}\\ \Downarrow & & \Downarrow\\ \text{EUF-1-naCMA} & \Leftarrow & \text{EUF-1-CMA}\end{array}$$

Beweis im Skript.

mail@nilslambertz.de
nilslambertz

1.8 Perfekte Sicherheit

In den Definitionen, z.B. bei EUF-CMA finden sich zwei Einschränkungen, die im folgenden erläutert werden:

1.8.1 Warum müssen wir uns auf PPT-Angreifer beschränken?

Durch Brute-Force könnte ein unbeschränkter Angreifer alle Signaturen durchprobieren und so valide Signaturen für beliebige Nachrichten finden, wodurch er beim Sicherheitsexperiment immer gewinnen würde.

1.8.2 Warum muss die Erfolgswahrscheinlichkeit des Angreifers nur vernachlässigbar sein?

Die Erfolgswahrscheinlichkeit kann nicht 0 sein, da der Angreifer durch zufälliges Raten eine gültige Signatur für eine beliebige Nachricht finden könnte, wodurch er das Sicherheitsexperiment gewinnt.

1.9 Erweiterung des Nachrichtenraumes

Wir konstruieren fast immer Signaturen mit "kleinem" Nachrichtenraum, z.B.

- $\mathbb{Z}_p = \{0, \dots, p-1\}$, p prim
- $\{0,1\}^{q(k)}$, q Polynom, k Sicherheitsparameter

Unser Ziel ist es jedoch, beliebige Nachrichten, z.B. $\{0,1\}^*$, zu signieren.

1.9.1 Hashfunktionen

Eine kryptographische Hashfunktion $H = (Gen_H, Eval_H)$ ist ein Tupel aus zwei PPT-Algorithmen:

- $Gen_H(1^k)$ berechnet t, sodass t eine Funktion

$$H_t : \{0,1\}^* \to \mathcal{M}_t$$

  spezifiziert
- $Eval_H(1^k, t, x)$ berechnet $H_t(x)$

1.9.2 Kollisionsresistenz

Eine Hashfunktion $H = (Gen_H, Eval_H)$ ist **kollisionsresistent**, falls für alle $t \leftarrow Gen_H(1^k)$ und für alle PPT $\mathcal{A}$ gilt, dass

$$\Pr[\mathcal{A}(1^k, t) = (x, x') : H_t(x) = H_t(x') \wedge x \neq x'] \leq negl(k)$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

1.9.3 Signatur mit unbeschränktem Nachrichtenraum (Hash-then-Sign)

Wir wollen nun Signaturen mit unbeschränktem Nachrichtenraum konstruieren. Gegeben:

- $\Sigma' = (Gen', Sign', Vfy')$ mit Nachrichtenraum $\mathcal{M}$
- kollisionsresistente Hashfunktion $H : \{0,1\}^* \to \mathcal{M}$

Konstruiere $\Sigma = (Gen, Sign, Vfy)$ mit Nachrichtenraum $\{0,1\}^*$:

- $Gen(1^k)$ berechnet $(pk, sk) \leftarrow Gen'(1^k)$
- $Sign(sk, m)$ berechnet $\sigma \leftarrow Sign'(sk, H(m))$
- $Vfy(pk, m, \sigma)$ gibt $Vfy'(pk, H(m), \sigma)$ aus

mail@nilslambertz.de
nilslambertz

2 q-mal Signaturen

2.1 Von EUF-naCMA-Sicherheit zu EUF-CMA-Sicherheit

Gegeben

- ein EUF-naCMA-sicheres Signaturverfahren Σ' und
- ein EUF-1-naCMA-sicheres Einmalsignaturverfahren $\Sigma^{(1)}$

können wir mittels **Transformation** ein **EUF-CMA**-sicheres Signaturverfahren Σ konstruieren.

2.1.1 Transformation

Gegeben:

- EUF-naCMA-sicheres Signaturverfahren $\Sigma' = (Gen', Sign', Vfy')$
- EUF-1-naCMA-sicheres Signaturverfahren $\Sigma^{(1)} = (Gen^{(1)}, Sign^{(1)}, Vfy^{(1)})$

Konstruiere nun $\Sigma = (Gen, Sign, Vfy)$ wie folgt:

- $Gen(1^k)$:

$$(pk, sk) := (pk', sk') \leftarrow Gen'(1^k)$$

- $Sign(sk, m)$:

$$\begin{aligned}(pk^{(1)}, sk^{(1)}) &\leftarrow Gen^{(1)}(1^k)\\ \sigma' &\leftarrow Sign'(sk, pk^{(1)})\\ \sigma^{(1)} &\leftarrow Sign^{(1)}(sk^{(1)}, m)\\ \sigma &:= (pk^{(1)}, \sigma^{(1)}, \sigma')\end{aligned}$$

- $Vfy(pk, m, \sigma)$ gibt 1 aus, wenn

$$Vfy'(pk, pk^{(1)}, \sigma') = 1 \land Vfy^{(1)}(pk^{(1)}, m, \sigma^{(1)}) = 1$$

  sonst 0

Es wird also für jede Signatur ein neues Einmalschlüsselpaar erzeugt.

2.2 Mehrmal-Signaturverfahren aus Einmalsignaturverfahren

Einmalsignaturverfahren sind effizient und einfach zu konstruieren, daher würden wir gerne eine Variation dieser verwenden, um mehrfach signieren zu können (q-mal-Signaturverfahren).

2.2.1 Naiver Ansatz: q Schlüsselpaare

Der naive Ansatz ist, q Schlüsselpaare zu verwenden und einen Zähler $st \in \{1, \dots, q\}$ als Zustand zu verwenden, der auch im Secret Key und in der Signatur vorkommt:

- $Gen(1^k)$:

$$\begin{aligned}&(pk_i, sk_i) \leftarrow Gen^{(1)}(1^k) \text{ für alle } i \in \{1, \dots, q\}\\ &pk := (pk_1, \dots, pk_q)\\ &sk := (sk_1, \dots, sk_q, st = 1)\end{aligned}$$

- $Sign(sk, m)$:

$$i := st$$
$$\sigma_i \leftarrow Sign^{(1)}(sk_i, m)$$
$$\sigma \leftarrow (\sigma_i, i)$$
$$st := st + 1$$

- $Vfy(pk, m, \sigma = (\sigma_i, i))$:

$$Vfy^{(1)}(pk_i, m, \sigma_i) \stackrel{?}{=} 1$$

Eigenschaften bezogen auf Signaturanzahl (q):

- $|pk| \in \Theta(q)$
- $|sk| \in \Theta(q)$
- $|\sigma| \in \Theta(1)$

2.2.2 Zwischenschritt: Hashfunktion verwenden

Ein weiterer möglicher Ansatz ist die verwendung einer Hashfunktion

- H Hashfunktion
- $Gen(1^k)$:

$$(pk_i, sk_i) \leftarrow Gen^{(1)}(1^k) \text{ für alle } i \in \{1, \ldots, q\}$$
$$pk := H(pk_1, \ldots, pk_q)$$
$$sk := (sk_1, \ldots, sk_q, pk_1, \ldots, pk_q, st = 1)$$

- $Sign(sk, m)$:

$$i := st$$
$$\sigma_i \leftarrow Sign^{(1)}(sk_i, m)$$
$$\sigma \leftarrow (\sigma_i, i, pk_1, \ldots, pk_q)$$
$$st := st + 1$$

- $Vfy(pk, m, \sigma)$:

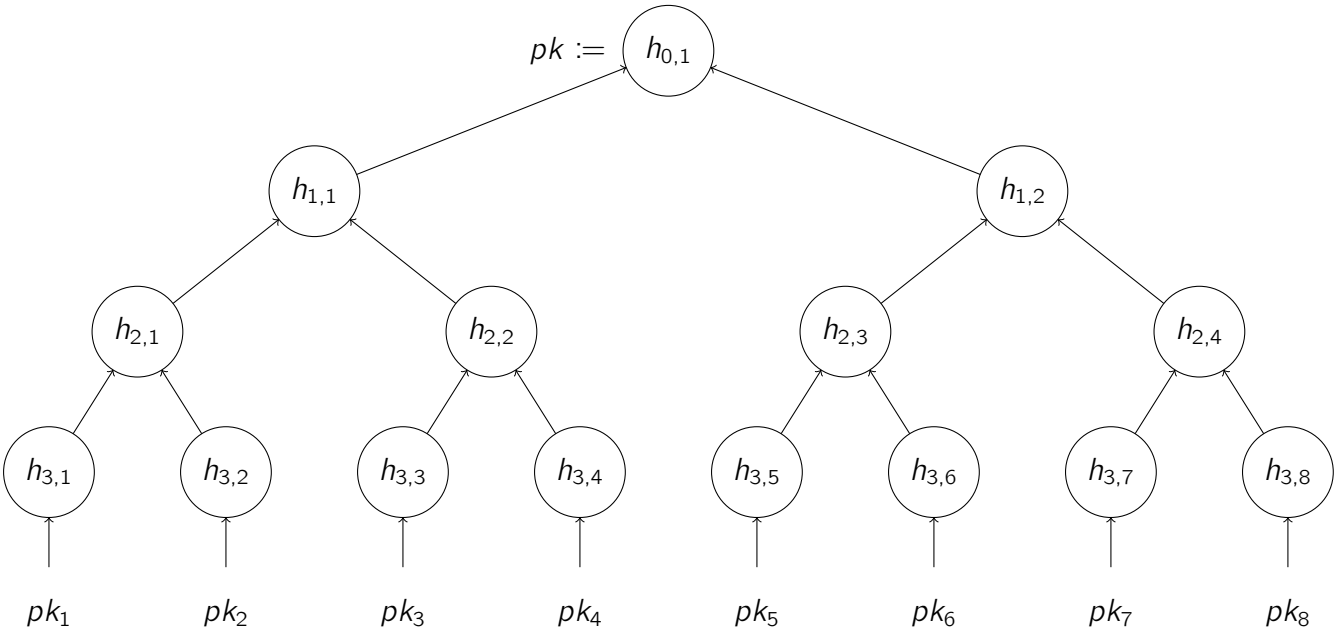
$$Vfy^{(1)}(pk_i, m, \sigma_i) \stackrel{?}{=} 1 \text{ und } H(pk_1, \ldots, pk_q) \stackrel{?}{=} pk$$

Eigenschaften bezogen auf Signaturanzahl (q):

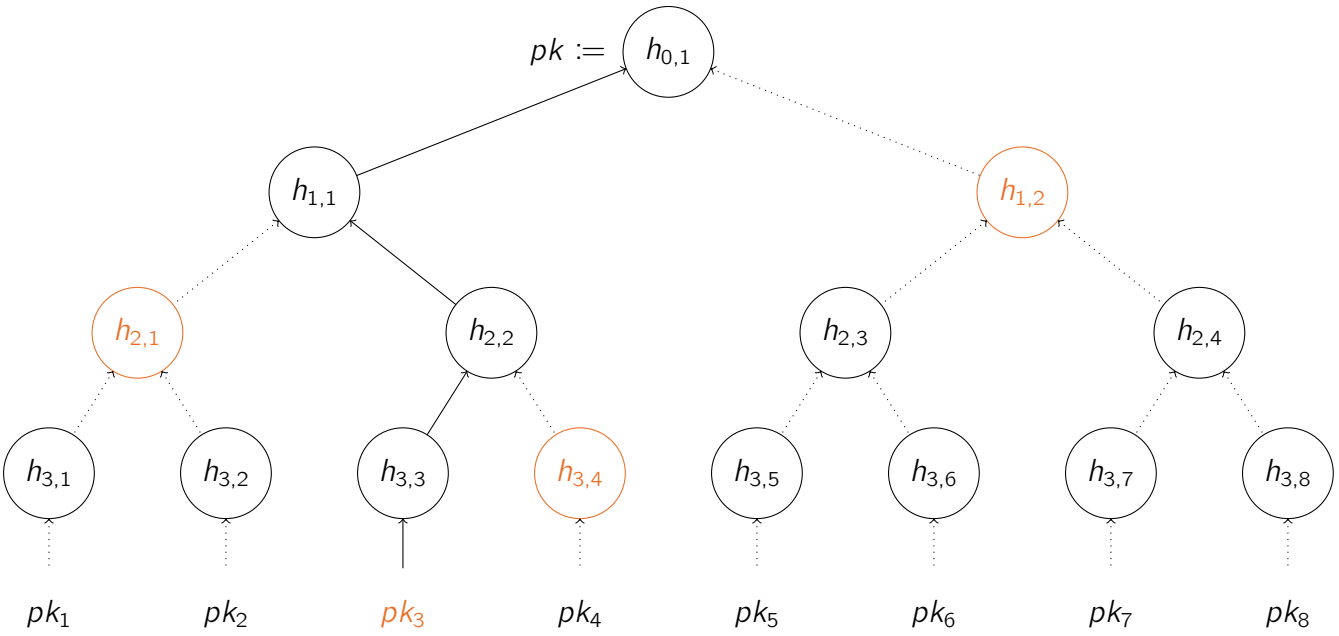
- $|pk| \in \Theta(1)$
- $|sk| \in \Theta(q)$
- $|\sigma| \in \Theta(q)$

2.2.3 Merkle-Bäume

Merkle-Bäume (auch **Hash-Bäume** genannt) sind (meist binäre) Bäume, bei denen die Blätter Hashwerte der Daten sind und jeder Knoten darüber aus Hashwerten seiner Kinder besteht:



Der Co-Pfad eines Knotens v in einem Binärbaum mit Wurzel r ist die Folge aller Knoten $u_1, \ldots, u_n$ wobei u_i der Geschwisterknoten des i-ten Knotens auf dem Pfad von v zu r ist:



Der Co-Pfad wird nun in die Signatur hinzugefügt, wodurch der pk von pk_3 ausgehend (in diesem Beispiel) in Vfy berechnet werden kann.

