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 Kryptographischen Verfahren

Kryptographische Verfahren 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:

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

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.

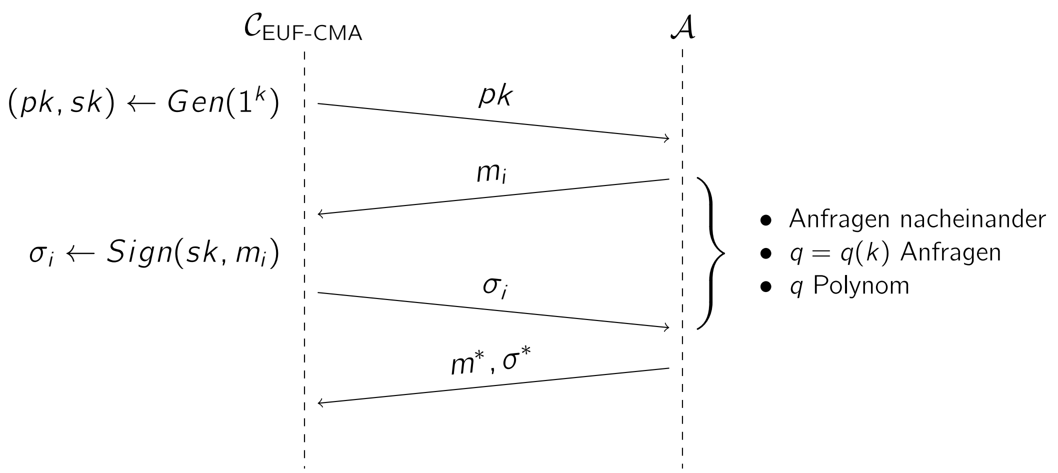
- EUF-CMA
- EUF-naCMA

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

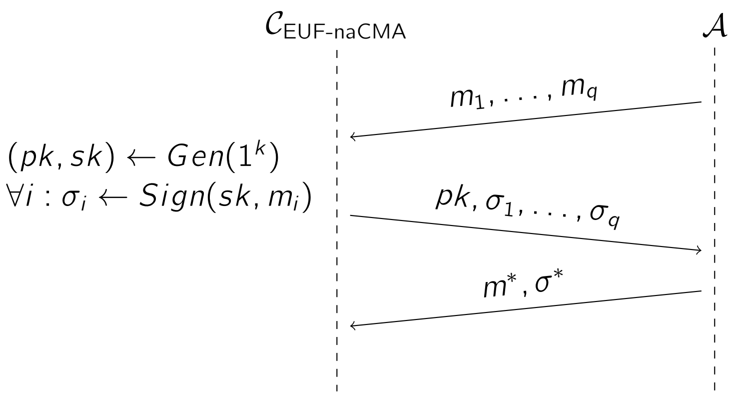
Ein digitales Signaturverfahren $\Sigma = (Gen, Sign, Vfy)$ ist *EUF-CMA-sicher*, wenn für alle PPT $\mathcal{A}$ gilt, dass

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

für eine im Sicherheitsparameter k vernachlässigbare Funktion $negl$.

1.6 EUF-naCMA-Sicherheitsexperiment

1.6.1 Visualisierung: EUF-naCMA-Sicherheitsexperiment



- $q = q(k)$ Nachrichten
- q Polynom

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

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

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

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

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

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

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:

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

mail@nilslambertz.de
nilslambertz

- $Sign(sk, m)$:

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

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

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

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
- $Gen(1^k)$:

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

- $Sign(sk, m)$:

