Digitale Signaturen

Tibor Jager
tibor.jager@upb.de
Universität Paderborn
Fachgruppe IT-Sicherheit

Letzte Aktualisierung: 23. Februar 2017

Ich bedanke mich bei Florian Böhl, Benny Fuhry, Kai Gellert, Felix Grün, Gunnar Hartung, Eduard Hauck, Max Hoffmann, Jan Holz, Björn Kaidel, Eike Kiltz, Evgheni Kirzner, Jessica Koch, Sebastian Lauer, Julia Rohlfing, Christoph Striecks und Sun Jing für hilfreiche Kommentare.

Inhaltsverzeichnis

Kapitel 1

Einführung

Ein Sender möchte eine Nachricht an einen Empfänger übertragen, und zwar so, dass der Empfänger bei Erhalt der Nachricht zwei Dinge verifizieren kann:

1. *Integrität der Nachricht*: Der Empfänger kann sicher sein, dass die Nachricht auf dem Transportweg nicht verändert wurde. Weder böswillig (also durch einen Angreifer, der die Nachricht gezielt verändert), noch durch Zufall (zum Beispiel durch einen Übertragungsfehler).

2. *Authentizität der Nachricht*: Der Empfänger kann feststellen, ob die Nachricht tatsächlich von einem bestimmten Absender stammt.

Falls die Nachricht auf Papier gedruckt ist, gibt es eine klassische Lösung für dieses Problem. Jeder Sender hat eine eindeutige Unterschrift, welche

1. nur vom Sender erzeugt werden kann, und

2. dem Empfänger bekannt ist.

Bei Erhalt der Nachricht überprüft der Empfänger, ob das erhaltene Papier eine unveränderte Nachricht und die Unterschrift des Senders trägt. Falls beides zutrifft, wird die signierte Nachricht als authentisch und integer akzeptiert. Häufig wollen wir jedoch nicht nur Nachrichten in Papierform übertragen, sondern auch digitale Nachrichten. Digitale Nachrichten sind Bitstrings, also Elemente aus der Menge $\{0,1\}^*$. Die klassische Lösung für Nachrichten auf Papier ist hier leider nicht anwendbar. Daher verwenden wir *digitale Signaturverfahren* (oder kurz: *digitale Signaturen*).

Wofür benötigen wir Unterschriften unter digitalen Nachrichten? Es gibt zahlreiche Beispiele für Anwendungen von digitalen Signaturen, und einige davon benutzen wir nahezu täglich.

Digitale Zertifikate im Internet. Digitale Zertifikate sind digital signierte kryptographische Schlüssel. Solche Zertifikate stellen einen wichtigen Grundbaustein von Sicherheitsinfrastrukturen im Internet, wie zum Beispiel so genannten *Public-Key Infrastrukturen* (PKI), dar. Wir kommen mit solchen Zertifikaten nahezu täglich in Kontakt, zum Beispiel wenn wir mit dem Webbrowser eine gesicherte Internetseite aufrufen, deren Adresse mit `https://` beginnt. Denn dann kommuniziert der Webbrowser mit dem Server

über das TLS Protokoll [DA99], welches den Kommunikationspartner über ein Zertifikat authentifiziert.

Elektronischer Zahlungsverkehr. Wenn wir eine elektronische Geldtransaktion tätigen, beispielsweise indem wir mit einer Kreditkarte bezahlen, dann wird auch hier die Authentizität und Integrität von Nachrichten durch digitale Signaturen sichergestellt. Insbesondere sind digitale Signaturen ein wichtiger Sicherheitsbaustein des EMV (Europay/Mastercard/VISA) Frameworks für sicheres Kreditkarten-Payment. Auch wenn wir mit der EC-Karte am Geldautomaten Geld abheben, wird die Authentizität der Karte durch ein Protokoll sichergestellt, für das auch digitale Signaturen verwendet werden.

Betriebssystem-Updates. Wenn wir ein Update für unser Betriebssystem, egal ob Windows, Linux oder MacOS, aus dem Internet herunterladen um es zu installieren, dann ist es wichtig, dass die Authentizität des Updates überprüft wird. Diese Aufgabe nimmt uns häufig die automatische Update- oder Paketverwaltung des Betriebssystems ab. Zur Sicherstellung der Authentizität der heruntergeladenen Software werden digitale Signaturen verwendet.

Elektonischer Personalausweis. Auch der neue „elektronische Personalausweis“, der derzeit in Deutschland eingeführt wird, enthält ein Schlüsselpaar für ein digitales Signaturverfahren. Ein Feature, mit dem für den elektronischen Personalausweis geworben wird, ist die Möglichkeit, seine Identität mittels digitaler Signaturen auch im Internet nachweisen zu können — beispielsweise beim Online-Shopping oder um Jugendschutzbestimmungen zu erfüllen.