- $Gen(1^k)$:

$$(pk_i, sk_i) \leftarrow Gen^{(1)}(1^k) \text{ für alle } i \in \{1, \ldots, q\}$$
$$pk := \text{Baum-Hash}(pk_1, \ldots, pk_q)$$
$$sk := (sk_1, \ldots, sk_q, pk_1, \ldots, pk_q, st = 1)$$

mail@nilslambertz.de
nilslambertz

- $Sign(sk, m)$:

$$
\begin{aligned}
&i \coloneqq st \\
&\sigma_i \leftarrow Sign^{(1)}(sk_i, m) \\
&\sigma \leftarrow (\sigma_i, i, pk_i, \text{Co-Pfad}) \\
&st \coloneqq st + 1
\end{aligned}
$$

- $Vfy(pk, m, \sigma)$:

$$
\begin{aligned}
&\text{Berechne Wurzel } h' \\
&Vfy^{(1)}(pk_i, m, \sigma_i) \stackrel{?}{=} 1 \text{ und } h' \stackrel{?}{=} pk
\end{aligned}
$$

Eigenschaften bezogen auf Signaturanzahl (q):

- $|pk| \in \Theta(1)$
- $|sk| \in \Theta(q)$
- $|\sigma| \in \Theta(\log q)$

Lemma:

Wenn $\Sigma^{(1)}$ EUF-1-naCMA-sicher ist und H kollisionsresistent, dann ist das obige Verfahren EUF-q-naCMA-sicher.

Wenn $\Sigma^{(1)}$ EUF-1-CMA-sicher ist und H kollisionsresistent, dann ist das obige Verfahren EUF-q-CMA-sicher.

2.3 Komprimieren des geheimen Schlüssels

Um den geheimen Schlüssel zu komprimieren, verwenden wir statt echtem Zufall *Pseudozufall* zur Generierung.

2.3.1 Pseudozufallsfunktion

Eine Pseudozufallsfunktion oder Pseudorandom function (**PRF**) ist eine Funktion, die ununterscheidbar von einer zufälligen Funktion ist. Sie erhält dafür zusätzlich einen *Seed* s mit Länge k als Eingabe:

$$
\begin{array}{ccccc}
\text{PRF: } \{0,1\}^k & \times & \{0,1\}^n & \rightarrow & \{0,1\}^l \\
\uparrow & & \uparrow & & \uparrow \\
\text{Seed } s & & \alpha & & PRF(s, \alpha)
\end{array}
$$

2.3.2 Schlüsselgenerierung

Bisher wird unser Schlüssel durch einen *probabilistischen* Algorithmus erzeugt:

$$(pk_i, sk_i) \leftarrow Gen^{(1)}(1^k) \qquad \forall i \in \{1, \ldots, q\}$$

Probabilistische Algorithmen können auch als **deterministische Algorithmen mit Zufall** r **als Eingabe** gesehen werden:

$$Gen^{(1)}(1^k) \qquad \hat{=} \qquad Gen^{(1)}_{\text{det}}(1^k, r)$$

mail@nilslambertz.de
nilslambertz

Damit ist die bishere Schlüsselberechnung äquivalent zu folgender:

$$(pk_i, sk_i) := Gen_{\text{det}}^{(1)}(1^k, r_i) \qquad \forall i \in \{1, \dots, q\} \qquad \text{für echt zufällige } r_i$$

Mit **echt zufälliger** Funktion F also:

$$(pk_i, sk_i) := Gen_{\text{det}}^{(1)}(1^k, F(i)) \qquad \forall i \in \{1, \dots, q\}$$

Mit einem zufälligen Seed s können wir den echten Zufall durch Pseudozufall ersetzen:

$$(pk_i, sk_i) := Gen_{\text{det}}^{(1)}(1^k, PRF(s, i)) \qquad \forall i \in \{1, \dots, q\} \qquad \text{für } s \xleftarrow{\$} \{0,1\}^k$$

Dadurch müssen nur noch der Seed s und der Zähler st im Secret Key gespeichert werden, bei Bedarf können die Schlüsselpaare neu berechnet werden:

$$sk = (s, st)$$

Eigenschaften bezogen auf Signaturanzahl (q):

- $|pk| \in \Theta(1)$
- $|sk| \in \Theta(1)$
- $|\sigma| \in \Theta(\log q)$

3 Chamäleon-Signaturen

Motivation: Wir wollen Signaturen der Form, dass A die Signatur von B verifizieren kann, jedoch einen anderen C nicht davon überzeugen kann, dass die Signatur von B kam (**Abstreitbarkeit** genannt).

3.1 Chamäleon-Hashfunktionen

3.1.1 Definition

Eine **Chamäleon-Hashfunktion** CH besteht aus zwei PPT-Algorithmen $(Gen_{ch}, TrapColl_{ch})$:

- $Gen_{ch}(1^k)$ gibt (ch, τ) aus:
  - ch ist eine Funktion $ch : \mathcal{M} \times \mathcal{R} \to \mathcal{N}$
    - $\mathcal{M}$ Nachrichtenraum
    - $\mathcal{R}$ Zufallsraum
    - $\mathcal{N}$ Zielraum
  - τ ist eine **Trapdoor** (Falltür)
- $TrapColl_{ch}(\tau, m, r, m')$ für $(m, r, m') \in \mathcal{M} \times \mathcal{R} \times \mathcal{M}$ berechnet $r' \in \mathcal{R}$, sodass

$$ch(m, r) = ch(m', r')$$

Wer τ kennt, kann also Kollisionen berechnen.

3.1.2 Kollisionsresistenz

Eine Chamäleon-Hashfunktion $CH = (Gen_{ch}, TrapColl_{ch})$ ist **kollisionsresistent**, falls für alle PPT $\mathcal{A}$ gilt, dass

$$\Pr\left[\begin{matrix}(ch, \tau) \leftarrow Gen_{ch}(1^k) \\ \mathcal{A}(1^k, ch) = (m, r, m', r')\end{matrix} : \begin{matrix}ch(m, r) = ch(m', r') \\ \wedge (m, r) \neq (m', r')\end{matrix}\right] \leq negl(k)$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

mail@nilslambertz.de
nilslambertz

3.1.3 DLog-Annahme

Setting:

- Zyklische Gruppe $\mathbb{G} = \langle g \rangle$
- Endliche Ordnung $|\mathbb{G}| = p$, p prim
- ($\mathbb{G}$ kommutativ)
- $\mathbb{G}$ abhängig vom Sicherheitsparameter k

Das **DLog-Problem** ist wie folgt definiert:

- Gegeben: Erzeuger g und $y \xleftarrow{\$} \mathbb{G}$
- Finde $x \in \mathbb{Z}_p : g^x = y$

Die **DLog-Annahme** ist folgende:

$\forall$ PPT $\mathcal{A}$ gilt:

$$\Pr[\mathcal{A}(1^k, g, g^x) = x : \langle g \rangle = \mathbb{G} \text{ zufällig}, x \xleftarrow{\$} \mathbb{Z}_p] \leq negl(k)$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

3.1.4 Chamäleon-Hashfunktion basierend auf DLog

Wir konstruieren nun eine Chamäleon-Hashfunktion basierend auf DLog mit einer Gruppe $\mathbb{G}$, $|\mathbb{G}| = p$ prim und g Erzeuger von $\mathbb{G}$:

- $Gen(1^k)$:
  - $x \xleftarrow{\$} \mathbb{Z}_p$
  - $h := g^x$
  - $ch := (g, h)$
  - $\tau := x$
- ch beschreibt Funktion:
  - $ch : \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{G}$
  - $ch(m, r) := g^m \cdot h^r$
- $TrapColl_{ch}(\tau, m, r, m')$: Berechnet r^*, sodass

$$\begin{aligned} & m + x \cdot r \equiv m^* + x \cdot r^* \mod p \\ \Leftrightarrow \quad & r^* \equiv \frac{m - m^*}{x} + r \mod p \end{aligned}$$

Damit folgt

$$ch(m, r) = g^m \cdot h^r = g^{m+xr} = g^{m^*+xr^*} = g^{m^*} \cdot h^{r^*} = ch(m^*, r^*)$$

Chamäleon-Hashfunktion basierend auf dem RSA-Problem und Shamir's Trick weggelassen.

mail@nilslambertz.de
nilslambertz

3.2 Chamäleon-Signaturen

3.2.1 Konstruktion

Gegeben sind

- $CH = (Gen_{ch}, TrapColl_{ch})$, $ch : \mathcal{M} \times \mathcal{R} \to \mathcal{N}$
- Signaturverfahren $\Sigma' = (Gen', Sign', Vfy')$

Konstruiere nun ein Chamäleon-Signaturverfahren $\Sigma = (Gen, Sign, Vfy)$:

- $Gen(1^k)$:

$$(pk', sk') \leftarrow Gen'(1^k)$$
$$pk := pk'$$
$$sk := sk'$$

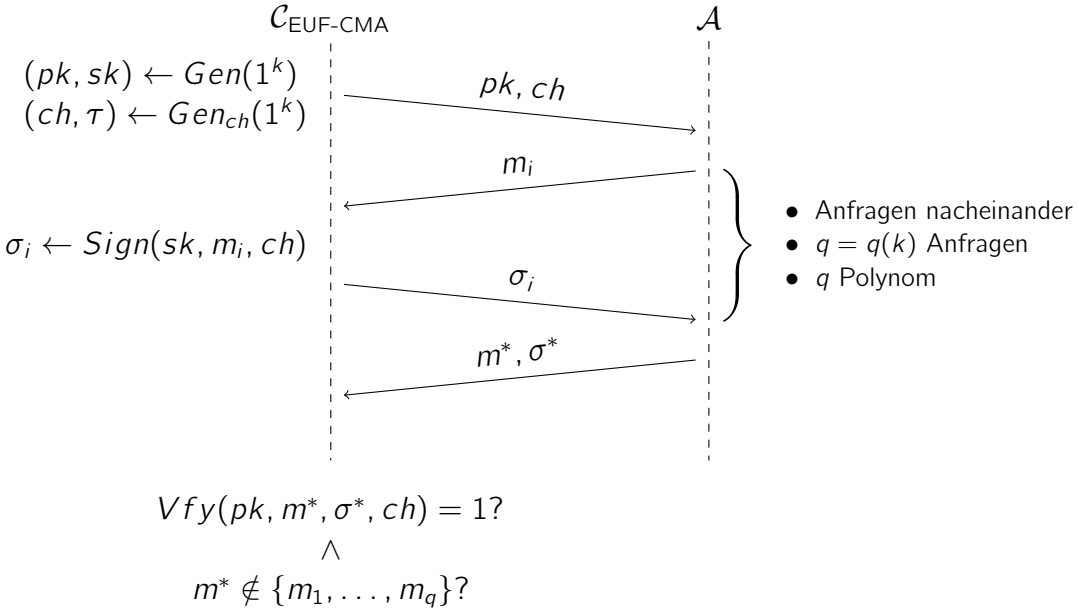
- $Sign(sk, m, ch)$ (ch ist die CH-Fkt. des **Empfängers**):

$$r \xleftarrow{\$} \mathcal{R}$$
$$y := ch(m, r)$$
$$\sigma' := Sign'(sk, y)$$
$$\sigma := (\sigma', r)$$

- $Vfy(pk, m, \sigma, ch)$:

$$Vfy'(pk, ch(m, r), \sigma') \stackrel{?}{=} 1$$

3.2.2 Visualisierung: EUF-CMA-Sicherheitsexperiment



$\mathcal{A}$ gewinnt, falls $Vfy(pk, m^*, \sigma^*, ch) = 1$ **und** $m^* \notin \{m_1, \ldots, m_q\}$

In dieser Variante wird ch vorgegeben, stärkere Sicherheit wird erreicht, wenn $\mathcal{A}$ die Chamäleon-Hashfunktion selbst wählen darf (wie es ein echter Angreifer könnte). *Beweis zur Sicherheit im Skript.*

3.3 Transformation von Chamäleon-Hashfunktion zu Einmalsignatur

Jede CH kann zu einem Einmalsignaturverfahren transformiert werden.

Gegeben $CH = (Gen_{ch}, TrapColl_{ch})$, konstruiere $\Sigma = (Gen, Sign, Vfy)$:

- $Gen(1^k)$:

$$
\begin{aligned}
&(ch, \tau) \leftarrow Gen_{ch}(1^k)\\
&(\tilde{m}, \tilde{r}) \xleftarrow{\$} \mathcal{M} \times \mathcal{R}\\
&c := ch(\tilde{m}, \tilde{r})\\
&pk := (ch, c)\\
&sk := (\tau, \tilde{m}, \tilde{r})
\end{aligned}
$$

- $Sign(sk, m)$:

$$
\begin{aligned}
&r := TrapColl_{ch}(\tau, \tilde{m}, \tilde{r}, m)\\
&\sigma := r
\end{aligned}
$$

- $Vfy(pk, m, \sigma)$:

$$c \stackrel{?}{=} ch(m, \sigma)$$

Σ ist EUF-1-naCMA-sicher, wenn CH kollisionsresistent ist.