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

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

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

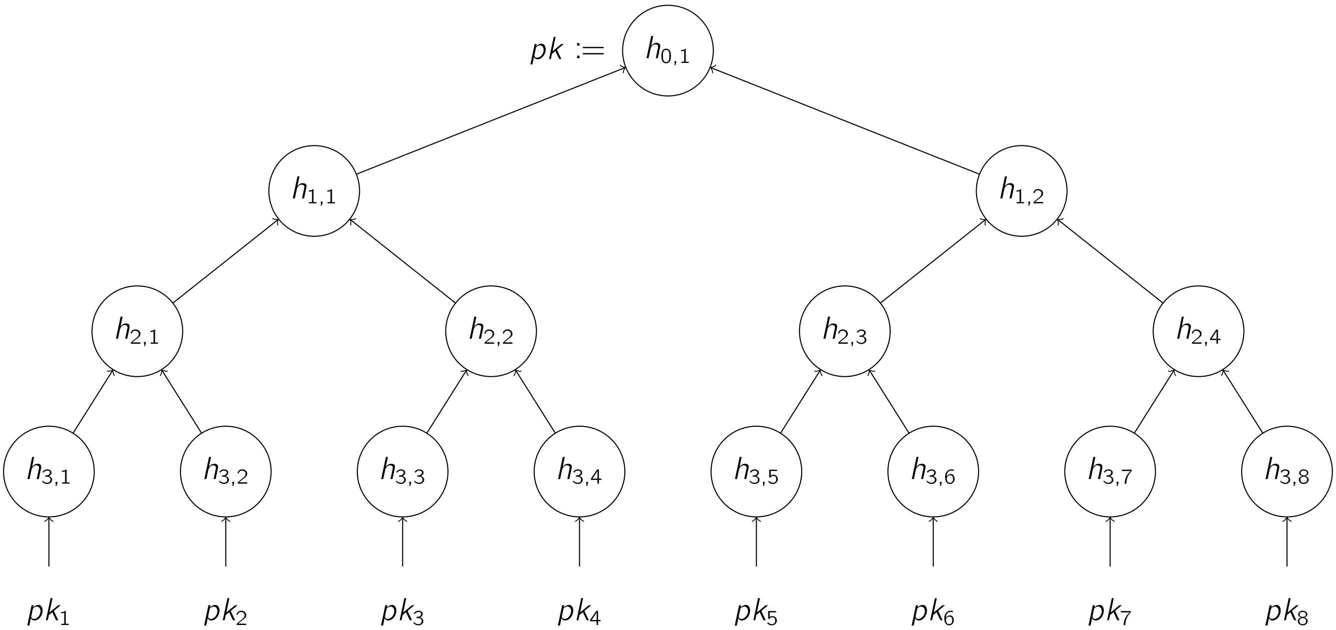
Eigenschaften bezogen auf Signaturanzahl (q):

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

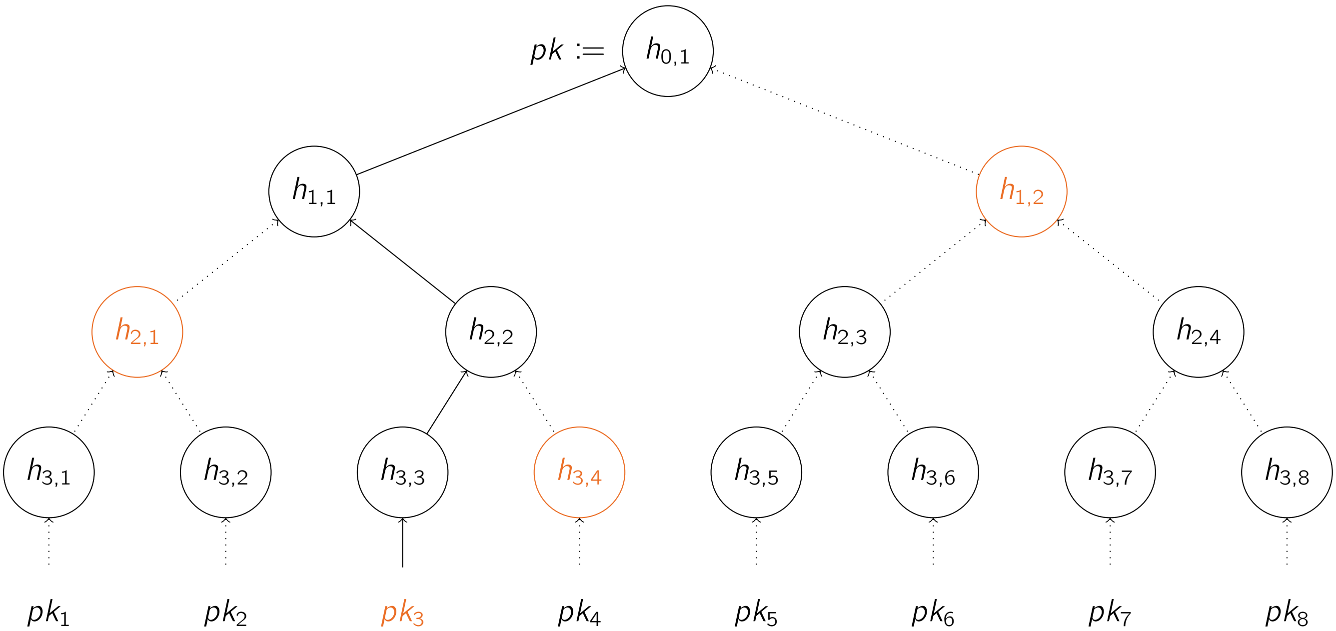
mail@nilslambertz.de
nilslambertz

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 $u_1, \dots, u_n$ 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.

