

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

1	Einführung	4
1.1	Ziel von Kryptographischen Verfahren	4
1.2	Informelle Definition von Signaturen	4
1.3	Digitale Signaturen	4
1.3.1	Definition	4
1.3.2	Correctness	4
1.4	Sicherheitsdefinitionen	4
1.4.1	Angreifermodelle	4
1.4.2	Angreiferziele	4
1.5	EUFCMA-Sicherheitsexperiment	5
1.5.1	Visualisierung: EUFCMA-Sicherheitsexperiment	5
1.5.2	Definition: Vernachlässigbarkeit	5
1.5.3	Definition: EUFCMA	5
1.6	EUFCMA-Sicherheitsexperiment	6
1.6.1	Visualisierung: EUFCMA-Sicherheitsexperiment	6
1.6.2	Definition: EUFCMA	6
1.7	Einmalsignaturen	6
1.7.1	Sicherheitsbegriffe für Einmalsignaturen	6
1.7.2	Beziehungen zwischen Sicherheitsdefinitionen	6
1.8	Perfekte Sicherheit	7
1.8.1	Warum müssen wir uns auf PPT-Angreifer beschränken?	7
1.8.2	Warum muss die Erfolgswahrscheinlichkeit des Angreifers nur vernachlässigbar sein?	7
1.9	Erweiterung des Nachrichtenraumes	7
1.9.1	Hashfunktionen	7
1.9.2	Kollisionsresistenz	7
1.9.3	Signatur mit unbeschränktem Nachrichtenraum (Hash-then-Sign)	7
2	q-mal Signaturen	8
2.1	Von EUFCMA-Sicherheit zu EUFCMA-Sicherheit	8
2.1.1	Transformation	8
2.2	Mehrmal-Signaturverfahren aus Einmalsignaturverfahren	8
2.2.1	Naiver Ansatz: q Schlüsselpaare	8
2.2.2	Zwischenschritt: Hashfunktion verwenden	9
2.2.3	Merkle-Bäume	10
2.3	Komprimieren des geheimen Schlüssels	11
2.3.1	Pseudozufallsfunktion	11
2.3.2	Schlüsselgenerierung	11
3	Chamäleon-Signaturen	12
3.1	Chamäleon-Hashfunktionen	12
3.1.1	Definition	12
3.1.2	Kollisionsresistenz	12
3.1.3	DLog-Annahme	13
3.1.4	Chamäleon-Hashfunktion basierend auf DLog	13
3.2	Chamäleon-Signaturen	14
3.2.1	Konstruktion	14

3.2.2	Visualisierung: EUF-CMA-Sicherheitsexperiment	14
3.3	Transformation von Chamäleon-Hashfunktion zu Einmalsignatur	15
3.4	EUF-CMA verstärken	15
3.4.1	Definition: sEUF-CMA	15
4	Pairings und BLS-Signaturen	16
4.1	Pairings	16
4.1.1	Definition	16
4.1.2	Typen von Pairing	16
4.1.3	Diffie-Hellman-Schlüsselaustausch	16
4.1.4	Joux 3-Parteien-Schlüsselaustausch	17
4.2	Boneh-Lynn-Shacham-Signaturen	17
4.2.1	Aggregierbarkeit	18
4.2.2	Batch-Verifikation	18
4.3	Computational-Diffie-Hellman-Problem	19
4.3.1	CDH-Problem	19
4.3.2	CDH-Annahme	19
4.4	Random-Oracle-Modell (ROM)	19
4.4.1	Das H-Orakel	19
4.4.2	Diskussion zum ROM	19
5	Waters-Signaturen	19
5.1	Programmierbare Hashfunktionen	19
5.1.1	Definition	19
5.1.2	Anforderungen an PHF	20
5.1.3	Waters Programmierbare Hashfunktion	20
5.2	Waters-Signaturen	21
5.2.1	Konstruktion	21
5.2.2	Korrektheit	22
5.2.3	Eigenschaften	22
6	RSA-basierte Signaturen	22
6.1	RSA-Problem und -Annahme	22
6.1.1	RSA-Problem	22
6.1.2	RSA-Annahme	22
6.2	Strong-RSA-Problem und -Annahme	22
6.2.1	Strong-RSA-Problem	22
6.2.2	Strong-RSA-Annahme	23
6.2.3	Unterschied zum normalen RSA-Problem und -Annahme	23
6.3	Textbook-RSA	23
6.3.1	Konstruktion	23
6.3.2	Korrektheit	23
6.3.3	Sicherheit	24
6.4	RSA Full-Domain-Hash	24
6.4.1	Idee	24
6.4.2	Konstruktion	24
6.4.3	Korrektheit	24
6.4.4	Sicherheit	24
6.5	RSA-PSS	25