Dlog-Einmalsignatur aus DLog-CH-Funktion weggelassen.

3.4 EUF-CMA verstärken

Statt wie bisher in EUF-CMA $m^* \notin \{m_1, \ldots, m_q\}$ zu fordern, könnten wir auch fordern, dann nur das Paar (m^*, σ^*) frisch sein muss, die Nachricht aber nicht unbedingt.

3.4.1 Definition: sEUF-CMA

Ein digitales Signaturverfahren $\Sigma = (Gen, Sign, Vfy)$ ist *sEUF-CMA-sicher*, wenn für alle PPT $\mathcal{A}$ gilt, dass

$$\Pr\left[\mathcal{A}^{\mathcal{C}_{\text{sEUF-CMA}}}(pk) = (m^*, \sigma^*) : \begin{matrix} Vfy(pk, m^*, \sigma^*) = 1 \quad \wedge \\ (m^*, \sigma^*) \notin \{(m_1, \sigma_1), \ldots, (m_q, \sigma_q)\} \end{matrix}\right] \leq negl(k)$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

Mit einem EUF-CMA-sicheren Signaturverfahren und einer CH-Funktion kann ein sEUF-CMA-sicheres Signaturverfahren konstruiert werden. *Details im Skript.*

4 Pairings und BLS-Signaturen

4.1 Pairings

4.1.1 Definition

Seien $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ zyklische Gruppen mit Ordnung p prim. Ein **Pairing** ist eine **bilineare** Abbildung

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

mit den Eigenschaften

- **Bilinearität**: $\forall g_1, g_1' \in \mathbb{G}_1, g_2, g_2' \in \mathbb{G}_2$:

$$e(g_1 \cdot g_1', g_2) = e(g_1, g_2) \cdot e(g_1', g_2)$$
$$e(g_1, g_2 \cdot g_2') = e(g_1, g_2) \cdot e(g_1, g_2')$$

  $\Rightarrow e(g_1^a, g_2) = e(g_1, g_2)^a = e(g_1, g_2^a)$
- **Nicht-Ausgeartetheit** (*non-degenerate*): Für Erzeuger $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ gilt:

$$e(g_1, g_2) \text{ ist Erzeuger von } \mathbb{G}_T \qquad (\overset{|\mathbb{G}_T| \text{ prim}}{\iff} e(g_1, g_2) \neq 1)$$

- Effiziente Berechenbarkeit

$\mathbb{G}_1, \mathbb{G}_2$ sind in der Regel **elliptische Kurven**.

4.1.2 Typen von Pairing

1. $\mathbb{G}_1 = \mathbb{G}_2$, **symmetrisches Pairing**

$$e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$$

2. $\mathbb{G}_1 \neq \mathbb{G}_2$, **asymmetrisches Pairing** und es existiert ein effizienter, nicht-trivialer Homomorphismus

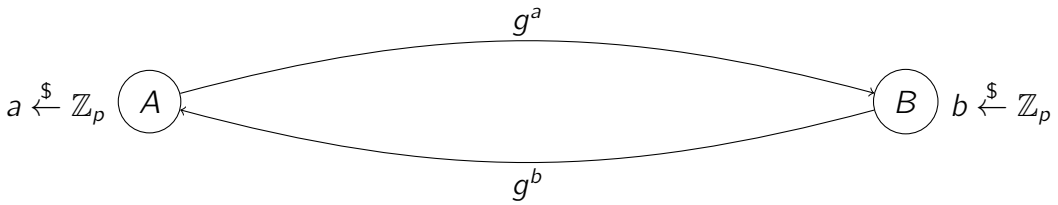
$$\psi : \mathbb{G}_1 \to \mathbb{G}_2$$

3. $\mathbb{G}_1 \neq \mathbb{G}_2$, **asymmetrisches Pairing** und es existiert *kein* effizienter, nicht-trivialer Homomorphismus

$$\psi : \mathbb{G}_1 \to \mathbb{G}_2$$

4.1.3 Diffie-Hellman-Schlüsselaustausch

Diffie-Hellman ist ein Protokoll, mit dem **zwei** Parteien einen gemeinsamen geheimen Schlüssel aushandeln können. Setting: Zyklische Gruppe $\mathbb{G} = \langle g \rangle$ mit Ordnung p



$k = (g^b)^a = g^{ab}$ $\qquad$ $k = (g^a)^b = g^{ab}$

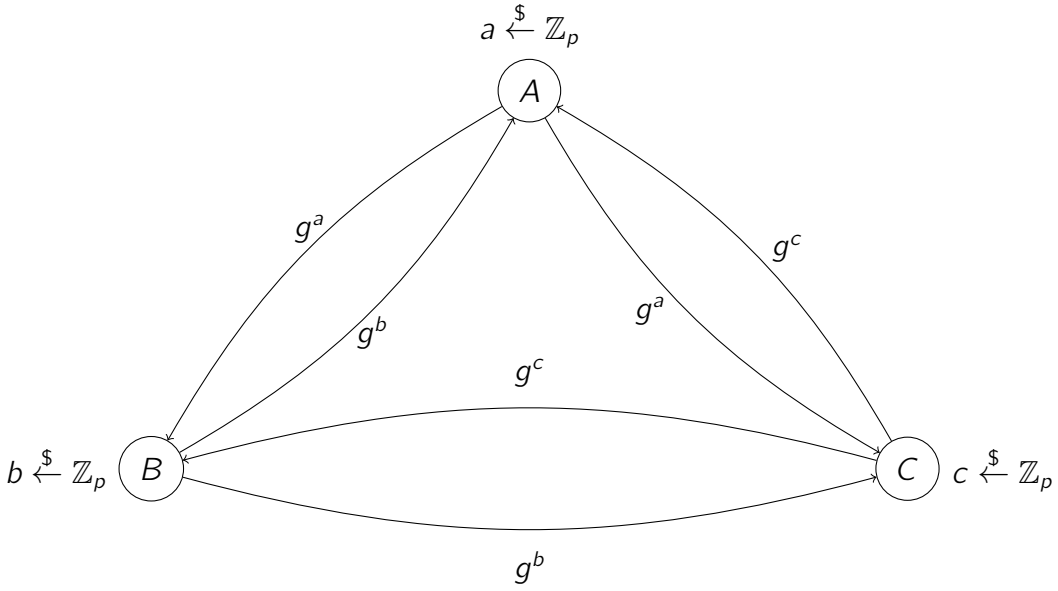
Ablauf:

1. A und B wählen ein zufälliges Element aus $\mathbb{Z}_p$
2. A und B senden dem Gegenüber g^a bzw. g^b
3. Beide können sich nun den gemeinsamen Schlüssel $k = g^{ab}$ berechnen

4.1.4 Joux 3-Parteien-Schlüsselaustausch

Joux' Verfahren [Joux, 2006] ist ähnlich zu Diffie-Hellman, erlaubt aber einen Schlüsselaustausch zwischen 3 Parteien.

$$k = e(g^b, g^c)^a = e(g, g)^{abc}$$



$k = e(g^a, g^c)^b = e(g, g)^{abc}$ $\qquad$ $k = e(g^a, g^b)^c = e(g, g)^{abc}$

Ablauf:

1. A, B und C wählen ein zufälliges Element aus $\mathbb{Z}_p$
2. Alle Teilnehmer senden sich gegenseitig ihre Werte g^a, g^b bzw. g^c
3. Alle Teilnehmer können sich nun den gemeinsamen Schlüssel $k = e(g, g)^{abc}$ berechnen

4.2 Boneh-Lynn-Shacham-Signaturen

BLS ist ein Pairing-basiertes Signaturverfahren mit kurzen Signaturen. Gegeben:

- $\mathbb{G}, \mathbb{G}_T$ Gruppen, $|\mathbb{G}| = |\mathbb{G}_T| = p$ prim, $\langle g \rangle = \mathbb{G}$
- Symmetrisches Pairing $\mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$
- Hashfunktion $H : \{0,1\}^* \to \mathbb{G}$

Konstruktion:

- $Gen(1^k)$:

$$x \xleftarrow{\$} \mathbb{Z}_p$$
$$pk := (g, g^x)$$
$$sk := x$$

- $Sign(sk, m)$:

$$\sigma := H(m)^x \in \mathbb{G}$$

- $Vfy(pk, m, \sigma)$:

$$e(H(m), g^x) \stackrel{?}{=} e(\sigma, g)$$

Correctness: $e(H(m), g^x) = e(H(m), g)^x = e(H(m)^x, g) = e(\sigma, g)$

BLS-Signaturen sind EUF-CMA-sicher unter der **CDH-Annahme** im **Random-Oracle-Modell**.

Sicherheitsbeweis für BLS weggelassen.

4.2.1 Aggregierbarkeit

BLS-Signaturen können **aggregiert** werden, d.h. zur Verifikation von $(m_1, \sigma_1), \ldots, (m_n, \sigma_n)$ müssen nicht alle Signaturen $\sigma_1, \ldots, \sigma_n$ mitgeschickt werden, sondern es kann eine aggregierte Signatur σ_{Agg} berechnet werden. Die Gültigkeit kann dann mit $Vfy(pk_1, \ldots, pk_n, m_1, \ldots, m_n, \sigma_{\text{Agg}}) \stackrel{?}{=} 1$ überprüft werden.

Dabei gilt außerdem, dass die aggregierte Signatur genau so lang ist, wie eine einzelne Signatur, also $|\sigma_{\text{Agg}}| = |\sigma|$. Zudem bieten sie einen Effizienzgewinn, da für n Signaturen statt $2n$ nur noch $n+1$ Pairingauswertungen erforderlich sind.

Aggregation:

- Signaturen haben die Form $H(m_i)^{x_i}$
- Aggregierer berechnet

$$\sigma_{\text{Agg}} = \prod_{i=1}^{n} \sigma_i$$

  Aggregation ist öffentlich, es wird kein geheimer Schlüssel benötigt
- Verifikation:

$$e(\sigma_{\text{Agg}}, g) \stackrel{?}{=} \prod_{i=1}^{n} e(H(m_i), g^{x_i})$$

- Correctness:

$$\begin{aligned} e(\sigma_{\text{Agg}}, g) &= e(\sigma_1, g) \cdot \ldots \cdot e(\sigma_n, g) \\ &= e(H(m_1)^{x_1}, g) \cdot \ldots \cdot e(H(m_n)^{x_n}, g) \\ &= e(H(m_1), g^{x_1}) \cdot \ldots \cdot e(H(m_n), g^{x_n}) \\ &= \prod_{i=1}^{n} e(H(m_i), g^{x_i}) \end{aligned}$$

4.2.2 Batch-Verifikation

Ein ähnliches Problem tritt bei der Verifikation auf, bisher verifizieren wir Nachrichten immer einzeln über $Vfy(pk_i, m_i, \sigma_i) \stackrel{?}{=} 1$.

Optimaler wäre ein Verfahren, mit dem wir $(m_1, \sigma_1), \ldots, (m_n, \sigma_n)$ auf einmal verifizieren können, die Lösung dafür ist die **Batch-Verifikation**:

- Gegeben: $\sigma_1, \ldots, \sigma_n$ Signaturen für $m_1, \ldots, m_n$ Nachrichten
- $h = \prod_{i=1}^{n} H(m_i)$
- $\sigma = \prod_{i=1}^{n} \sigma_i$
- Prüfe, ob $e(\sigma, g) \stackrel{?}{=} e(h, g^x)$

mail@nilslambertz.de
nilslambertz

4.3 Computational-Diffie-Hellman-Problem

4.3.1 CDH-Problem

Sei g ein zufälliger Erzeuger und $x, y \xleftarrow{\$} \mathbb{Z}_p$.

CDH-Problem: Gegeben (g, g^x, g^y), berechne g^{xy}

4.3.2 CDH-Annahme

$\forall$ PPT $\mathcal{A}$ gilt:

$$\Pr[\mathcal{A}(1^k, g, g^x, g^y) = g^{xy} : g \text{ mit } \langle g \rangle = \mathbb{G} \text{ zufällig}, x, y \xleftarrow{\$} \mathbb{Z}_p] \leq negl(k)$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

4.4 Random-Oracle-Modell (ROM)

Wir betrachten eine idealisierte Hashfunktion $H : \mathcal{D} \to \mathcal{R}$, bei der die Ausgaben $H(m)$ **gleichverteilt zufällig** sind für jede Eingabe m. H wird als Orakel modelliert, das von allen Teilnehmern benutzt wird und die Hashwerte ausgibt.

4.4.1 Das H-Orakel

Das **H-Orakel** besitzt intern einen *Key-Value-Store* T. Falls es eine Hash-Anfrage für Nachricht m erhält, schaut es in T nach, ob $T[m]$ bereits existiert. Wenn ja, wird $T[m]$ zurückgegeben, ansonsten wählt das Orakel $y \xleftarrow{\$} \mathcal{R}$, setzt $T[m] := y$ und gibt y zurück.