- $Gen(1^k)$:

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

mail@nilslambertz.de
nilslambertz

- $Sign(sk, m)$:

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

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

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

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:

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

2.3.2 Schlüsselgenerierung

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

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

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:

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

Mit **echt zufälliger** Funktion F also:

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

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

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

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 $y \overset{\$}{\leftarrow} \mathbb{G}$
- Finde $x \in \mathbb{Z}_p : g^x = y$

Die **DLog-Annahme** ist folgende:

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

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

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 $\mathbb{G}$:

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

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

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

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

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

- $Gen(1^k)$:

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

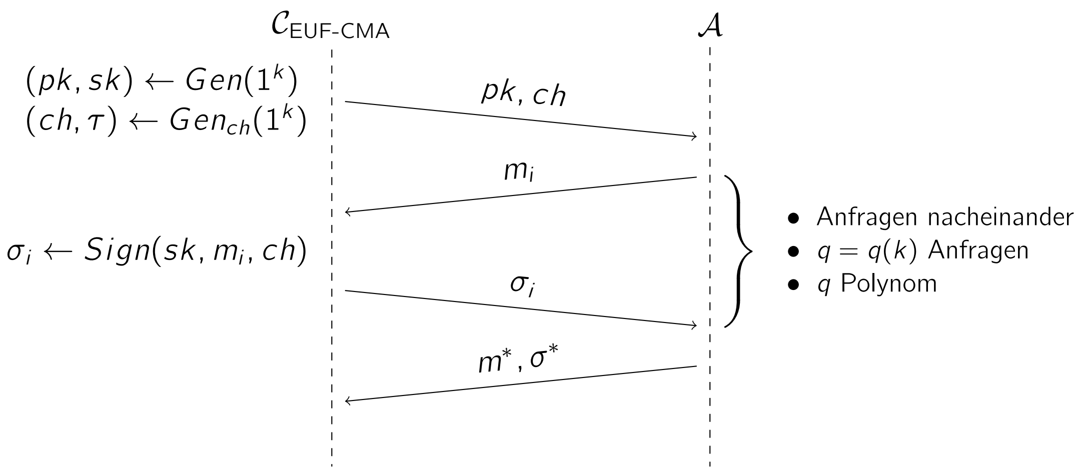
- $Sign(sk, m, ch)$ (ch ist die CH-Fkt. des **Empfängers**):

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

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

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

3.2.2 Visualisierung: EUF-CMA-Sicherheitsexperiment



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

$\mathcal{A}$ 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.*

mail@nilslambertz.de
nilslambertz

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

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

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.*

mail@nilslambertz.de
nilslambertz

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

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

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:

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

- 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

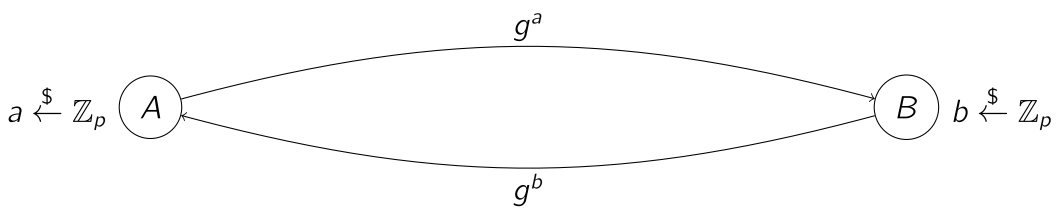
$$\psi : \mathbb{G}_1 \to \mathbb{G}_2$$