6.5.1	Konstruktion	25
6.6	GHR-Signaturen	25
6.6.1	Konstruktion	25
6.6.2	Hashfunktionen für GHR-Signaturen	25
6.6.3	Hashfunktionen, die auf Primzahlen abbilden	26
6.6.4	Hashfunktionen, für die für alle m gilt, dass $\text{ggT}(h(m), \varphi(N)) = 1$	26
7	Message Authentication Codes	26
7.1	Grundlagen	26
7.1.1	Definition	26
7.1.2	Fixed- und variable-length MACs	26
7.1.3	Kanonische Verifikation	27
7.2	Sicherheitsbegriffe für MACs	27
7.2.1	strong MACs durch eindeutige Tags	27
7.2.2	Anmerkungen zu eindeutigen Tags	27
7.3	Konstruktion von MACs aus PRFs	27
7.3.1	Konstruktion eines fixed-length MAC	27
7.3.2	Erweiterung auf variable-length MAC	28
7.4	CBC-MAC	28
7.4.1	Konstruktion	28
7.4.2	Anmerkungen	28
7.4.3	Unterschiede zum CBC-Modus bei Verschlüsselung	29
7.5	MACs aus Hashfunktionen	29
7.5.1	Merkle-Damgård-Transformation	29
7.5.2	Secretly Keyed Hashfunctions	29
7.5.3	NMAC	30
7.5.4	HMAC	30

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:

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

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, \dots, m_q
 - Angreifer erhält **danach** pk und Signaturen $\sigma_1, \dots, \sigma_q$
3. **(adaptive) chosen-message attack (CMA)**
 - Angreifer erhält pk
 - Angreifer wählt dann (adaptiv) m_1, \dots, m_q und erhält Signaturen $\sigma_1, \dots, \sigma_q$
 - 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 Unforgeability (EUF)