4.4.2 Diskussion zum ROM

- Es existiert kein ROM in der realen Welt
- Manche kryptographischen Probleme sind **nur** im ROM lösbar
- Lösungen im ROM sind oft effizienter und einfacher zu konstruieren als im Standardmodell
- Für viele Konstruktionen im ROM sind keine realen Angriffe bekannt

5 Waters-Signaturen

5.1 Programmierbare Hashfunktionen

5.1.1 Definition

Es sei $H_\kappa : \{0,1\}^\ell \to \mathbb{G}$ eine Hashfunktion und $\mathbb{G}$ eine zyklische, endliche Gruppe mit g, h Erzeuger.

Eine **Programmierbare Hashfunktion** (PHF) ist ein Tupel von 4 (P)PT-Algorithmen

- $Gen(g) \to \kappa$ (Schlüsselerzeugung)
- $Eval(\kappa, m) \to H_\kappa(m)$ (deterministische Auswertung)
- $TrapGen(g, h) \to (\kappa, \tau)$ (Schlüsselerzeugung mit Trapdoor)
- $TrapEval(\tau, m) \to (a, b)$ mit $h^a g^b = H_\kappa(m)$ (deterministisch)

Intuition: Trapdoor liefert uns "Zerlegung" (a, b) von $H_\kappa(m)$, sodass $h^a g^b = H_\kappa(m)$

5.1.2 Anforderungen an PHF

- κ ist für Gen und $TrapGen$ gleichverteilt, d.h. es ist unmöglich unterscheiden, mit welchem Algorithmus es erstellt wurde
- (v, w, γ)-**Wohlverteilung**: Seien $v, w \in \mathbb{N}, \gamma \in [0, 1]$. Für alle
  - Erzeuger g, h von $\mathbb{G}$
  - $m_1^*, \ldots, m_v^* \in \{0,1\}^\ell$
  - $m_1, \ldots, m_w \in \{0,1\}^\ell$ (alle m_i^* und m_j paarweise verschieden)

  gilt

$$\Pr\begin{bmatrix} a_i^* = 0 & \forall i = 1, \ldots, v & \wedge \\ a_j \neq 0 & \forall j = 1, \ldots, w & \end{bmatrix} \geq \gamma$$

  wobei

$$\begin{aligned} (\kappa, \tau) &\leftarrow TrapGen(g, h) \\ (a_i^*, b_i^*) &:= TrapEval(\tau, m_i^*) \quad \forall i = 1, \ldots, v \\ (a_j, b_j) &:= TrapEval(\tau, m_j) \quad \forall j = 1, \ldots, w \end{aligned}$$

  Eine (v, w, γ)-wohlverteilte PHF heißt auch (v, w, γ)-**PHF**.

5.1.3 Waters Programmierbare Hashfunktion

- $Gen(g)$:

$$\begin{aligned} (u_0, \ldots, u_\ell) &\xleftarrow{\$} \mathbb{G} \\ \kappa &= (u_0, \ldots, u_\ell) \end{aligned}$$

- $Eval(\kappa, m = m_1 \ldots m_\ell)$:

$$H_\kappa(m) = u_0 \prod_{i=1}^{\ell} u_i^{m_i}$$

  ($m_i \in \{0,1\}$ ist das i-te Bit von m)
  Intuition: $H_\kappa(m)$ ist das Produkt von u_0 und aller u_i mit $m_i = 1$
- $TrapGen(g, h)$:

$$\begin{aligned} \hat{a}_i &\xleftarrow{\$} \{-1, 0, 1\} \in \mathbb{Z}_p \\ \hat{b}_i &\xleftarrow{\$} \mathbb{Z}_p \\ u_i &:= h^{\hat{a}_i} g^{\hat{b}_i} \qquad \forall i \in \{0, \ldots, \ell\} \\ \kappa &:= (u_0, \ldots, u_\ell) \\ \tau &:= (\hat{a}_0, \ldots, \hat{a}_\ell, \hat{b}_0, \ldots, \hat{b}_\ell) \end{aligned}$$

mail@nilslambertz.de
nilslambertz

- $TrapEval(\tau, m = m_1 \dots m_\ell)$: Berechne

$$a = \hat{a_0} + \sum_{i=1}^{\ell} m_i \hat{a_i} \qquad \text{und}$$
$$b = \hat{b_0} + \sum_{i=1}^{\ell} m_i \hat{b_i}$$

Dann gilt

$$\begin{aligned} h^a g^b &= h^{\hat{a_0}} \prod_{i=1}^{\ell} h^{\hat{a_i} m_i} \cdot g^{\hat{b_0}} \prod_{i=1}^{\ell} g^{\hat{b_i} m_i} \\ &= (h^{\hat{a_0}} g^{\hat{b_0}}) \cdot \prod_{i=1}^{\ell} (h^{\hat{a_i} m_i} g^{\hat{b_i} m_i}) \\ &= (h^{\hat{a_0}} g^{\hat{b_0}}) \cdot \prod_{i=1}^{\ell} (h^{\hat{a_i}} g^{\hat{b_i}})^{m_i} \\ &= u_0 \cdot \prod_{i=1}^{\ell} u_i^{m_i} \\ &= H_\kappa(m) \end{aligned}$$

5.2 Waters-Signaturen

5.2.1 Konstruktion

- $Gen(1^k)$:

$$\begin{aligned} g^\alpha &\xleftarrow{\$} \mathbb{G} \\ \kappa &\leftarrow Gen_{\text{PHF}}(g) \\ sk &:= g^\alpha \\ pk &:= (g, \kappa, e(g, g^\alpha)) \end{aligned}$$

  (Wir müssen α nicht kennen, da g Erzeuger ist)
- $Sign(sk, m)$:

$$\begin{aligned} r &\xleftarrow{\$} \mathbb{Z}_p \\ \sigma_1 &:= g^r \\ \sigma_2 &:= g^\alpha \cdot H_\kappa(m)^r \\ \sigma &:= (\sigma_1, \sigma_2) \in \mathbb{G}^2 \end{aligned}$$

- $Vfy(pk, m, \sigma)$:

$$e(g, \sigma_2) \stackrel{?}{=} e(g, g^\alpha) * e(\sigma_1, H_\kappa(m))$$

5.2.2 Korrektheit

$$\begin{aligned} e(g, \sigma_2) &= e(g, g^{\alpha} \cdot H_{\kappa}(m)^{r}) \\ &= e(g, g^{\alpha}) \cdot e(g, H_{\kappa}(m)^{r}) \\ &= e(g, g^{\alpha}) \cdot e(g^{r}, H_{\kappa}(m)) \\ &= e(g, g^{\alpha}) \cdot e(\sigma_1, H_{\kappa}(m)) \end{aligned}$$

5.2.3 Eigenschaften

- EUF-CMA-sicher unter der **CDH-Annahme** im **Standardmodell**
- $Gen, Sign, Vfy$ sind effiziente Algorithmen
- Kleine Signaturen (zwei Gruppenelemente)
- Public Key enthält $\kappa := (u_0, \dots, u_\ell)$ (mit ℓ Länge der Nachricht), dadurch **sehr groß**
- Bisher ist die $(1, q, \gamma)$-PHF von Walters die einzig bekannte $(1, q, \gamma)$-PHF

6 RSA-basierte Signaturen

6.1 RSA-Problem und -Annahme

6.1.1 RSA-Problem

Setting:

- P, Q "große" Primzahlen
- $N = P \cdot Q$
- $\varphi(N) = (P-1)(Q-1) = |\mathbb{Z}_N^*|$ (Eulersche Phi-Funktion)
- Wähle $e \in \mathbb{N}$ zufällig, sodass $\mathrm{ggT}(e, \varphi(N)) = 1$
- Dann existiert $d \in \mathbb{N}$ mit $e \cdot d \equiv 1 \mod \varphi(N)$
- Für $x \in \mathbb{Z}_N$ gilt dann auch $x^{e \cdot d} \equiv x \mod N$

RSA-Problem: Gegeben N, e (wie oben) und $y \xleftarrow{\$} \mathbb{Z}_N$, finde $x \in \mathbb{Z}_N : x^e \equiv y \mod N$

6.1.2 RSA-Annahme

Für alle PPT $\mathcal{A}$ gilt:

$$\Pr\left[\mathcal{A}(1^k, N, e, y) = x : \begin{array}{c} N = P \cdot Q, e \xleftarrow{\$} \mathbb{Z}^*_{\varphi(N)} \\ y \xleftarrow{\$} \mathbb{Z}_N, x^e \equiv y \mod N \end{array}\right] \leq negl(k)$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

6.2 Strong-RSA-Problem und -Annahme

6.2.1 Strong-RSA-Problem

Viele RSA-Algorithmen sind nur im Random-Oracle-Modell sicher. Wir würden allerdings gerne ein EUF-CMA-sicheres Signaturverfahren haben, das auf der RSA-Annahme basiert, aber auch im Standardmodell sicher ist.

Dies ist bei einigen Algorithmen mithilfe der **Strong**-RSA-Annahme gegeben. Das Setting ist analog wie zuvor.

Strong-RSA-Problem: Gegeben N "geeignet" und $y \xleftarrow{\$} \mathbb{Z}_N$, finde $x \in \mathbb{Z}_N$ und $e \in \mathbb{N}, e > 1$ mit $x^e \equiv y \mod N$

6.2.2 Strong-RSA-Annahme

Für alle PPT $\mathcal{A}$ gilt:

$$\Pr\left[\mathcal{A}(1^k, N, y) = (x, e) : \begin{matrix} N = P \cdot Q, e > 1 \\ y \xleftarrow{\$} \mathbb{Z}_N, x^e \equiv y \mod N \end{matrix}\right] \leq negl(k)$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

6.2.3 Unterschied zum normalen RSA-Problem und -Annahme

Die Strong-RSA-Annahme ist **stärker** als die RSA-Annahme, da der Angreifer mehr Kontrolle hat.

Das Strong-RSA-Problem hingegen ist **schwächer** als das RSA-Problem, da die Gewinnbedingung "einfacher" zu erfüllen ist.

6.3 Textbook-RSA

6.3.1 Konstruktion

- $Gen(1^k)$:

  Ziehe zufällige Primzahlen P, Q
  $N := P \cdot Q$
  Wähle $e > 2$ mit $\text{ggT}(e, \varphi(N)) = 1$
  $d := e^{-1} \mod \varphi(N)$
  $pk := (N, e)$
  $sk := d$

- $Sign(sk, m)$:

  $$\sigma := m^d \mod N$$

- $Vfy(pk, m, \sigma)$:

  $$\sigma^e \stackrel{?}{\equiv} m \mod N$$

6.3.2 Korrektheit

$$\sigma^e \equiv (m^d)^e \equiv m^{de} \overset{\mod \varphi(N)}{\equiv} m^1 \equiv m \mod N$$

mail@nilslambertz.de
nilslambertz

6.3.3 Sicherheit

Textbook-RSA ist **nicht EUF-1-naCMA-sicher**, da Nachrichten aus zufälligen Signaturen berechnet werden können:

- Wähle $\sigma^* \xleftarrow{\$} \mathbb{Z}_N$
- Berechne $m^* := (\sigma^*)^e \mod N$
- Gebe (m^*, σ^*) als Fälschung aus

Für das Verfahren ist keine einzige Signaturanfrage nötig!

Zudem ist das Verfahren *homomorph*:

- Seien σ_1, σ_2 gültige Signaturen für m_1, m_2
- Dann ist $\sigma_3 := \sigma_1\sigma_2 \mod N$ gültig für $m_3 := m_1 m_2 \mod N$
- da $\sigma_3^e \equiv (\sigma_1\sigma_2)^e \equiv \sigma_1^e\sigma_2^e \equiv m_1 m_2 \equiv m_3 \mod N$

Zur Konstruktion von sicheren RSA-basierten Signaturen wird häufig die Nachricht vorverarbeitet, bevor signiert wird.

6.4 RSA Full-Domain-Hash

6.4.1 Idee

Es sei $H := \{0,1\}^* \to \mathbb{Z}_N$ eine kollisionsresistente Hashfunktion.

- Signiere $H(m)$ mit Textbook-RSA
- Nachrichtenraum (Domäne) bei Textbook-RSA: $\mathbb{Z}_N$
- H soll auf die gesamte Domäne $\mathbb{Z}_N$ abbilden (**Full-Domain-Hash**)

6.4.2 Konstruktion

- $Gen(1^k)$:

  Wie bei Textbook-RSA, außer
  $pk := (N, e, H)$

- $Sign(sk, m)$:

$$\sigma := H(m)^d \mod N$$

- $Vfy(pk, m, \sigma)$:

$$\sigma^e \stackrel{?}{\equiv} H(m) \mod N$$

6.4.3 Korrektheit

$$\sigma^e \equiv (H(m)^d)^e \equiv H(m)^{de} \mod \varphi(N) \equiv H(m)^1 \equiv H(m) \mod N$$

6.4.4 Sicherheit

Wenn die RSA-Annahme gilt, dann ist das RSA-FDH EUF-CMA-sicher im Random-Oracle-Modell.

mail@nilslambertz.de
nilslambertz

6.5 RSA-PSS

Bei **RSA-PSS** wird die Nachricht vorverarbeitet und dann signiert.