Kryptographische Theorie. Digitale Signaturen haben nicht nur praktische Anwendungen, sondern sind auch für die kryptographische Theorie von wesentlicher Bedeutung. Insbesondere sind sie ein wichtiger Baustein zur Konstruktion anderer Kryptosyteme. Dies umfasst zum Beispiel *public key* Verschlüsselungsverfahren mit starken Sicherheitseigenschaften wie *adaptive chosen-ciphertext* (CCA) Sicherheit oder *simulation-sound zero-knowledge* Beweissysteme [Sah99, CHK04, Lin03, Gro06].

Was genau sind digitale Signaturverfahren? Digitale Signaturverfahren sind ein kryptographisches Äquivalent zu klassischen, handgeschriebenen Signaturen.

- Der Sender hat ein (in der Praxis eindeutiges) kryptographisches Schlüsselpaar (pk, sk).

  sk wird vom Sender benutzt, um eine Unterschrift zu erzeugen. Dieser Schlüssel ist geheim, also nur dem Sender bekannt. Wir sagen, sk ist der *geheime Schlüssel* („secret key“).

  pk ist öffentlich bekannt, insbesondere kennt der Empfänger pk. Dieser Schlüssel wird verwendet, um digitale Signaturen zu verifizieren. Wir sagen, pk ist der *öffentliche Schlüssel* („*public key*“).

- Um eine Nachricht m zu signieren, berechnet der Sender eine Signatur σ mit Hilfe des geheimen Schlüssels:

  $$\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$$

- Um zu verifizieren, ob σ eine gültige Signatur für eine Nachricht m ist, berechnet der Empfänger eine Funktion $\mathsf{Vfy}(pk, m, \sigma)$.

Was sind digitale Signaturverfahren nicht? Ein häufiges Missverständnis ist, dass digitale Signaturen mit *public key* Verschlüsselungsverfahren verwandt sind. Oft liest man in Büchern oder Vorlesungsskripten:

> *„Digital signieren bedeutet, mit dem geheimen Schlüssel zu verschlüsseln."*

Das stimmt *nicht*. Es trifft auf nahezu alle Signatur- und Verschlüsselungsverfahren nicht zu. Das „Lehrbuch"-RSA Verfahren zur Verschlüsselung und digitalen Signatur scheint das einzige Beispiel zu sein, auf das die obige Aussage zutrifft.

1.1 Definition von digitalen Signaturverfahren

Wir definieren digitale Signaturverfahren wie folgt:

Definition 1. Ein digitales Signaturverfahren ist ein Tripel $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ von (möglicherweise probabilistischen) Polynomialzeit-Algorithmen, wobei

- Algorithmus $\mathsf{Gen}(1^k)$ erhält als Eingabe einen Sicherheitsparameter (*„security parameter"*) k (in unärer Notation) und gibt ein Schlüsselpaar (pk, sk) aus.
- Algorithmus $\mathsf{Sign}(sk, m)$ erhält als Eingabe den Sicherheitsparameter, den *secret key* sk und eine Nachricht $m \in \{0,1\}^n$. Er gibt eine Signatur σ aus.
- Algorithmus $\mathsf{Vfy}(pk, m, \sigma)$ erhält als Eingabe den Sicherheitsparameter, den *public key* pk, eine Nachricht m und eine Signatur σ. Er gibt 0 oder 1 aus.

  Dabei bedeutet 1, dass σ als Signatur für Nachricht m und *public key* pk akzeptiert wird. 0 bedeutet, dass die Signatur nicht akzeptiert wird.

Eigenschaften eines Signaturverfahrens: Correctness und Soundness. Wir erwarten von einem digitalen Signaturverfahren, dass es zumindest die zwei Eigenschaften *„Correctness"* und *„Soundness"* erfüllt.

Correctness. Correctness bedeutet im Wesentlichen:

> *„Das Verfahren funktioniert."*

Für digitale Signaturen bedeutet dies: wenn wir

1. mittels $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^k)$ ein Schlüsselpaar erzeugen, und dann
2. für eine beliebige Nachricht m eine Signatur $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$ erstellen, und dann
3. den Verifikationsalgorithmus $\mathsf{Vfy}(pk, m, \sigma)$ ausführen,

dann wird die Signatur stets akzeptiert, das heißt, es ist $\mathsf{Vfy}(pk, m, \sigma) = 1$. Dies muss gelten für alle möglichen Nachrichten m und für alle (möglicherweise bis auf einen vernachlässigbar kleinen Anteil) existierenden Schlüsselpaare (pk, sk).

Soundness. Soundness bedeutet im Wesentlichen:

„Das Verfahren ist sicher."