- Angreifer kann Nachricht m beliebig wählen und diese signieren

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

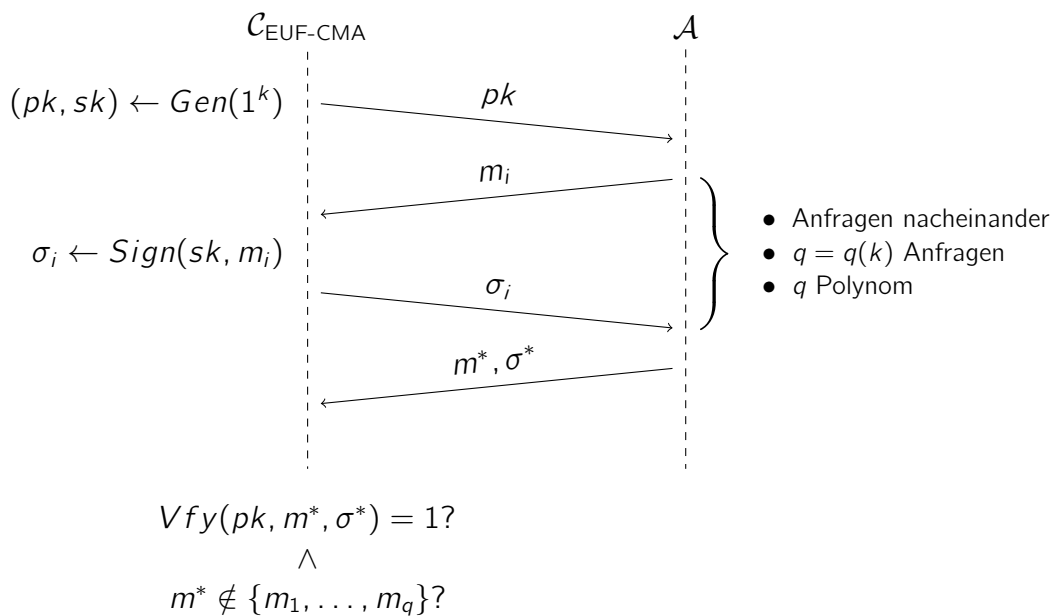
- EUF-CMA
- EUF-naCMA

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



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

1.5.2 Definition: Vernachlässigbarkeit

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

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

1.5.3 Definition: EUF-CMA

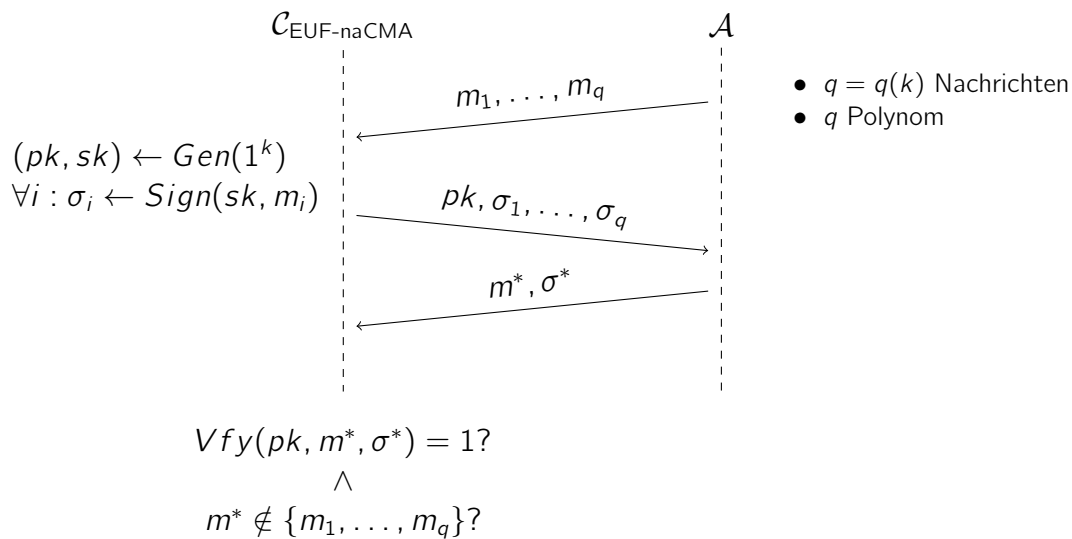
Ein digitales Signaturverfahren $\Sigma = (\text{Gen}, \text{Sign}, \text{Vfy})$ ist *EUF-CMA-sicher*, wenn für alle PPT \mathcal{A} gilt, dass

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

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** $m^* \notin \{m_1, \dots, m_q\}$

1.6.2 Definition: EUF-naCMA

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

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

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

EUF-naCMA \Leftarrow EUF-CMA

\Downarrow

\Downarrow

EUF-1-naCMA \Leftarrow EUF-1-CMA

Beweis im Skript.

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

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

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

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

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 $H : \{0, 1\}^* \rightarrow \mathcal{M}$

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

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

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

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
- $Gen(1^k)$:

$$(pk_i, sk_i) \leftarrow Gen^{(1)}(1^k) \text{ für alle } i \in \{1, \dots, q\}$$

$$pk := H(pk_1, \dots, pk_q)$$

$$sk := (sk_1, \dots, sk_q, pk_1, \dots, pk_q, st = 1)$$

- $Sign(sk, m)$:

$$i := st$$

$$\sigma_i \leftarrow Sign^{(1)}(sk_i, m)$$

$$\sigma \leftarrow (\sigma_i, i, pk_1, \dots, pk_q)$$

$$st := st + 1$$

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

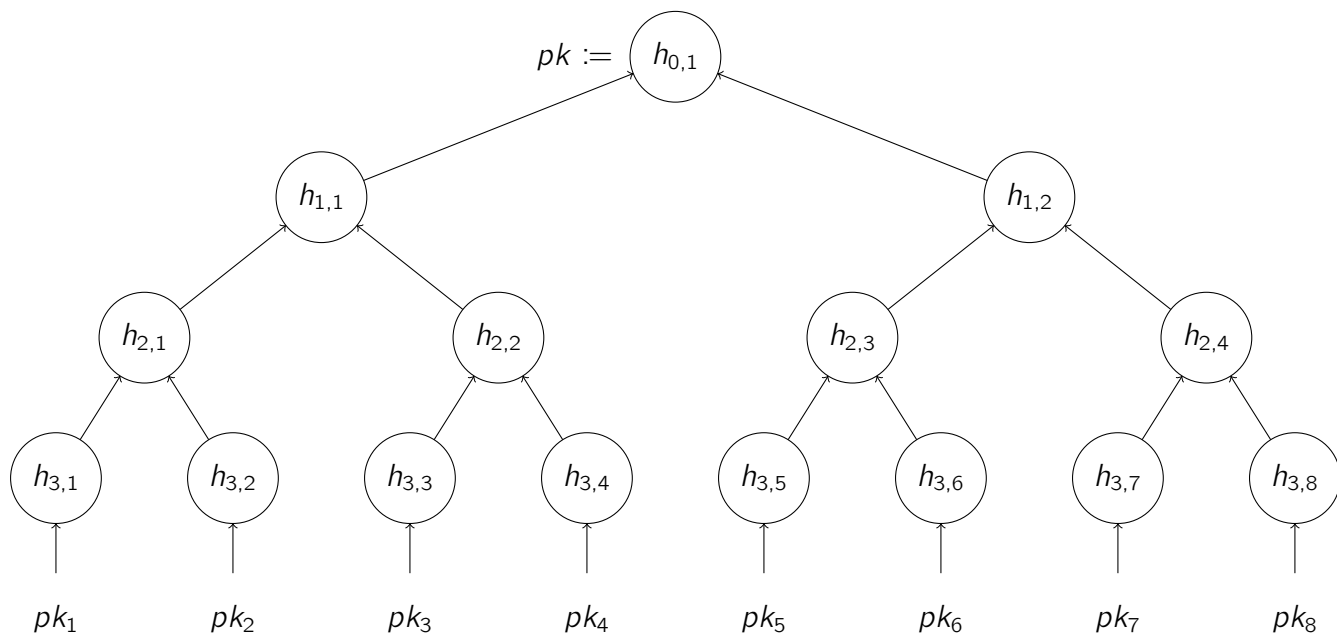
$$Vfy^{(1)}(pk_i, m, \sigma_i) \stackrel{?}{=} 1 \text{ und } H(pk_1, \dots, pk_q) \stackrel{?}{=} pk$$

Eigenschaften bezogen auf Signaturanzahl (q):

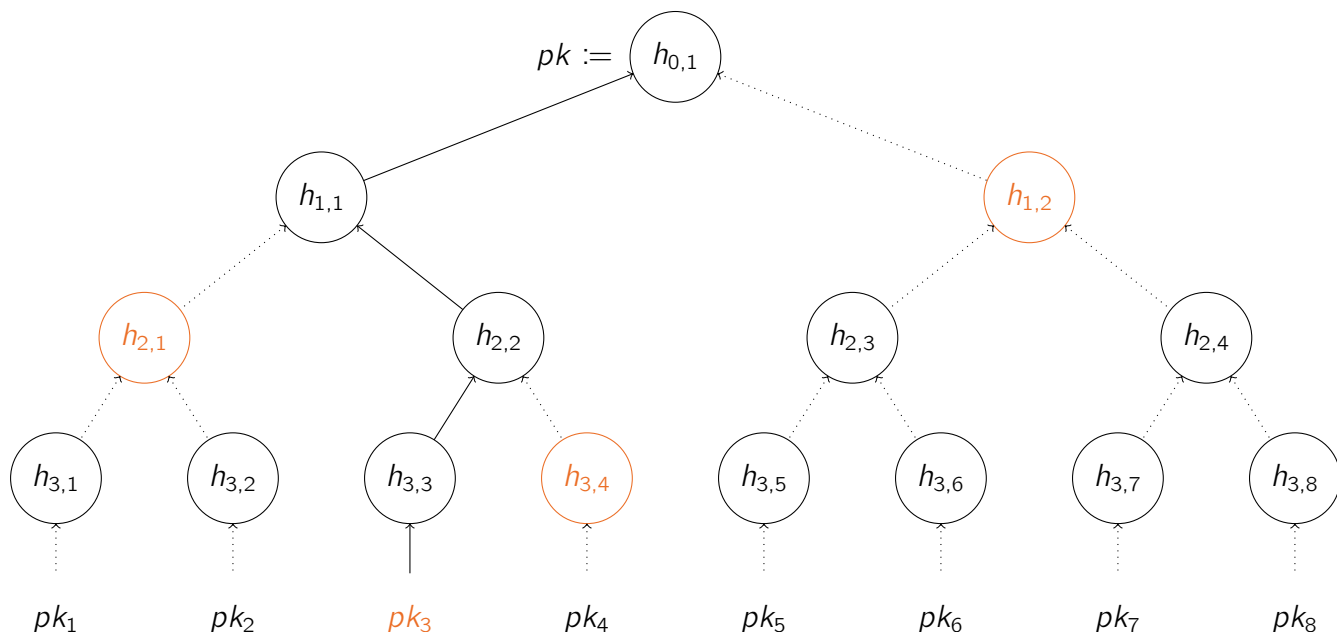
- $|pk| \in \Theta(1)$
- $|sk| \in \Theta(q)$
- $|\sigma| \in \Theta(q)$