6.5.1 Konstruktion

- $Gen(1^k)$:

  Wie bei Textbook-RSA

- $Sign(sk, m)$:

  $$\sigma := \text{PSS-Encode}(m)^d \mod N$$

- $Vfy(pk, m, \sigma)$:

  Berechne $y = \sigma^e \mod N$
  Gib 1 aus gdw. y eine gültige Codierung von m ist

6.6 GHR-Signaturen

6.6.1 Konstruktion

Gennaro-Halevi-Rabin-Signaturen (**GHR-Signaturen**) basieren auf der Strong-RSA-Annahme und benötigen (anders als die vorherigen Verfahren) kein ROM.

Es sei $h := \{0,1\}^* \to \mathbb{P}$ eine Hashfunktion ($\mathbb{P}$ = Primzahlen).

- $Gen(1^k)$:

  Ziehe zufällige Primzahlen P, Q
  $N := P \cdot Q$
  $s \xleftarrow{\$} \mathbb{Z}_N$
  Wähle h so, dass für alle m $\text{ggT}(h(m), \varphi(N)) = 1$ gilt
  $pk := (N, s, h)$
  $sk := \varphi(N) = (P-1)(Q-1)$

- $Sign(sk, m)$:

  $$\sigma := s^{1/h(m)} \mod N$$

- $Vfy(pk, m, \sigma)$:

  $$\sigma^{h(m)} \stackrel{?}{\equiv} s \mod N$$

6.6.2 Hashfunktionen für GHR-Signaturen

Wir haben zwei Bedingungen an unsere Hashfunktion h:

1. h muss auf Primzahlen abbilden
2. h muss so gewählt sein, dass für alle m $\text{ggT}(h(m), \varphi(N)) = 1$ gilt

mail@nilslambertz.de
nilslambertz

6.6.3 Hashfunktionen, die auf Primzahlen abbilden

Hashfunktionen, die auf Primzahlen abbilden, können wie folgt konstruiert werden:

- Sei $H : \{0,1\}^* \to \{0,1\}^\ell$ eine kollisionsresistente Hashfunktion
- Annahme: H bildet "einigermaßen gleichverteilt" nach $\{0,1\}^\ell$ ab
- Definiere $h : \{0,1\}^* \to \mathbb{P} \cap \{0,1\}^\ell$ durch

$$h(m) := H(m \parallel \gamma)$$

  mit $\gamma \in \mathbb{N}$ kleinste Zahl, sodass $H(m \parallel \gamma)$ prim.
- γ kann bei der Auswertung von h durch Hochzählen gefunden werden

6.6.4 Hashfunktionen, für die für alle m gilt, dass ggT$(h(m), \varphi(N)) = 1$

Die Bedingung kann mit der Verwendung von **Strong Primes** erfüllt werden:

- Primzahl P ist eine Strong Prime $\Leftrightarrow$ Es existiert eine Primzahl p mit $P = 2p + 1$
- Wähle Strong Primes $P = 2p + 1, Q = 2q + 1$
- $N = P \cdot Q$
- Dann gilt $\varphi(N) = 4 \cdot p \cdot q$
- Wähle p, q so, dass sie keine Ausgabe von H sind (z.B. $p, q > 2^\ell$)
- Passe H so an, dass 2 und 4 keine möglichen Hashwerte sind

7 Message Authentication Codes

Bei den bisher vorgestellten Signaturverfahren handelt es sich um asymmetrische Verfahren, die ein **Schlüsselpaar** (sk, pk) verwendet haben.

Message Authentication Codes (**MAC**s) werden ebenfalls zur Sicherstellung der Authentizität und Integrität von Nachrichten verwendet, basieren aber auf einem **symmetrischen** System, bei dem das Authentifizieren **und** das Verifizieren mit dem gleichen Schlüssel κ geschieht.

7.1 Grundlagen

7.1.1 Definition

Ein **Message Authentication Code** (MAC) Π ist ein Tupel $(Gen, Sign, Vfy)$ von (P)PT-Algorithmen:

- $Gen(1^k) \to \kappa$
- $Sign(\kappa, m) \to t,\ m \in \{0,1\}^*$
- $Vfy(\kappa, m, t) \in \{0,1\}$

Korrektheit: $\forall \kappa \leftarrow Gen(1^k) \forall m \in \{0,1\}^* : Vfy(\kappa, m, Sign(\kappa, m)) = 1$

$Sign$ kann entweder probabilistisch oder deterministisch sein, der MAC wird dann je nach Fall als **probabilistischer MAC** oder **deterministischer MAC** bezeichnet. t wird "MAC" oder "Tag" genannt.

7.1.2 Fixed- und variable-length MACs

Fixed-length MAC: Es gibt eine Funktion ℓ, sodass für alle $\kappa \leftarrow Gen(1^k)$ der $Sign$-Algorithmus nur für Nachrichten $m \in \{0,1\}^{\ell(k)}$ definiert ist.

MACs mit variabler Nachrichtenlänge werden **variable-length MAC**s genannt.

7.1.3 Kanonische Verifikation

Bei deterministischen MACs kann die Verifikation erfolgen, indem das Tag einfach neu berechnet wird:

$Vfy(\kappa, m, t)$:

- $\tilde{t} := Sign(\kappa, m)$
- `return` $\tilde{t} \stackrel{?}{=} t$

7.2 Sicherheitsbegriffe für MACs

Die Sicherheitsbegriffe **EUF-CMA** und **sEUF-CMA** sind analog zu den Sicherheitsbegriffen für Digitale Signaturen definiert. Bei den jeweiligen Sicherheitsexperimenten sendet der Challenger am Anfang jedoch nun keine Nachricht mehr, da es keinen pk mehr gibt, sondern nur noch einen Schlüssel κ.

sEUF-CMA-sichere MACS werden **strong MACs** genannt.

7.2.1 strong MACs durch eindeutige Tags

Ein MAC-Verfahren hat **eindeutige Tags** $\Leftrightarrow$ Für κ und m gibt es genau einen Tag t mit $Vfy(\kappa, m, t) = 1$.

Ein EUF-CMA-sicherer MAC Π mit eindeutigen Tags ist sEUF-CMA-sicher.

7.2.2 Anmerkungen zu eindeutigen Tags

- MACs, die die kanonische Verifikation verwenden, haben eindeutige Tags
- Deterministische MACs haben eindeutige Tags

7.3 Konstruktion von MACs aus PRFs

7.3.1 Konstruktion eines fixed-length MAC

Es sei $PRF : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k$ eine PRF. Definiere MAC Π für Nachrichten $m \in \{0,1\}^k$ wie folgt:

- $Gen(1^k)$:

$$\kappa \xleftarrow{\$} \{0,1\}^k$$

- $Sign(\kappa, m)$:

$$t \leftarrow PRF(\kappa, m)$$

- $Vfy(\kappa, m, t)$: (*kanonisch*)

$$\tilde{t} := Sign(\kappa, m)$$
$$\tilde{t} \stackrel{?}{=} t$$

7.3.2 Erweiterung auf variable-length MAC

Analog zu Signaturen kann auch bei MACs das Hash-then-Sign-Paradigma verwendet werden.

Es sei $H : \{0,1\}^* \to \{0,1\}^k$ eine Hashfunktion und $\Pi' = (Gen', Sign', Vfy')$ ein fixed-length MAC für Nachrichten $m \in \{0,1\}^k$. Definiere den variable-length MAC Π für Nachrichten $m \in \{0,1\}^*$ wie folgt:

- $Gen(1^k)$:

$$\kappa \leftarrow Gen'(1^k)$$

- $Sign(\kappa, m)$:

$$t \leftarrow Sign'(\kappa, H(m))$$

- $Vfy(\kappa, m, t)$:

$$b \leftarrow Vfy'(\kappa, H(m), t)$$
$$b \stackrel{?}{=} 1$$

7.4 CBC-MAC

Der **CBC-MAC** (**C**ipher **B**lock **C**haining **M**essage **A**uthentication **C**ode) ist verwandt mit dem CBC-Modus für symmetrische Verschlüsselung. Es handelt sich um einen **fixed-length MAC**, der auf **PRF**s basiert.

7.4.1 Konstruktion

Es sei $PRF : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k$ eine PRF, $m \in \{0,1\}^{\ell \cdot k}$ mit $m = m_1, \ldots, m_\ell$ und $|m_i| = k$

- $Gen(1^k)$:

$$\kappa \xleftarrow{\$} \{0,1\}^k$$

- $Sign(\kappa, m)$:

$$t_0 := 0^k$$
$$\texttt{for } i \texttt{ from 1 to } \ell:$$
$$\quad t_i \leftarrow PRF(\kappa, t_{i-1} \oplus m_i)$$
$$\texttt{return } t := t_\ell$$

- $Vfy(\kappa, m, t)$:

$$\texttt{if } |m| \neq \ell \cdot k:$$
$$\quad \texttt{return } 0$$
$$t \stackrel{?}{=} Sign(\kappa, m)$$

7.4.2 Anmerkungen

- Wird unsicher, wenn Nachrichten verschiedener Länge authentifiziert werden
- Wird unsicher, wenn der Initialisierungsvektor $t_0 = 0^k$ durch ein zufälliges t_0 ersetzt wird

7.4.3 Unterschiede zum CBC-Modus bei Verschlüsselung

	CBC-MAC	CBC-Modus bei Verschlüsselung
Initialisierungsvektor	Fest 0^k	Zufällig
Ausgaben	Nur der finale Wert t_ℓ	Alle "Zwischenwerte" t_i

7.5 MACs aus Hashfunktionen

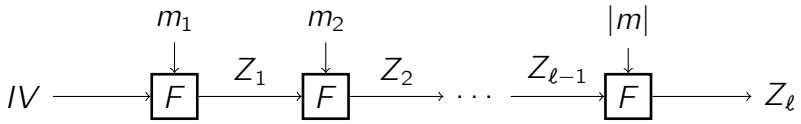
Wir wollen nun MACs basierend auf Hashfunktionen konstruieren. Hashfunktionen sind definiert als $H : \{0,1\}^* \to \{0,1\}^n$ und können mit der **Merkle-Damgård-Transformation** konstruiert werden:

1. Baue Hashfunktion mit *fester* Eingabelänge (**Kompressionsfunktion**)
2. Baue daraus Hashfunktion mit *variabler* Eingabelänge

7.5.1 Merkle-Damgård-Transformation

Gegeben ist eine Kompressionsfunktion $F : \{0,1\}^{2n} \to \{0,1\}^n$. Konstruiere nun eine Hashfunktion $H : \{0,1\}^* \to \{0,1\}^n$ wie folgt:

1. Wähle festen $IV \in \{0,1\}^n$ und setze $Z_0 := IV$
2. Teile m in Blöcke der Länge n auf (falls nötig padde den letzten Block mit Nullen)
3. Füge einen weiteren Block am Ende an, der $|m|$ kodiert
4. Die Anzahl der Blöcke ist dann $\ell = \lceil \frac{|m|}{n} + 1 \rceil$
5. $\forall i \in \{1, \dots, \ell\} : Z_i := F(Z_{i-1} \parallel m_i)$
6. $H(m) := Z_\ell$

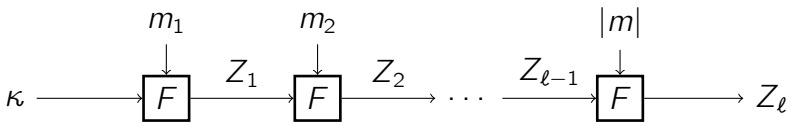


Eigenschaften:

- **Length-Extension-Angriff**: Wenn Hashwert $H(X_1 \parallel X_2)$ bekannt ist, kann $H(X_1 \parallel X_2 \parallel X_3)$ berechnet werden
- Viele bekannte Hashfunktionen basieren auf Merkle-Damgård, z.B. MD5, SHA-1 und SHA-2

7.5.2 Secretly Keyed Hashfunctions

Es sei H eine Hashfunktion, die die Merkle-Damgård-Konstruktion nutzt und F die Kompressionsfunktion in H. Wenn wir nun einen Schlüssel κ in H unterbringen wollen, können wir κ als Initialisierungsvektor IV verwenden:



Notation: $H_{IV}(m)$ ist H mit geheimem Initialisierungsvektor IV angewendet auf m.