Das bedeutet, es soll kein effizienter Algorithmus (=Angreifer) existieren, der die gewünschte Sicherheitseigenschaft des Verfahrens mit nicht-vernachlässigbarer Wahrscheinlichkeit bricht. Was genau die „gewünschte Sicherheitseigenschaft" ist, hängt von dem Anwendungsbereich des Signaturverfahrens ab. Wir werden später verschiedene sinnvolle Sicherheitsdefinitionen für digitale Signaturen kennenlernen.

1.2 Sicherheit digitaler Signaturverfahren

Der Begriff „Sicherheit" ansich ist recht einfach, jedoch auch ein wenig schwammig. Um später die Sicherheit eines Verfahrens im mathematischen Sinne beweisen zu können, müssen wir diesen Begriff präzisieren.

1.2.1 Angreifermodell und Angreiferziel

Um präzise zu beschreiben, was „Sicherheit" für ein kryptographisches Protokoll (z.B. ein Signaturverfahren) bedeutet, müssen wir zwei Dinge angeben:

1. Ein *Angreifermodell*, welches beschreibt über welche „Fähigkeiten" der Angreifer verfügt. Im Kontext von Signaturen kann dies zum Beispiel sein:

   (a) Der Angreifer erhält nur den *public key* pk als Eingabe. Er hat jedoch keinen Zugriff auf gültige Signaturen, die vom ehrlichen Sender erstellt wurden. Sein Ziel ist es, eine gültige Signatur zu fälschen.

   Diese Form von Angriffen nennt man *no-message attacks* (NMA). Dies ist kein besonders starkes Angreifermodell, da angenommen wird, dass der Angreifer keinen Zugriff auf gültige Signaturen hat. Diese könnten ihm jedoch dabei helfen, eine neue Signatur zu fälschen.

   (b) Der Angreifer darf eine Liste $(m_1, \ldots, m_q)$ von Nachrichten auswählen. Danach erhält er den *public key* pk sowie Signaturen $(\sigma_1, \ldots, \sigma_q)$ für die gewählten Nachrichten. Das Ziel des Angreifers ist, eine gültige Signatur für eine *neue* Nachricht m^* zu fälschen.

   Diese Form von Angriffen nennt man *non-adaptive chosen-message attacks* (naCMA). Dieses Angreifermodell zieht in Betracht, dass ein Angreifer Zugang zu gültigen Signaturen für bestimmte Nachrichten haben könnte.

   (c) Der Angreifer erhält den *public key* als Eingabe. Danach kann er den Sender dazu überlisten, dass er Signaturen $(\sigma_1, \ldots, \sigma_q)$ für vom Angreifer gewählte Nachrichten $(m_1, \ldots, m_q)$ ausstellt. Er darf dabei die Nachrichten m_i *adaptiv* wählen, also beispielsweise abhängig vom pk oder von bereits erhaltenen Signaturen σ_j für $j < i$. Das Ziel des Angreifers ist, eine gültige Signatur für eine neue Nachricht zu fälschen.

   Diese Form von Angriffen nennt man *adaptive chosen-message attacks* (CMA). Dieses Modell zieht in Betracht, dass ein Angreifer auch die Benutzer des Signaturverfahrens dazu verleiten könnte, bestimmte Nachrichten zu signieren, die der

Angreifer abhängig vom pk und von bereits beobachteten signierten Nachrichten gewählt hat.

2. Weiterhin müssen wir ein *Angreiferziel* definieren: was muss ein Angreifer tun, um die Sicherheit des Signaturverfahrens zu brechen? Im Kontext von Signaturen kann dies zum Beispiel sein:

   (a) Der Angreifer muss eine gültige Signatur für eine *vorgegebene* Nachricht m^* fälschen. Dabei ist m^* zufällig gewählt. Er muss eine gültige Signatur σ^* für genau diese Nachricht ausgeben, um das Verfahren zu brechen.

   Im Falle von chosen-message attacks muss man natürlich ausschließen, dass der Angreifer den Sender überlisten kann, eine Signatur für m^* auszugeben.

   Ein Signaturverfahren das sicher gegen solche Angreifer ist, nennt man „sicher gegen universelle Fälschungen“ (*universal unforgeable*, UUF).

   (b) Der Angreifer muss eine gültige Signatur für eine *beliebige* (also vom Angreifer selbst gewählte) Nachricht ausgeben.

   Im Falle von chosen-message attacks muss man natürlich fordern, dass der Angreifer eine Signatur für eine neue Nachricht ausgibt, die er nicht vom Sender erhalten hat.

   Ein Signaturverfahren das sicher gegen solche Angreifer ist nennt man „sicher gegen existentielle Fälschungen“ (*existential unforgeable*, EUF).

Sicherheitsdefinitionen. Eine Sicherheitsdefinition (*„security notion“*) ergibt sich nun aus der Kombination von einem Angreifermodell mit einem Angreiferziel.