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, \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.

- $Gen(1^k)$:

$$(pk_i, sk_i) \leftarrow Gen^{(1)}(1^k) \text{ für alle } i \in \{1, \dots, q\}$$

$$pk := \text{Baum-Hash}(pk_1, \dots, pk_q)$$

$$sk := (sk_1, \dots, sk_q, pk_1, \dots, pk_q, st = 1)$$

- $Sign(sk, m)$:

$i := st$

$\sigma_i \leftarrow Sign^{(1)}(sk_i, m)$

$\sigma \leftarrow (\sigma_i, i, pk_i, \text{Co-Pfad})$

$st := st + 1$

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

Berechne Wurzel h'

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

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:

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

2.3.2 Schlüsselgenerierung

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

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

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

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

Damit ist die bishere Schlüsselberechnung äquivalent zu folgender:

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

Mit **echt zufälliger** Funktion F also:

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

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

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

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 $(\text{Gen}_{ch}, \text{TrapColl}_{ch})$:

- $\text{Gen}_{ch}(1^k)$ gibt (ch, τ) aus:
 - ch ist eine Funktion $ch: \mathcal{M} \times \mathcal{R} \rightarrow \mathcal{N}$
 - * \mathcal{M} Nachrichtenraum
 - * \mathcal{R} Zufallsraum
 - * \mathcal{N} Zielraum
 - τ ist eine **Trapdoor** (Falltür)
- $\text{TrapColl}_{ch}(\tau, m, r, m')$ 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 $CH = (\text{Gen}_{ch}, \text{TrapColl}_{ch})$ ist **kollisionsresistent**, falls für alle PPT \mathcal{A} gilt, dass

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

für eine im Sicherheitsparameter k vernachlässigbare Funktion negl .

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:

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

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

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

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.

3.2 Chamäleon-Signaturen

3.2.1 Konstruktion

Gegeben sind

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

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**):