$H_{IV}(m)$ wird **Secretly Keyed Hashfunction** genannt.

mail@nilslambertz.de
nilslambertz

7.5.3 NMAC

Es sei H eine Hashfunktion, die die Merkle-Damgård-Konstruktion nutzt, F die Kompressionsfunktion in H und $m \in \{0,1\}^*$:

- $Gen(1^k)$:

$$\kappa := (\kappa_1, \kappa_2) \xleftarrow{\$} \{0,1\}^k \times \{0,1\}^k$$

- $Sign(\kappa, m)$:

$$t \leftarrow F(\kappa_1, H_{\kappa_2}(m))$$

- $Vfy(\kappa, m, t)$: (*kanonisch*)

$$\tilde{t} := Sign(\kappa, m)$$
$$\tilde{t} \stackrel{?}{=} t$$

Eigenschaften:

- Wenn H kollisionsresistent und F in H EUF-CMA-sicherer fixed-length MAC, dann ist ein NMAC EUF-CMA-sicherer variable-length MAC
- NMAC ist auch sicher, wenn $\kappa_2 := 0^k$ fest ist
- Vorteil von NMAC: Sicherheit gilt auch unter *schwächerer* Annahme
- Nachteil von NMAC: IV muss explizit gesetzt werden, H nicht mehr als Black Box verwendbar

7.5.4 HMAC

Ein **HMAC** (Keyed-**H**ash **M**essage **A**uthentication **C**ode) ist ein Spezialfall von NMAC. κ_1, κ_2 werden hier durch Anwendung von F erzeugt (IV ist fest):

1. Definiere zwei Konstanten $opad, ipad \in \{0,1\}^k$
2. Sei κ der Schlüssel im HMAC
3. Setze $\kappa_1 := F(IV \parallel \kappa \oplus opad)$
4. Setze $\kappa_2 := F(IV \parallel \kappa \oplus ipad)$

Es sei $H : \{0,1\}^* \to \{0,1\}^k$ eine Hashfunktion, $opad, ipad \in \{0,1\}^k$ zwei Konstanten und $m \in \{0,1\}^*$:

- $Gen(1^k)$:

$$\kappa \xleftarrow{\$} \{0,1\}^k$$

- $Sign(\kappa, m)$:

$$t \leftarrow H((\kappa \oplus opad) \parallel H((\kappa \oplus ipad) \parallel m))$$

- $Vfy(\kappa, m, t)$: (*kanonisch*)

$$\tilde{t} := Sign(\kappa, m)$$
$$\tilde{t} \stackrel{?}{=} t$$

Informell: HMAC ist genau so sicher wie NMAC, wenn die von F erzeugten κ_1, κ_2 genau so gut wie echte Schlüssel sind.

HMAC ist ein variable-length MAC, der nur einen Schlüssel κ benötigt und *unkeyed* Hashfunktionen verwendet. Er ist standardisiert und in der Praxis weit verbreitet (z.B. in TLS, SSL und SSH).

mail@nilslambertz.de
nilslambertz

8 Symmetrische Verschlüsselung

8.1 Grundlagen

8.1.1 Sicherheitsziel

Bisher wollten wir die Authentizität und Integrität von Nachrichten sicherstellen, entweder asymmetrisch (Digitale Signaturen) oder symmetrisch (MACs).

Nun soll die **Vertraulichkeit** betrachtet werden, also das Geheimhalten des Inhalts durch **Verschlüsselung**. Hierbei gibt es ebenfalls den symmetrischen und den asymmetrischen Ansatz.

8.1.2 Symmetrisches Primitiv

Im **symmetrischen** Fall besitzen Sender und Empfänger den gleichen geheimen Schlüssel. Die Verfahren werden **SKE-Schema** ("secret-key encryption scheme") genannt.

8.1.3 Ablauf

1. Sender und Empfänger einigen sich auf einen geheimen Schlüssel κ
2. Der Sender verschlüsselt die Nachricht m mit κ
3. Der Sender schickt das Chiffrat c an den Empfänger
4. Der Empfänger entschlüsselt das Chiffrat c mit κ und erhält wieder die Nachricht m

8.1.4 Definition: SKE-Schema

Ein SKE-Schema SKE ist ein Tupel (Gen, Enc, Dec) von (P)PT-Algorithmen:

- $Gen(1^k) \to \kappa$
- $Enc(\kappa, m) \to c,\ m \in \mathcal{M}$ (meistens $\mathcal{M} \subseteq \{0,1\}^*$)
- $Dec(\kappa, c) \to m$ (oder $\perp$, falls c kein gültiges Chiffrat)

Korrektheit: $\forall \kappa \leftarrow Gen(1^k) \forall m \in \mathcal{M} : Dec(\kappa, Enc(\kappa, m)) = m$

8.2 Sicherheitsbegriffe

Wie bei Signaturen bestehen unsere Sicherheitsdefinitionen aus einem **Angreiferziel** und einem **Angreifermodell**. Angreiferziele sind beispielsweise OW (*One-way*), IND (*Indistinguishability*), ROR (*Real-Or-Random*) und NM (*Non-Malleability*). Angreifermodelle sind CPA (*chosen-plaintext attack*), CCA1 (*non-adaptive chosen-ciphertext attack*) und CCA2 (*adaptive chosen-ciphertext attack*).

Wir betrachten **IND-CPA** und **IND-CCA2** (oft auch nur IND-CCA genannt).

8.2.1 IND-CPA-Sicherheit

Bei **IND-CPA**-Sicherheit ist das Angreiferziel **Ind**istinguishability (kein Angreifer kann die Chiffrate von zwei selbstgewählten Nachrichten voneinander unterscheiden) und das Angreifermodell **c**hosen-**p**laintext **a**ttack (Angreifer hat Zugriff auf Orakel, über das er beliebige Nachrichten verschlüsseln lassen kann).

8.2.2 Definition: IND-CPA für SKE

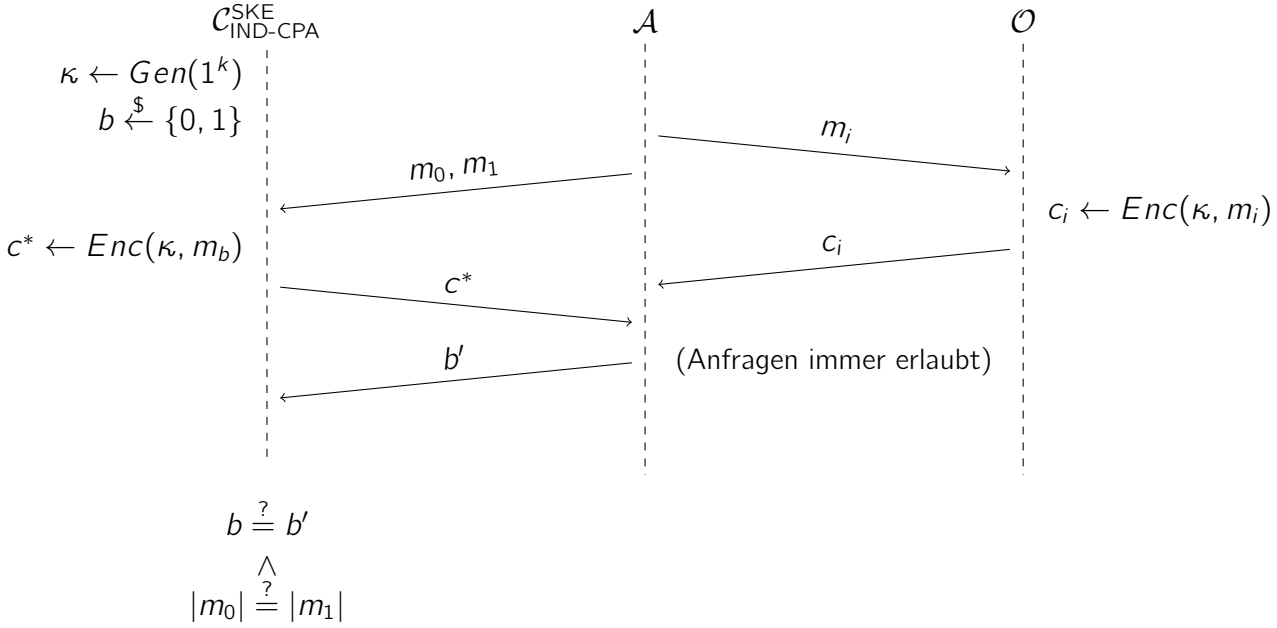
Ein SKE-Schema SKE $= (Gen, Enc, Dec)$ ist *IND-CPA-sicher*, wenn für alle PPT $\mathcal{A}$ gilt, dass

$$\begin{aligned}&\Pr[\mathcal{A} \text{ gewinnt IND-CPA-Experiment}]\\ &= \Pr[\mathcal{A}^{\mathcal{C}^{\text{SKE}}_{\text{IND-CPA}}}(1^k) = b' : b = b' \wedge |m_0| = |m_1|]\\ &\leq \frac{1}{2} + negl(k)\end{aligned}$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

8.2.3 Visualisierung: IND-CPA-Sicherheitsexperiment

Bei dem IND-CPA-Sicherheitsexperiment wird das Verschlüsselungs-Orakel $\mathcal{O}$ ebenfalls vom Challenger implementiert und ist in der Visualisierung zur besseren Lesbarkeit getrennt. Der Angreifer $\mathcal{A}$ darf zu jedem Zeitpunkt Verschlüsselungs-Anfragen an das Orakel stellen.



$\mathcal{A}$ gewinnt, falls $b = b'$ **und** $|m_0| = |m_1|$

8.2.4 IND-CCA2-Sicherheit

Bei **IND-CCA2**-Sicherheit ist das Angreiferziel **Ind**istinguishability (kein Angreifer kann die Chiffrate von zwei selbstgewählten Nachrichten voneinander unterscheiden) und das Angreifermodell (adaptive) **c**hosen-**c**iphertext **a**ttack (Angreifer hat Zugriff auf Orakel, über das er beliebige Nachrichten verschlüsseln lassen kann und beliebige Nachrichten außer das *Challenge-Chiffrat* entschlüsseln lassen kann).

8.2.5 Definition: IND-CCA2 für SKE

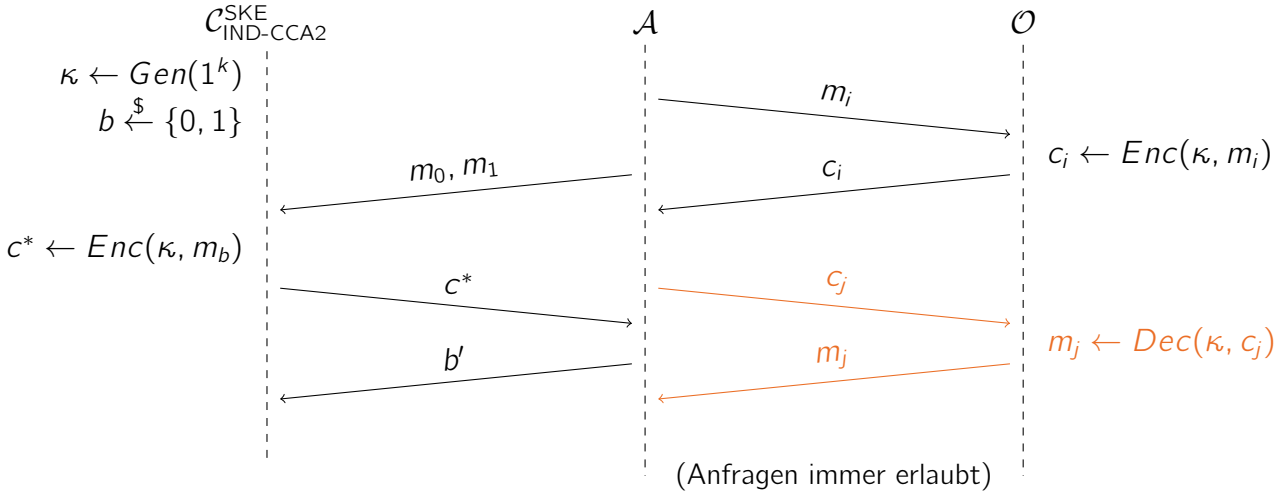
Ein SKE-Schema SKE $= (Gen, Enc, Dec)$ ist *IND-CCA2-sicher*, wenn für alle PPT $\mathcal{A}$ gilt, dass

$$\begin{aligned}&\Pr[\mathcal{A} \text{ gewinnt IND-CCA2-Experiment}]\\ &= \Pr[\mathcal{A}^{\mathcal{C}^{\text{SKE}}_{\text{IND-CCA2}}}(1^k) = b' : b = b' \wedge |m_0| = |m_1| \wedge c^* \neq c_j \forall j]\\ &\leq \frac{1}{2} + negl(k)\end{aligned}$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

8.2.6 Visualisierung: IND-CCA2-Sicherheitsexperiment

Bei dem IND-CCA2-Sicherheitsexperiment wird das Orakel $\mathcal{O}$ ebenfalls vom Challenger implementiert und ist in der Visualisierung zur besseren Lesbarkeit getrennt. Der Angreifer $\mathcal{A}$ darf zu jedem Zeitpunkt Verschlüsselungs-Anfragen und Entschlüsselungs-Anfragen an das Orakel stellen.



$b \stackrel{?}{=} b' \land |m_0| \stackrel{?}{=} |m_1| \land c^* \neq c_j \forall j$

$\mathcal{A}$ gewinnt, falls $b = b'$ **und** $|m_0| = |m_1|$ **und** $c^* \neq c_j \forall j$

8.3 Konstruktion eines IND-CCA2-sicheren SKE-Schemas

Wir wollen nun ein IND-CCA2-sicheres SKE-Schema aus einem IND-CPA-sicheren SKE-Schema konstruieren. Dazu verschlüsseln wir erst und wenden dann einen MAC an.