Sicherheitsdefinition = Angreifermodell + Angreiferziel

Die wichtigste gängige Sicherheitsdefinition für digitale Signaturverfahren ist beispielsweise *existential unforgeability under adaptive chosen-message attacks* (EUF-CMA), welche von Goldwasser, Micali und Rivest eingeführt wurde [GMR85, GMR88]. Diese ergibt sich aus der Kombination von 1.(c) mit 2.(b).

Auf diese Art ergeben sich viele weitere Sicherheitsdefinitionen durch andere Kombinationen von Angreifermodellen mit Angreiferzielen, siehe Tabelle 1.1. Glücklicherweise werden wir in dieser Vorlesung nicht alle diese Sicherheitsdefinitionen benötigen. Es ist wohl nun klar, dass es viele verschiedene Möglichkeiten gibt, „Sicherheit“ digitaler Signaturen zu definieren.

	1.(a)	1.(b)	1.(c)
2.(a)	UUF-NMA	UUF-naCMA	UUF-CMA
2.(b)	EUF-NMA	EUF-naCMA	EUF-CMA

Tabelle 1.1: Verschiedene Sicherheitsdefinitionen.

Neben den bislang vorgestellten Möglichkeiten finden sich in der Literatur noch zahlreiche Weitere. In dieser Vorlesung werden wir uns hauptsächlich mit der Sicherheit im Sinne von *existential unforgeability under adaptive/non-adaptive chosen message attacks*, also EUF-CMA und EUF-naCMA, befassen. An einigen Stellen werden wir noch Varianten dieser Sicherheitsdefinitionen kennenlernen.

1.2.2 Sicherheitsexperimente

Wir wollen später beweisen, dass ein gegebenes Signaturverfahren „sicher“ ist (unter bestimmten Annahmen). Dazu ist es hilfreich, die Angreifermodelle und Angreiferziele noch genauer zu beschreiben. Eine elegante Technik zur präzisen Beschreibung einer Sicherheitsdefinition ist die Angabe eines Sicherheitsexperiments.

An einem Sicherheitsexperiment nehmen zwei Parteien teil:

1. Der Angreifer $\mathcal{A}$.
2. Ein „Challenger“ („Herausforderer“) $\mathcal{C}$.

Im Experiment „spielt“ der Angreifer ein Spiel gegen den Challenger. Er gewinnt das Spiel, wenn er die Sicherheit des Signaturverfahrens „bricht“.

Das EUF-CMA Sicherheitsexperiment. Am besten lässt sich die Idee von Sicherheitsexperimenten an einem Beispiel erklären, siehe Abbildung 1.1. Das EUF-CMA Sicherheitsexperiment mit Angreifer $\mathcal{A}$, Challenger $\mathcal{C}$ und Signaturverfahren $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ läuft wie folgt ab:

1. Der Challenger $\mathcal{C}$ generiert ein Schlüsselpaar $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^k)$. Der Angreifer erhält pk.
2. Nun darf der Angreifer $\mathcal{A}$ beliebige Nachrichten $m_1, \ldots, m_q$ vom Challenger signieren lassen.

   Dazu sendet er Nachricht m_i an den Challenger. Dieser berechnet $\sigma_i \xleftarrow{\$} \mathsf{Sign}(sk, m)$ und antwortet mit σ_i.

   Dieser Schritt kann vom Angreifer beliebig oft wiederholt werden. Wenn wir Angreifer mit polynomiell beschränkter Laufzeit betrachten, ist $q = q(k)$ üblicherweise ein Polynom im Sicherheitsparameter.
3. Am Ende gibt $\mathcal{A}$ eine Nachricht m^* mit Signatur σ^* aus. Er „gewinnt“ das Spiel, wenn

   $$\mathsf{Vfy}(pk, m^*, \sigma^*) = 1 \qquad \text{und} \qquad m^* \notin \{m_1, \ldots, m_q\}.$$

   $\mathcal{A}$ gewinnt also, wenn σ^* eine gültige Signatur für m^* ist, und er den Challenger $\mathcal{C}$ nicht nach einer Signatur für m^* gefragt hat. Die zweite Bedingung ist natürlich notwendig, um triviale Angreifer auszuschließen.

Definition 2. Wir sagen, dass $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ sicher ist im Sinne von EUF-CMA, falls für alle PPT Angreifer $\mathcal{A}$ im EUF-CMA-Experiment gilt, dass

$$\Pr[\mathcal{A}^{\mathcal{C}}(pk) = (m^*, \sigma^*) : \mathsf{Vfy}(pk, m^*, \sigma^*) = 1 \wedge m^* \notin \{m_1, \ldots, m_q\}] \leq \mathsf{negl}(k)$$

für eine vernachlässigbare Funktion negl im Sicherheitsparameter.