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



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

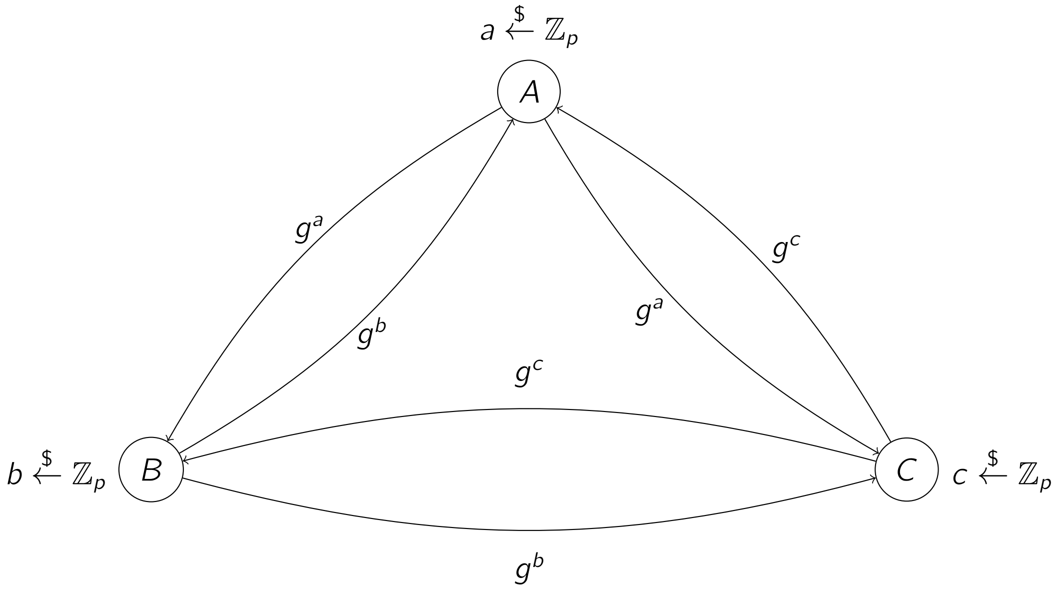
Ablauf:

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:

- $Gen(1^k)$:

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

- $Sign(sk, m)$:

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

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

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

mail@nilslambertz.de
nilslambertz

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 $h^a g^b = H_\kappa(m)$

mail@nilslambertz.de
nilslambertz

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

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

  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 $m_i = 1$
- $TrapGen(g, h)$:

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

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

$$\begin{aligned} h^a g^b &= h^{\hat{a}_0} \prod_{i=1}^{\ell} h^{\hat{a}_i m_i} \cdot g^{\hat{b}_0} \prod_{i=1}^{\ell} g^{\hat{b}_i m_i} \\ &= (h^{\hat{a}_0} g^{\hat{b}_0}) \cdot \prod_{i=1}^{\ell} (h^{\hat{a}_i m_i} g^{\hat{b}_i m_i}) \\ &= (h^{\hat{a}_0} g^{\hat{b}_0}) \cdot \prod_{i=1}^{\ell} \left(h^{\hat{a}_i} g^{\hat{b}_i}\right)^{m_i} \\ &= u_0 \cdot \prod_{i=1}^{\ell} {u_i}^{m_i} \\ &= H_\kappa(m) \end{aligned}$$

5.2 Waters-Signaturen

5.2.1 Konstruktion

- $Gen(1^k)$:

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

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

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

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

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

mail@nilslambertz.de
nilslambertz

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 $\kappa := (u_0, \ldots, u_\ell)$ (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

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 $\text{ggT}(e, \varphi(N)) = 1$
- 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 $x \in \mathbb{Z}_N : x^e \equiv y \mod N$

6.2 Textbook-RSA

6.2.1 Konstruktion

- $Gen(1^k)$:

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

- $Sign(sk, m)$:

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

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

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

mail@nilslambertz.de
nilslambertz

6.2.2 Korrektheit

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

6.2.3 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.

6.3 RSA Full-Domain-Hash

6.3.1 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**)

6.3.2 Konstruktion

- $Gen(1^k)$:

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

- $Sign(sk, m)$:

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

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

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

6.3.3 Korrektheit

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

mail@nilslambertz.de
nilslambertz

6.3.4 Sicherheit

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

6.4 RSA-PSS

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

6.4.1 Konstruktion

- $Gen(1^k)$:

$$\text{Wie bei Textbook-RSA}$$

- $Sign(sk, m)$:

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

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

$$\begin{aligned}&\text{Berechne } y = \sigma^e \mod N\\ &\text{Gib 1 aus gdw. } y \text{ eine gültige Codierung von } m \text{ ist}\end{aligned}$$

mail@nilslambertz.de
nilslambertz

Literatur

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