8.3.1 Konstruktion

Sei $SKE' = (Gen', Enc', Dec')$ ein IND-CPA-sicheres SKE-Schema und $\Pi = (Gen_\Pi, Sign, Vfy)$ ein EUF-CMA-sicherer MAC. Unser SKE-Schema SKE ist wie folgt definiert:

- $Gen(1^k)$:

$$\begin{aligned}\kappa_1 &\leftarrow Gen'(1^k)\\ \kappa_2 &\leftarrow Gen_\Pi(1^k)\\ \kappa &:= (\kappa_1, \kappa_2)\end{aligned}$$

- $Enc(\kappa, m)$:

$$\begin{aligned}c' &\leftarrow Enc'(\kappa_1, m)\\ t &\leftarrow Sign(\kappa_2, c')\\ c &:= (c', t)\end{aligned}$$

- $Dec(\kappa, c)$:

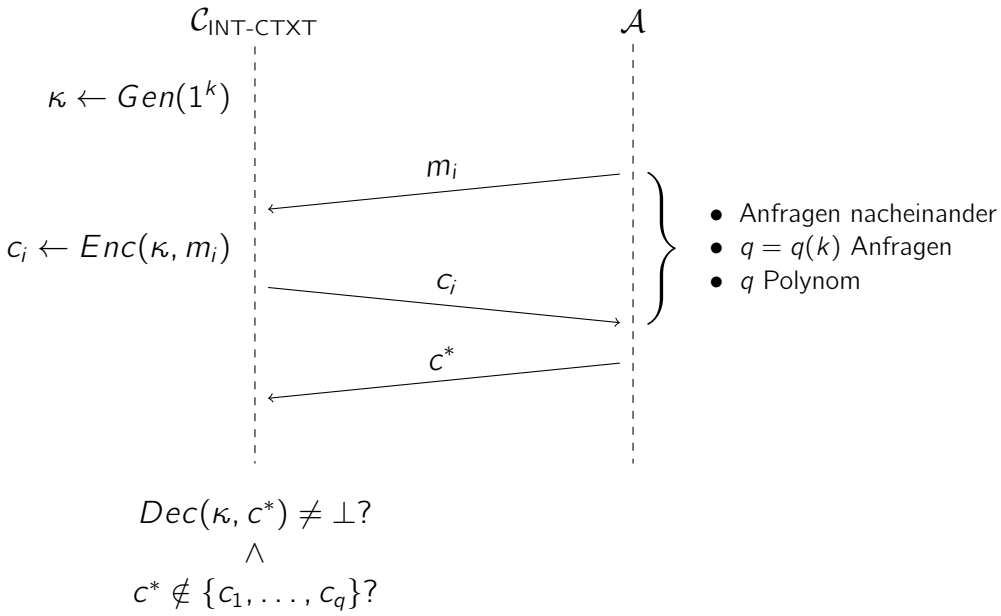
$$\begin{aligned}&\texttt{if } Vfy(\kappa_2, c', t) = 0, \texttt{ gib } \perp \texttt{ aus}\\ &m := Dec'(\kappa_1, c')\end{aligned}$$

Wenn SKE' ein IND-CPA-sicheres SKE-Schema und Π ein EUF-CMA-sicherer MAC mit eindeutigen Tags ist, dann ist SKE IND-CCA2-sicher.

8.4 Authentifizierte Verschlüsselung

MACs stellen die **Integrität** von Nachrichten sicher, während symmetrische Verschlüsselung die **Vertraulichkeit** von Nachrichten sicherstellt. **Authentifizierte Verschlüsselung** stellt beide Eigenschaften sicher.

8.4.1 Visualisierung: INT-CTXT-Sicherheitsexperiment für SKE



$\mathcal{A}$ gewinnt, falls $Vfy(pk, m^*, \sigma^*) = 1$ **und** $m^* \notin \{m_1, \ldots, m_q\}$

8.4.2 Definition: INT-CTXT für SKE

Ein SKE-Schema SKE $= (Gen, Enc, Dec)$ besitzt *ciphertext integrity (INT-CTXT)*, wenn für alle PPT $\mathcal{A}$ gilt, dass

$$\begin{aligned}&\Pr[\mathcal{A} \text{ gewinnt INT-CTXT-Experiment}]\\&= \Pr[\mathcal{A}^{\mathcal{C}_{\text{INT-CTXT}}}(1^k) = c^* : Dec(\kappa, c^*) \neq \bot \wedge c^* \notin \{c_1, \ldots, c_q\}]\\&\leq negl(k)\end{aligned}$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

8.4.3 Definition: Authentifizierte Verschlüsselung

Ein Verschlüsselungsschema $AE = (Gen, Enc, Dec)$ ist ein **authentifiziertes Verschlüsselungsschema** (bzw. *AE-sicher*), falls es IND-CPA-sicher und INT-CTXT-sicher ist.

8.4.4 Ansätze zur Konstruktion eines Authentifiziertes Verschlüsselungsschemas

Als Bausteine für ein authentifiziertes Verschlüsselungsschema verwenden wir *symmetrische Verschlüsselung* (IND-CPA-sicher) und *MACs* (EUF-CMA-sicher mit eindeutigen Tags). Möglichkeiten zur Konstruktion:

- "Encrypt-and-Mac" (verschlüsseln und authentifizieren): **Nicht immer sicher**, da ein MAC die Nachricht nicht geheim halten muss
- "Mac-then-Encrypt" (erst authentifizieren, dann verschlüsseln): **Nicht immer sicher**, da die Verschlüsselung "außen" ist und das Chiffrat verändert werden kann
- "Encrypt-then-Mac" (erst verschlüsseln, dann authentifizieren): **Beweisbar sicher**

8.4.5 Anmerkungen zu authentifizierter Verschlüsselung

- **Sicherheitsziele** bei Authentifizierter Verschlüsselung und IND-CCA2-SKE sind **unterschiedlich**
- es müssen **zwei verschiedene Schlüssel** für den MAC bzw. das SKE verwendet werden, sonst kann das Verfahren unsicher sein
- jedes AE-Schema ist auch IND-CCA2-sicher
- Encrypt-then-Mac hat in der Praxis Nachteile (langsamer als SKE, zwei Schlüssel nötig), daher werden meist Blockchiffren mit speziellen Betriebsmodi verwendet, die authentifizierte Verschlüsselung garantieren (z.B. der GCM-Betriebsmodus)

9 Asymmetrische Verschlüsselung

9.1 Grundlagen

9.1.1 Asymmetrisches Primitiv

Im **asymmetrischen** Fall erzeugt ein Sender ein Schlüsselpaar (pk, sk). pk wird zum Verschlüsseln verwendet und kann öffentlich geteilt werden, während sk zum Entschlüsseln verwendet wird. Die Verfahren werden **PKE-Schema** ("public-key encryption scheme") genannt.

9.1.2 Ablauf

1. Empfänger erzeugt ein Schlüsselpaar (pk, sk) und teilt pk mit dem Sender
2. Der Sender verschlüsselt die Nachricht m mit pk
3. Der Sender schickt das Chiffrat c an den Empfänger
4. Der Empfänger entschlüsselt das Chiffrat c mit sk und erhält wieder die Nachricht m

9.1.3 Definition: PKE-Schema

Ein PKE-Schema PKE ist ein Tupel (Gen, Enc, Dec) von (P)PT-Algorithmen:

- $Gen(1^k) \to (pk, sk)$
- $Enc(pk, m) \to c,\ m \in \mathcal{M}$
- $Dec(sk, c) \to m$

Korrektheit: $\forall (pk, sk) \leftarrow Gen(1^k) \forall m \in \mathcal{M} : Dec(sk, Enc(pk, m)) = m$

9.2 Sicherheitsbegriffe

Für PKE werden dieselben Sicherheitsbegriffe wie bei symmetrischer Verschlüsselung verwendet (Unterscheidung z.B. durch Notation: *SKE-IND-CPA* und *PKE-IND-CPA*). Bei Sicherheitsexperimenten mit PKE erhält $\mathcal{A}$ pk, aber natürlich nicht sk. Damit kann er Chiffrate selbst berechnen und benötigt kein Orakel.

9.2.1 Definition: IND-CPA für PKE

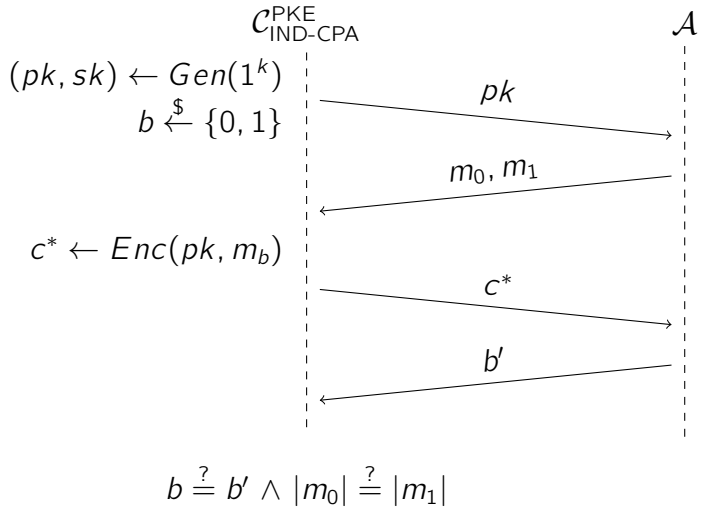
Ein PKE-Schema PKE $= (Gen, Enc, Dec)$ ist *IND-CPA-sicher*, wenn für alle PPT $\mathcal{A}$ gilt, dass

$$\begin{aligned} &\Pr[\mathcal{A} \text{ gewinnt IND-CPA-Experiment}] \\ &= \Pr[\mathcal{A}^{\mathcal{C}^{\text{PKE}}_{\text{IND-CPA}}}(1^k) = b' : b = b' \wedge |m_0| = |m_1|] \\ &\leq \frac{1}{2} + negl(k) \end{aligned}$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

9.2.2 Visualisierung: IND-CPA-Sicherheitsexperiment

Der Angreifer $\mathcal{A}$ kann mit pk selbst jederzeit Nachrichten verschlüsseln.



$\mathcal{A}$ gewinnt, falls $b = b'$ **und** $|m_0| = |m_1|$

9.2.3 Definition: IND-CCA2 für PKE

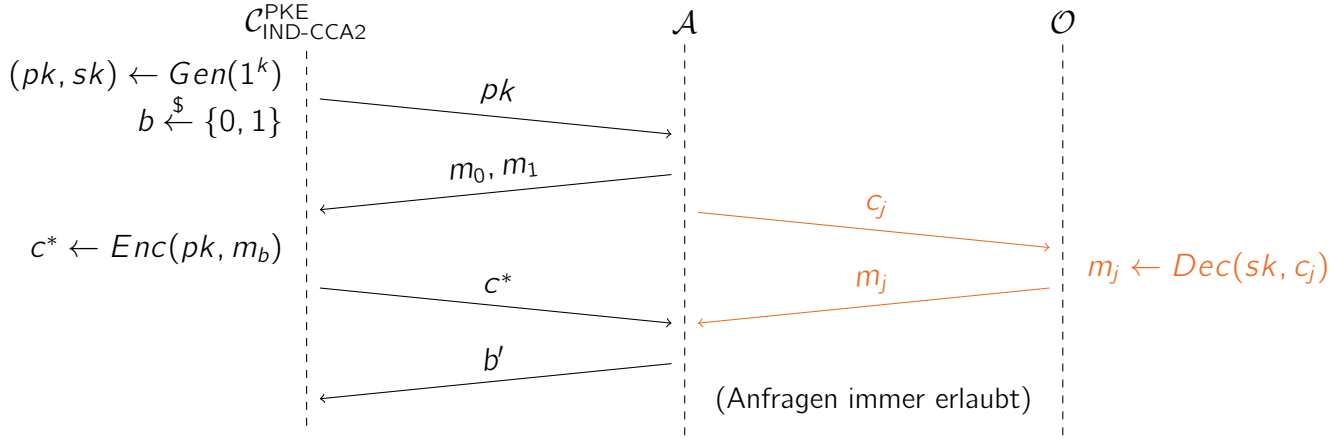
Ein PKE-Schema PKE $= (Gen, Enc, Dec)$ ist *IND-CCA2-sicher*, wenn für alle PPT $\mathcal{A}$ gilt, dass

$$\begin{aligned}&\Pr[\mathcal{A} \text{ gewinnt IND-CCA2-Experiment}]\\ &= \Pr[\mathcal{A}^{\mathcal{C}^{\text{PKE}}_{\text{IND-CCA2}}}(1^k) = b' : b = b' \wedge |m_0| = |m_1| \wedge c^* \neq c_j \forall j]\\ &\leq \frac{1}{2} + negl(k)\end{aligned}$$

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

9.2.4 Visualisierung: IND-CCA2-Sicherheitsexperiment

Das Entschlüsselungs-Orakel $\mathcal{O}$ wird ebenfalls vom Challenger implementiert.



$\mathcal{A}$ gewinnt, falls $b = b'$ **und** $|m_0| = |m_1|$ **und** $c^* \neq c_j \forall j$

mail@nilslambertz.de
nilslambertz

9.3 Signcryption

9.3.1 Problem bei der Konstruktion eines IND-CCA2-sicheren PKE-Schemas

Wir wollen auch für PKE ein IND-CCA2-sicheres Verfahren konstruieren. Bei SKE konnten wir dafür Encrypt-then-Mac verwenden.