Remark 3. Zur Erinnerung, eine Funktion $\mathsf{negl} : \mathbb{N} \to [0, 1]$ ist *vernachlässigbar*, falls es für jede Konstante $c \geq 0$ eine Zahl k_c gibt, sodass für alle $k > k_c$ gilt dass

$$\mathsf{negl}(k) \leq \frac{1}{k^c}.$$

Anders ausgedrückt, $\mathsf{negl}(k)$ ist eine vernachlässigbare Funktion, wenn sie schneller gegen 0 konvergiert als das Inverse jedes Polynoms in k, also $\mathsf{negl}(k) = o(1/\mathsf{poly}(k))$.

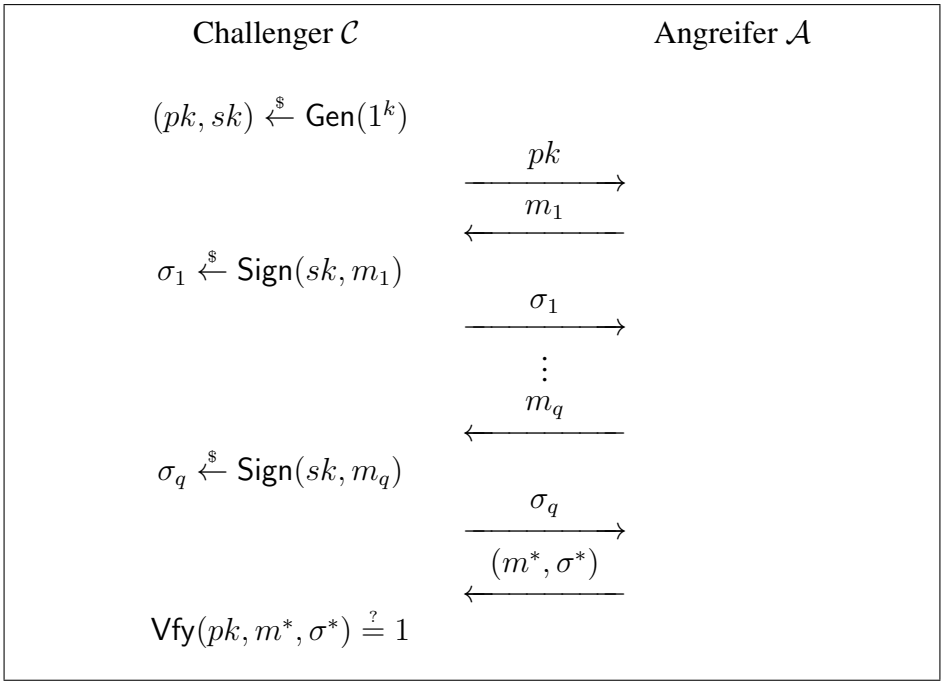

Abbildung 1.1: Das EUF-CMA Sicherheitsexperiment.

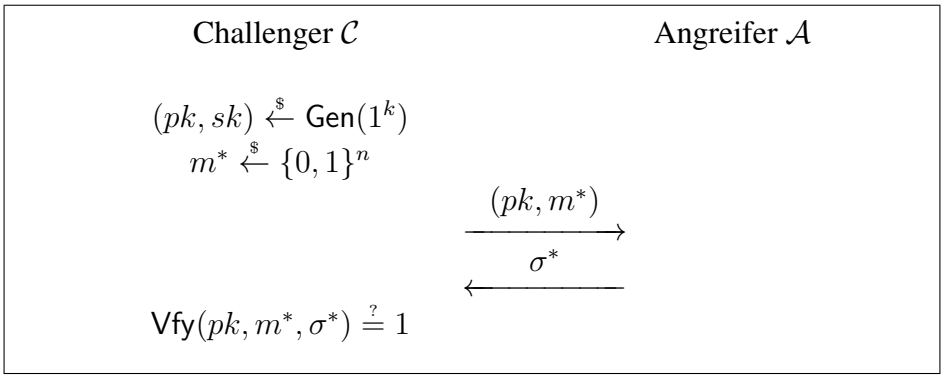

Abbildung 1.2: Das UUF-NMA Sicherheitsexperiment.

Noch ein Beispiel: Das UUF-NMA Sicherheitsexperiment. Das UUF-NMA Experiment läuft wie folgt ab:

1. Der Challenger $\mathcal{C}$ generiert ein Schlüsselpaar $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^k)$. Ausserdem wählt er eine zufällige Nachricht $m^* \xleftarrow{\$} \{0,1\}^n$. Der Angreifer erhält (pk, m^*).
2. Am Ende gibt $\mathcal{A}$ eine Signatur σ^* aus. Er „gewinnt“ das Spiel, wenn

   $$\mathsf{Vfy}(pk, m^*, \sigma^*) = 1.$$

   $\mathcal{A}$ gewinnt also, wenn σ^* eine gültige Signatur für m^* ist.