$$r \xleftarrow{\$} \mathcal{R}$$

$$y := ch(m, r)$$

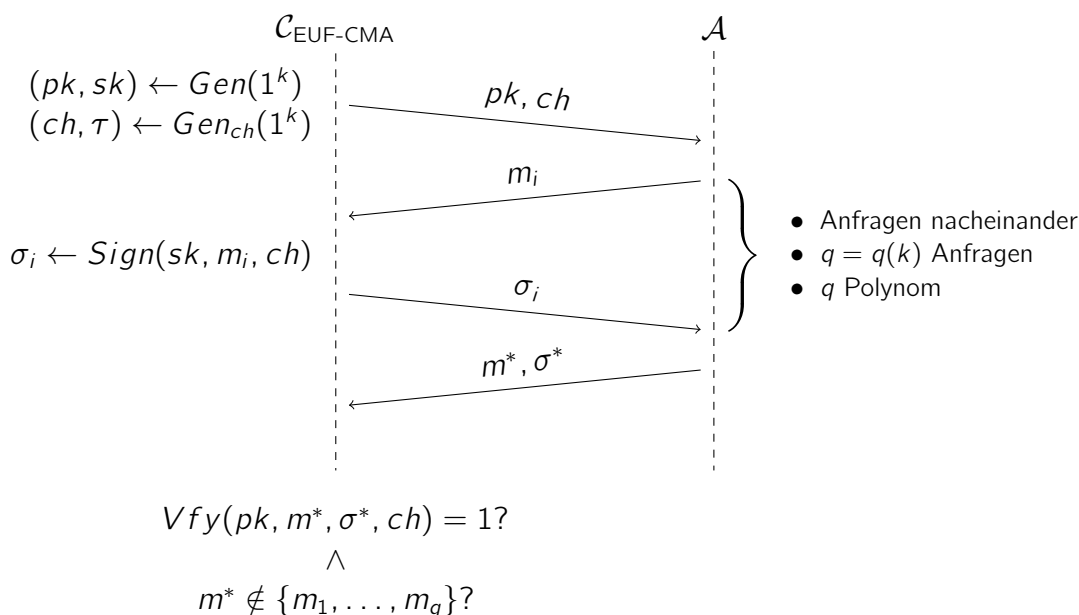
$$\sigma' := Sign'(sk, y)$$

$$\sigma := (\sigma', r)$$

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

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

3.2.2 Visualisierung: EUF-CMA-Sicherheitsexperiment



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

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 $\Sigma = (Gen, Sign, Vfy)$:

- $Gen(1^k)$:

$$(ch, \tau) \leftarrow Gen_{ch}(1^k)$$

$$(\tilde{m}, \tilde{r}) \xleftarrow{\$} \mathcal{M} \times \mathcal{R}$$

$$c := ch(\tilde{m}, \tilde{r})$$

$$pk := (ch, c)$$

$$sk := (\tau, \tilde{m}, \tilde{r})$$

- $Sign(sk, m)$:

$$r := TrapColl_{ch}(\tau, \tilde{m}, \tilde{r}, m)$$

$$\sigma := r$$

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

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

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 $m^* \notin \{m_1, \dots, m_q\}$ 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

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

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

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

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:

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

- 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**

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

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

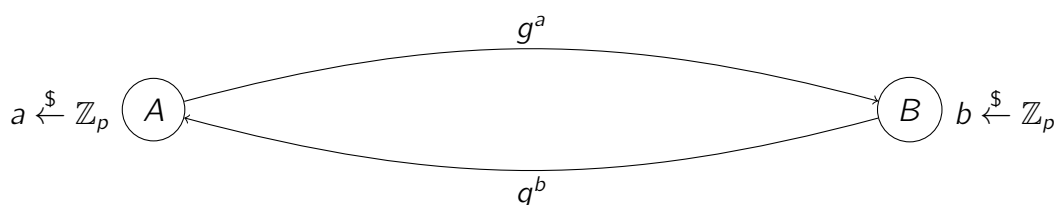
$$\psi : \mathbb{G}_1 \rightarrow \mathbb{G}_2$$

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

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

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



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

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

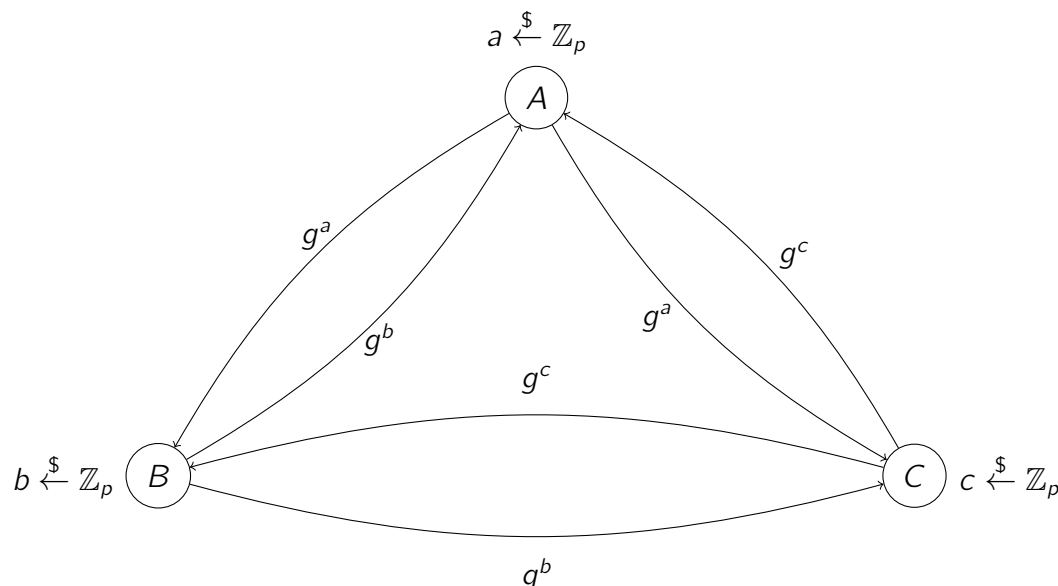
Ablauf:

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.