Das analoge Prinzip "Encrypt-then-Sign" (erst mit pk des Empfängers verschlüsseln und dann mit sk des Senders signieren) funktioniert für PKE allerdings nicht, da jeder, der den pk des Empfängers kennt, eine Nachricht an ihn verschlüsseln kann.

9.3.2 Grundlagen der Signcryption

Signcryption ist der asymmetrische Variante von authentifizierter Verschlüsselung, die durch die Kombination eines PKE-Schemas und eines Signaturverfahrens erreicht wird. Dabei sollen **Vertraulichkeit** (IND-CCA2), **Integrität** (INT-CTXT) und **Authentizität** sichergestellt werden.

9.3.3 Authentizität bei Signcryption

Naive Ansätze der Signcryption ("Sign-then-Encrypt" oder "Encrypt-then-Sign") sind anfällig für **Man-in-the-Middle-Angriffe**, da jeder die Nachricht abfangen, entschlüsseln und neu verschlüsseln kann.

Die Signatur und das Chiffrat muss also zusätzliche Informationen (wie *Identitäten*) beinhalten.

9.3.4 Definition: PKE-Schema mit assoziierten Daten

Ein **PKE-Schema mit assoziierten Daten** PKE_{AD} ist ein Tripel (Gen, Enc, Dec) von PPT-Algorithmen:

- $Gen(1^k) \to (pk, sk)$
- $Enc(pk, m, d) \to c,\ m \in \mathcal{M}, d \in \mathcal{D}$
- $Dec(sk, c, d) \to m$ (oder $\perp$, falls c kein gültiges Chiffrat)

Korrektheit: $\forall (pk, sk) \leftarrow Gen(1^k) \forall m \in \mathcal{M} \forall d \in \mathcal{D} : Dec(sk, Enc(pk, m, d), d) = m$

d ist hierbei das *assoziierte Datum* und **bleibt nicht geheim** (z.B. Header oder Metadaten).

9.3.5 Definition: Signcryption-Schema

Ein **Signcryption-Schema** SC ist ein Tripel (Gen, Enc, Dec) von PPT-Algorithmen:

- $Gen(1^k) \to (pk, sk)$
- $Enc(sk_S, id_S, pk_R, id_R, m) \to c,\ m \in \mathcal{M}, id_S, id_R \in \mathcal{ID}$
- $Dec(pk_S, id_S, sk_R, id_R, c) \to m$ (oder $\perp$, falls c kein gültiges Chiffrat)

Korrektheit:

$$
\begin{aligned}
&\forall (pk_S, sk_S) \leftarrow Gen(1^k) \\
&\forall (pk_R, sk_R) \leftarrow Gen(1^k) \\
&\forall id_S, id_R \in \mathcal{ID} \\
&\forall m \in \mathcal{M} : \\
&Dec(pk_S, id_S, sk_R, id_R, Enc(sk_S, id_S, pk_R, id_R, m)) = m
\end{aligned}
$$

INT-CTXT-Experiment für Signcryption ausgelassen.

9.3.6 Definition: Ciphertext integrity für Signcryption

Ein Signcryption-Schema $SC = (Gen, Enc, Dec)$ ist INT-CTXT-sicher, wenn jeder PPT-Angreifer $\mathcal{A}$ das INT-CTXT-Experiment nur mit vernachlässigbarer Wahrscheinlichkeit gewinnt.

9.3.7 Definition: Sicheres Signcryption-Schema

Ein Signcryption-Schema $SC = (Gen, Enc, Dec)$ ist sicher, wenn es INT-CTXT-sicher und IND-CCA2-sicher ist.

9.3.8 Konstruktion von Signcryption

Zur Konstruktion eines Signcryption-Verfahrens kombinieren wir ein PKE-Schema mit assoziierten Daten und ein Signaturverfahren. Die Identität des Senders ist im Chiffrat gespeichert und die des Empfängers in der Signatur.

Es kann entweder "Encrypt-then-Sign" oder "Sign-then-Encrypt" verwendet werden. Grundbausteine für das Signcryption-Schema $SC = (Gen, Enc, Dec)$:

- PKE-Schema $PKE_{AD} = (Gen', Enc', Dec')$
- Signaturverfahren $\Sigma = (Gen^*, Sign, Vfy)$
- $Gen(1^k)$:

$$
\begin{aligned}
&(pk', sk') \leftarrow Gen'(1^k)\\
&(pk^*, sk^*) \leftarrow Gen^*(1^k)\\
&pk := (pk', pk^*)\\
&sk := (sk', sk^*)
\end{aligned}
$$

9.3.9 Signcryption: Encrypt-then-Sign

- $Enc(sk_S, id_S, pk_R, id_R, m)$:

$$
\begin{aligned}
&c \leftarrow Enc'(pk'_R, m, id_S)\\
&\sigma \leftarrow Sign(sk^*_S, (c, id_R))\\
&\textbf{Chiffrat: } (c, \sigma)
\end{aligned}
$$

- $Dec(pk_S, id_S, sk_R, id_R, c)$:

$$
\begin{aligned}
&\texttt{if } Vfy(pk^*_S, (c, id_R), \sigma) = 0, \texttt{return } \perp\\
&m := Dec'(sk'_R, c, id_S)
\end{aligned}
$$

SC ist sicher, wenn PKE_{AD} IND-CCA2-sicher und Σ sEUF-CMA-sicher ist.

9.3.10 Signcryption: Sign-then-Encrypt

- $Enc(sk_S, id_S, pk_R, id_R, m)$:

$$
\begin{aligned}
&\sigma \leftarrow Sign(sk^*_S, (m, id_R))\\
&c \leftarrow Enc'(pk'_R, (m, \sigma), id_S)\\
&\textbf{Chiffrat: } c
\end{aligned}
$$

- $Dec(pk_S, id_S, sk_R, id_R, c)$:

$$(m, \sigma) := Dec'(sk*'_R, c, id_S)$$
$$\texttt{if } Vfy(pk^*_S, (m, id_R), \sigma) = 0, \texttt{return } \perp$$
$$\texttt{return } m$$

SC ist sicher, wenn PKE_{AD} IND-CCA2-sicher und Σ EUF-CMA-sicher ist.

9.4 Naor-Yung-Konstruktion

Ziel: IND-CCA2-sicheres PKE-Schema

9.4.1 Überblick der Naor-Yung-Konstruktion

- Generische Konstruktion für ein IND-CCA2-sicheres PKE-Schema aus einem IND-CPA-sicheren PKE-Schema
- Nutzt *SS-NIZKs* als Baustein
- Idee: **Doppelverschlüsselung**

9.4.2 Definition: NP-Sprache

Eine Sprache L ist genau dann in NP, wenn es eine Relation $R_L \subseteq (L \times \{0,1\}^*)$ mit $R_L \in \text{P}$ gibt, wobei

$$x \in L \iff \exists w : (x, w) \in R_L$$

Weiterhin muss $|w|$ beschränkt sein durch $p(|x|)$, wobei p wieder ein beliebiges Polynom ist. w heißt *Zeuge* und R_L *Zeugenrelation*.

9.4.3 Definition: SS-NIZK

Ziel: Zeige mittels eines *Beweissystems*, dass $x \in L$ gilt.

Sei $L \in$ NP eine NP-Sprache mit PPT-Zeugenrelation R_L. Ein **S**imulation-**S**ound **N**on-**I**nteractive **Z**ero-**K**nowledge Proof System für L besteht aus drei Algorithmen:

- $Setup(1^k)$: Generiert einen CRS (common reference string)
- $Prove(CRS, x, w)$: Gibt einen Beweis π aus
- $Verify(CRS, x, \pi)$: Gibt 0 oder 1 aus

9.4.4 Eigenschaften: SS-NIZK

- **Vollständigkeit:** Für $x \in L$ generiert $Prove$ einen Beweis, der von $Verify$ akzeptiert wird
- **Zero-Knowledge:** Der Verifizierer lernt außer $x \in L$ oder $x \notin L$ nichts (intuitiv)
- **Soundness:** Es ist schwer einen korrekten (also von $Verify$ akzeptierten) Beweis π für eine falsche Aussage $x \notin L$ zu finden

9.4.5 Bausteine für die Naor-Yung-Konstruktion

Der erste Baustein ist ein IND-CPA-sicheres PKE-Schema $PKE_{CPA} = (Gen_{CPA}, Enc_{CPA}, Dec_{CPA})$.

Zudem wird die Sprache L_{NY} benötigt:

$$L_{NY} := \{(pk_1, pk_2, c_1, c_2) | \exists m, r_1, r_2 : c_1 = Enc(pk_1, m; r_1) \wedge c_2 = Enc(pk_2, m; r_2)\}$$

Die Notation $Enc(pk_1, m; r_1)$ bedeutet, dass die Nachricht m unter dem öffentlichen Schlüssel pk_1 verschlüsselt wird und dabei der Zufall r_1 verwendet wird.

Die Elemente von L_{NY} sind also Paare von Chiffraten derselben Nachricht, die zugehörigen Zeugen haben die Form (m, r_1, r_2).

9.4.6 Naor-Yung-Konstruktion

Definiere $PKE_{NY} = (Gen_{NY}, Enc_{NY}, Dec_{NY})$ wir folgt:

- $Gen_{NY}(1^k)$:

$$
\begin{aligned}
&(pk_1, sk_1) \leftarrow Gen_{CPA}(1^k)\\
&(pk_2, sk_2) \leftarrow Gen_{CPA}(1^k)\\
&CRS \leftarrow Setup(1^k)\\
&pk := (pk_1, pk_2, CRS)\\
&sk := (pk, sk_1)
\end{aligned}
$$

- $Enc_{NY}(pk, m)$:

$$
\begin{aligned}
&(pk_1, pk_2, CRS) := pk\\
&c_1 \leftarrow Enc_{CPA}(pk_1, m; r_1)\\
&c_2 \leftarrow Enc_{CPA}(pk_2, m; r_2)\\
&x := (pk_1, pk_2, c_1, c_2)\\
&w := (m, r_1, r_2)\\
&\pi \leftarrow Prove(CRS, x, w)\\
&c := (c_1, c_2, \pi)
\end{aligned}
$$

- $Dec_{NY}(sk, c)$:

$$
\begin{aligned}
&(pk, sk_1) := sk\\
&(pk_1, pk_2, CRS) := pk\\
&(c_1, c_2, \pi) := c\\
&x := (pk_1, pk_2, c_1, c_2)\\
&m \leftarrow Dec_{CPA}(sk_1, c_1)\\
&\texttt{if } Verify(CRS, x, \pi) = 0 : \texttt{Gib } \perp \texttt{ aus}\\
&\texttt{else: Gib } m \texttt{ aus}
\end{aligned}
$$

Wenn das verwendete Beweissystem ein SS-NIZK ist und PKE_{CPA} IND-CPA-sicher ist, dann ist PKE_{NY} IND-CCA2-sicher.

mail@nilslambertz.de
nilslambertz

10 Identitätsbasierte Verschlüsselung

10.1 Grundlagen

10.2 Motivation

- Für SKE-Schemata wird ein geheimer Schlüssel benötigt, der vor der Kommunikation ausgetauscht werden muss
- Für PKE-Schemata muss der Sender den *pk* des Empfängers kennen

Identitätsbasierte Verschlüsselung bzw. *Identity-based encryption* (IBE) kommt komplett ohne Schlüsselaustausch aus.

10.2.1 Überblick

- Man muss nur die *Identität des Empfängers* kennen, um ihm eine **verschlüsselte** Nachricht zu senden
- Identität kann z.B. eine Mailadresse sein
- Der Empfänger kann sich zum Entschlüsseln einen *user secret key* von einer vertrauenswürdigen Entität besorgen
- Diese Entität kennt den *master secret key*, mit dem *user secret keys* generiert werden können
- Ein *master public key* wird zum Verschlüsseln verwendet

10.2.2 Definition: IBE-Schema

Ein IBE-Schema mit Nachrichtenraum $\mathcal{M}_k$ und Identitätsraum $\mathcal{ID}_k$ ist ein Tupel $IBE = (Gen, Ext, Enc, Dec)$ Tupel aus vier PPT-Algorithmen:

- $(mpk, msk) \leftarrow Gen(1^k)$: Generiert das Master-Schlüsselpaar
- $usk_{id} \leftarrow Ext(msk, id)$ (*Extraktionsalgorithmus*): Gibt *user secret key* für Identität $id \in \mathcal{ID}_k$ aus
- $c \leftarrow Enc(mpk, id, m)$ (*Verschlüsselungsalgorithmus*): Verschlüsselt eine Nachricht an id
- $m \leftarrow Dec(usk_{id}, c)$ (*Entschlüsselungsalgorithmus*): Entschlüsselt eine Nachricht

Korrektheit:

$$
\begin{aligned}
&\forall k \in \mathbb{N} \\
&\forall m \in \mathcal{M}_k \\
&\forall id \in \mathcal{ID}_k \\
&\forall (mpk, msk) \leftarrow Gen(1^k) \\
&\forall usk_{id} \leftarrow Ext(msk, id) \\
&\forall c \leftarrow Enc(mpk, id, m) \\
&Dec(usk_{id}, c) = m
\end{aligned}
$$

mail@nilslambertz.de
nilslambertz

Literatur

[Joux, 2006] Joux, A. (2006). A one round protocol for tripartite diffie–hellman. volume 17, pages 385–393.