Definition 4. Wir sagen, dass $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ sicher ist im Sinne von UUF-NMA, falls für alle PPT Angreifer $\mathcal{A}$ im UUF-NMA-Experiment gilt, dass

$$\Pr[\mathcal{A}^{\mathcal{C}}(pk, m^*) = \sigma^* : \mathsf{Vfy}(pk, m^*, \sigma^*) = 1] \leq \mathsf{negl}(k)$$

für eine vernachlässigbare Funktion negl im Sicherheitsparameter.

Wenn wir in Zukunft die Sicherheit eines Signaturverfahrens analysieren wollen, werden wir zur Angabe des Angreifers stets ein Sicherheitsexperiment angeben, zusammen mit einer Definition was der Angreifer tun muss um das „Spiel“ zu „gewinnen“.

Excercise 5. Beschreiben Sie das EUF-naCMA Sicherheitsexperiment, und geben Sie eine geeignete Definition an, die beschreibt, wann ein Signaturverfahren als EUF-naCMA-sicher bezeichnet wird.

1.2.3 Beziehungen zwischen Sicherheitsdefinitionen

Durch Kombination der in diesem Buch bislang vorgestellten Angreiferziele (EUF,UUF) mit den vorgestellten Angreifermodellen (NMA,naCMA,CMA) erhalten wir 6 Sicherheitsdefinitionen, siehe Tabelle 1.1. Eine wichtige Frage ist nun, wie sich diese Definitionen zueinander verhalten.

Intuition. Intuitiv erscheint klar, dass UUF-Sicherheit uns geringere Sicherheitsgarantien bietet als EUF-Sicherheit, denn:

- Um ein Signaturverfahren im Sinne von UUF zu brechen muss der Angreifer eine Signatur für eine *vorgegebene* Nachricht fälschen, er darf sie nicht selbst wählen.
- Um ein Signaturverfahren im Sinne von EUF zu brechen reicht es, wenn er eine Signatur für eine *selbst gewählte* Nachricht fälscht. Insbesondere kann er die Nachricht auswählen, für die es für ihn am Leichtesten ist, eine Signatur zu fälschen.

Es erscheint ebenso klar, dass uns NMA-Sicherheit geringere Sicherheitsgarantien bietet als naCMA-Sicherheit, und dass naCMA-Sicherheit geringere Sicherheitsgarantien bietet als CMA-Sicherheit, denn:

- Der Unterschied zwischen dem NMA-Experiment und naCMA-Experiment ist, dass der Angreifer im naCMA-Experiment ein „Signaturorakel“ zur Verfügung hat. Wenn also ein Signaturverfahren sicher ist gegen einen Angreifer, dem ein Signaturorakel zur Verfügung steht, dann ist es insbesondere auch dann sicher, wenn der Angreifer keinen Zugriff auf ein solches Orakel hat.
- Im naCMA-Experiment muss sich der Angreifer die Nachrichten, zu denen er eine Signatur anfragen möchte, überlegen, bevor er den Public-Key kennt. Im CMA-Experiment darf er jederzeit Signaturen anfragen. Wenn also ein Signaturverfahren sicher ist gegen einen Angreifer, der seine Anfragen an das Signaturorakel adaptiv wählen kann, dann ist es insbesondere auch dann sicher, wenn dies dem Angreifer nicht möglich ist.

Es scheint also eine Hierarchie zwischen den uns bekannten Sicherheitsdefinitionen zu geben:

$$\begin{array}{ccccc} \mathsf{UUF\text{-}NMA} & \leq & \mathsf{UUF\text{-}naCMA} & \leq & \mathsf{UUF\text{-}CMA} \\ \mid\!\wedge & & \mid\!\wedge & & \mid\!\wedge \\ \mathsf{EUF\text{-}NMA} & \leq & \mathsf{EUF\text{-}naCMA} & \leq & \mathsf{EUF\text{-}CMA} \end{array}$$

Demnach scheint unter den bislang vorgestellten Definitionen UUF-NMA die „Schwächste“ zu sein, und EUF-CMA die Stärkste. Diese Beziehungen kann man durch Reduktion beweisen.

Alle „$\leq$“-Beziehungen einzeln zu beweisen wäre nicht besonders interessant, da die Beweise stets nach dem gleichen „Kochrezept“ ablaufen. Daher betrachten wir nun als einfaches Beispiel nur die Aussage

$$\mathsf{UUF\text{-}NMA} \leq \mathsf{EUF\text{-}CMA}$$

Die übrigen Beziehungen im obigen Diagramm kann man auf die gleiche Art zeigen.

Theorem 6. *Sei $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ ein Signaturverfahren. Wenn Σ sicher ist im Sinne von EUF-CMA, dann ist es auch sicher im Sinne von UUF-NMA.*

Beweis. Wir nehmen an, dass Σ EUF-CMA-sicher ist, aber nicht UUF-NMA-sicher. Falls wir diese Annahme zu einem Widerspruch führen können, so muss sie falsch sein.