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



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

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

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 $\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$
- Hashfunktion $H : \{0, 1\}^* \rightarrow \mathbb{G}$

Konstruktion:

- $Gen(1^k)$:

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

- $Sign(sk, m)$:

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

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

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

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 $(m_1, \sigma_1), \dots, (m_n, \sigma_n)$ müssen nicht alle Signaturen $\sigma_1, \dots, \sigma_n$ mitgeschickt werden, sondern es kann eine aggregierte Signatur σ_{Agg} berechnet werden. Die Gültigkeit kann dann mit $Vfy(pk_1, \dots, pk_n, m_1, \dots, m_n, \sigma_{\text{Agg}}) \stackrel{?}{=} 1$ 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 $H(m_i)^{x_i}$
- Aggregator 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:

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

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 $(m_1, \sigma_1), \dots, (m_n, \sigma_n)$ auf einmal verifizieren können, die Lösung dafür ist die **Batch-Verifikation**:

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

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:

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

für eine im Sicherheitsparameter k vernachlässigbare Funktion negl .

4.4 Random-Oracle-Modell (ROM)

Wir betrachten eine idealisierte Hashfunktion $H : \mathcal{D} \rightarrow \mathcal{R}$, 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 $H_\kappa : \{0, 1\}^\ell \rightarrow \mathbb{G}$ 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

- $\text{Gen}(g) \rightarrow \kappa$ (Schlüsselerzeugung)
- $\text{Eval}(\kappa, m) \rightarrow H_\kappa(m)$ (deterministische Auswertung)
- $\text{TrapGen}(g, h) \rightarrow (\kappa, \tau)$ (Schlüsselerzeugung mit Trapdoor)
- $\text{TrapEval}(\tau, m) \rightarrow (a, b)$ 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}
 - $m_1^*, \dots, m_v^* \in \{0, 1\}^\ell$
 - $m_1, \dots, m_w \in \{0, 1\}^\ell$ (alle m_i^* und m_j paarweise verschieden)
 gilt

$$\Pr \left[\begin{matrix} a_i^* = 0 & \forall i = 1, \dots, v \\ a_j^* = 0 & \forall j = 1, \dots, w \end{matrix} \right] \geq \gamma$$

wobei

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

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

5.1.3 Waters Programmierbare Hashfunktion

- $Gen(g)$:

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

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

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

($m_i \in \{0, 1\}$ ist das i -te Bit von m)

Intuition: $H_\kappa(m)$ ist das Produkt von u_0 und aller u_i mit $m_i = 1$

- $TrapGen(g, h)$:

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

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

$$a = \hat{a}_0 + \sum_{i=1}^{\ell} m_i \hat{a}_i \quad \text{und}$$

$$b = \hat{b}_0 + \sum_{i=1}^{\ell} m_i \hat{b}_i$$

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 $\text{ggT}(e, \varphi(N)) = 1$
- Dann existiert $d \in \mathbb{N}$ mit $e \cdot d \equiv 1 \pmod{\varphi(N)}$
- Für $x \in \mathbb{Z}_N$ gilt dann auch $x^{e \cdot d} \equiv x \pmod{N}$

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

6.1.2 RSA-Annahme

Für alle PPT \mathcal{A} gilt:

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

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 $x^e \equiv y \pmod{N}$

6.2.2 Strong-RSA-Annahme

Für alle PPT \mathcal{A} gilt:

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

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

- $\text{Gen}(1^k)$:

Ziehe zufällige Primzahlen P, Q

$$N := P \cdot Q$$

Wähle $e > 2$ mit $\text{ggT}(e, \varphi(N)) = 1$

$$d := e^{-1} \pmod{\varphi(N)}$$

$$pk := (N, e)$$

$$sk := d$$

- $\text{Sign}(sk, m)$:

$$\sigma := m^d \pmod{N}$$

- $\text{Vfy}(pk, m, \sigma)$:

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

6.3.2 Korrektheit

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

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 $m^* := (\sigma^*)^e \bmod N$
- 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 $\sigma_3 := \sigma_1 \sigma_2 \bmod N$ gültig für $m_3 := m_1 m_2 \bmod N$
- da $\sigma_3^e \equiv (\sigma_1 \sigma_2)^e \equiv \sigma_1^e \sigma_2^e \equiv m_1 m_2 \equiv m_3 \bmod N$

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 $H := \{0, 1\}^* \rightarrow \mathbb{Z}_N$ 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
 $pk := (N, e, H)$

- $Sign(sk, m)$:

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

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

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

6.4.3 Korrektheit

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

6.4.4 Sicherheit

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

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 $H := \{0, 1\}^* \rightarrow \mathbb{P}$ 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

$$pk := (N, s, h)$$

$$sk := \varphi(N) = (P - 1)(Q - 1)$$

- $Sign(sk, m)$:

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

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

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

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

6.6.3 Hashfunktionen, die auf Primzahlen abbilden

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

- Sei $H : \{0, 1\}^* \rightarrow \{0, 1\}^\ell$ eine kollisionsresistente Hashfunktion
- Annahme: H bildet "einigermaßen gleichverteilt" nach $\{0, 1\}^\ell$ ab
- Definiere $h : \{0, 1\}^* \rightarrow \mathbb{P} \cap \{0, 1\}^\ell$ 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 $\text{ggT}(h(m), \varphi(N)) = 1$

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 (**MACs**) 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:

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

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 MACs** genannt.

7.1.3 Kanonische Verifikation

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

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

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

7.2 Sicherheitsbegriffe für MACs

Die Sicherheitsbegriffe **EUFCMA** und **sEUFCMA** 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 .

sEUFCMA-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 EUFCMA-sicherer MAC Π mit eindeutigen Tags ist sEUFCMA-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 $PRF : \{0, 1\}^k \times \{0, 1\}^k \rightarrow \{0, 1\}^k$ 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)

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

$$\tilde{t} \stackrel{?}{=} t$$

7.3.2 Erweiterung auf variable-length MAC

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

Es sei $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ 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** (Cipher Block Chaining Message Authentication Code) ist verwandt mit dem CBC-Modus für symmetrische Verschlüsselung. Es handelt sich um einen **fixed-length MAC**, der auf **PRFs** basiert.

7.4.1 Konstruktion

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

- $Gen(1^k)$:

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

- $Sign(\kappa, m)$:

$$t_0 := 0^k$$

for i from 1 to ℓ :

$$t_i \leftarrow PRF(\kappa, t_{i-1} \oplus m_i)$$

return $t := t_\ell$

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

if $|m| \neq \ell \cdot k$:

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

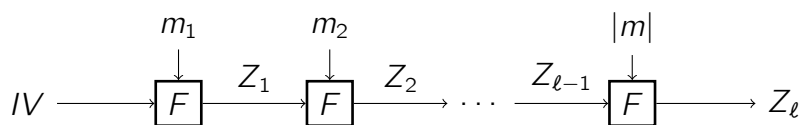
Wir wollen nun MACs basierend auf Hashfunktionen konstruieren. Hashfunktionen sind definiert als $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ und können mit der **Merkle-Damgård-Transformation** konstruiert werden:

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

7.5.1 Merkle-Damgård-Transformation

Gegeben ist eine Kompressionsfunktion $F : \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$. Konstruiere nun eine Hashfunktion $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ 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 $\ell = \lceil \frac{|m|}{n} \rceil + 1$
5. $\forall i \in \{1, \dots, \ell\} : Z_i := F(Z_{i-1} \parallel m_i)$
6. $H(m) := Z_\ell$

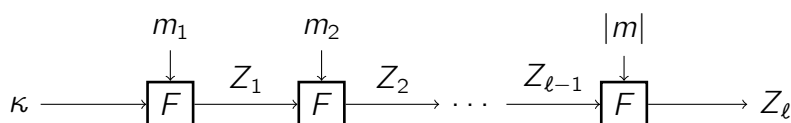


Eigenschaften:

- **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.

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-Hash **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 $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ 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).

Literatur

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