Nach Annahme ist Σ nicht UUF-NMA-sicher, also existiert ein Angreifer $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$ der in polynomieller Zeit läuft, sodass

$$\Pr[\mathcal{A}_{\mathsf{UUF\text{-}NMA}}(pk, m^*) = \sigma^* : \mathsf{Vfy}(pk, m^*, \sigma^*) = 1] \geq \frac{1}{\mathsf{poly}(k)}$$

für ein Polynom $\mathsf{poly}(\cdot)$ im Sicherheitsparameter k. Wir zeigen, dass wir aus diesem Angreifer einen neuen Angreifer $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ konstruieren können, der die EUF-CMA-Sicherheit von Σ bricht. Die Laufzeit von $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ ist ungefähr gleich zur Laufzeit von $\mathcal{A}_{\mathsf{UUF\text{-}CMA}}$, und die Erfolgswahrscheinlichkeit ist ebenfalls mindestens $1/\mathsf{poly}(k)$, und damit nicht-vernachlässigbar. Das bedeutet, dass Σ nicht EUF-CMA-sicher sein kann, Widerspruch!

Angreifer $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ benutzt $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$ wie folgt (siehe auch Abbildung 1.3).

1. $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ erhält als Eingabe den *public key* pk. Er wählt dann eine zufällige Nachricht $m^* \xleftarrow{\$} \{0,1\}^n$ und startet $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$ mit Eingabe (pk, m^*).

2. Wenn $A_{\mathsf{UUF\text{-}NMA}}$ eine Signatur σ^* mit $\mathsf{Vfy}(pk, m^*, \sigma^*) = 1$ ausgibt (dies passiert nach Annahme mit einer Wahrscheinlichkeit von mindestens $1/\mathsf{poly}(k)$), dann gibt $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ das Tupel (m^*, σ^*) aus. Ansonsten gibt $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ ein Fehlersymbol $\perp$ aus und terminiert.

Offensichtlich ist (m^*, σ^*) eine gültige Fälschung im Sinne von EUF-CMA, da $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ seinen Challenger niemals nach einer Signatur für m^* gefragt hat ($\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ fragt niemals nach irgendeiner Signatur für eine Nachricht, es ist also eigentlich sogar ein EUF-NMA Angreifer).

Die Laufzeit von $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ entspricht ungefähr der Laufzeit von $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$, bis auf einen kleinen Overhead. Die Erfolgswahrscheinlicheit von $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ ist identisch zur Erfolgswahrscheinlichkeit von $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$. Somit haben wir gezeigt: Wenn Σ nicht UUF-NMA-sicher ist, dann ist es auch nicht EUF-CMA-sicher. Dies gilt für jedes beliebige Signaturverfahren Σ. □

Excercise 7. Überlegen Sie, wie Sie die Aussage „UUF-CMA $\leq$ EUF-CMA“ beweisen würden.

Excercise 8. Zeigen Sie, dass UUF-Sicherheit *echt schwächer* als EUF-Sicherheit ist. Nehmen Sie dazu an, Sie hätten ein UUF-CMA-sicheres Verfahren gegeben, und zeigen Sie wie Sie dieses Verfahren so modifizieren können, dass es beweisbar UUF-CMA-sicher bleibt, aber nicht mehr EUF-NMA-sicher ist.

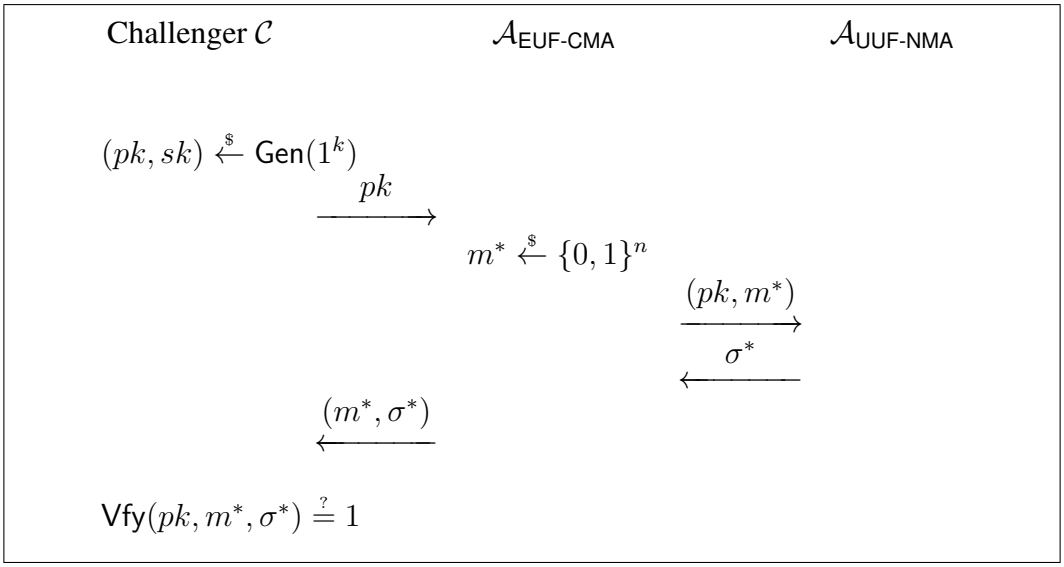

Abbildung 1.3: Konstruktion von $\mathcal{A}_{\mathsf{EUF\text{-}CMA}}$ aus $\mathcal{A}_{\mathsf{UUF\text{-}NMA}}$.

Excercise 9. Zeigen Sie, dass NMA-Sicherheit *echt schwächer* als CMA-Sicherheit ist. Nehmen Sie dazu an, Sie hätten ein EUF-NMA-sicheres Verfahren gegeben, und zeigen Sie wie Sie dieses Verfahren so modifizieren können, dass es beweisbar EUF-NMA-sicher bleibt, aber nicht mehr UUF-CMA-sicher ist.

1.2.4 Es gibt keine perfekt sicheren digitalen Signaturverfahren

Für manche kryptographischen Aufgaben, wie zum Beispiel symmetrische Verschlüsselung oder kryptographisches Secret-Sharing, gibt es *informationstheoretisch sichere* Lösungen, die zwar (oft) leider recht unpraktisch sind, aber dafür die starke Eigenschaft haben, dass sie selbst durch unbeschränkte Angreifer nicht gebrochen werden können. Eine natürliche Frage ist, ob es auch solch starke Signaturverfahren geben kann.

Es ist leider recht leicht zu sehen, dass es so starke Signaturverfahren nicht geben kann.

> *Es gibt kein Signaturverfahren, das sicher ist (im Sinne einer sinnvollen Sicherheitsdefinition) gegen einen Angreifer, dem unbeschränkte Ressourcen (insbes. Rechenzeit) zur Verfügung stehen.*

Obige Aussage ist natürlich recht ungenau, da nicht klar ist, was hier mit „einer sinnvollen Sicherheitsdefinition“ gemeint ist. Sie trifft jedoch den Kern und gilt für jede uns bislang bekannte Sicherheitsdefinition. Es scheint, dass man hier jede sinnvolle Definition von Sicherheit einsetzen kann.

Insbesondere können wir zeigen, dass nicht einmal unsere bislang schwächste Sicherheitsdefinition UUF-NMA erfüllbar ist, wenn wir unbeschränkte Angreifer erlauben.

Theorem 10. *Sei* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ *ein beliebiges Signaturverfahren. Es gibt einen UUF-NMA-Angreifer* $\mathcal{A}$ *gegen* Σ *mit*

- *Laufzeit* $O(2^L)$ *und*
- *Erfolgswahrscheinlichkeit* 1.

Dabei ist L die kleinste natürliche Zahl, sodass für jede Nachricht m eine gültige Signatur σ der Bitlänge L existiert.

Der Beweis dieses Theorems ist nicht besonders schwierig, wir lassen ihn daher als Übungsaufgabe.

Excercise 11. Beweisen Sie Theorem 10, indem Sie den Angreifer $\mathcal{A}$ beschreiben.

Wir müssen also stets die Laufzeit der von uns betrachteten Angreifer einschränken. Daher modellieren wir Angreifer als (möglicherweise probabilistische) Turingmaschine mit polynomieller Laufzeit *(probabilistic polynomial-time, PPT)*.

Als nächstes kann man sich nun fragen, ob es denn wenigstens Signaturverfahren gibt, die *perfekt sicher* sind, wenn wir den Angreifer auf eine polynomielle Laufzeit im Sicherheitsparameter k einschränken. *„Perfekt sicher“* bedeutet hier, dass die Erfolgswahrscheinlichkeit des Angreifers gleich Null ist. Es ist wieder leicht zu zeigen, dass auch dies nicht möglich ist.

Theorem 12. *Sei $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ ein beliebiges Signaturverfahren. Es gibt einen UUF-NMA-Angreifer $\mathcal{A}$ gegen Σ mit*

- *Laufzeit $O(L)$ und*
- *Erfolgswahrscheinlichkeit 2^{-L}.*

Dabei ist L die kleinste natürliche Zahl, sodass für jede Nachricht m eine gültige Signatur σ der Bitlänge L existiert.

Wir müssen also selbst einen auf polynomielle Laufzeit beschränkten Angreifer stets eine kleine Erfolgswahrscheinlichkeit einräumen. Auch dieser Beweis ist nicht besonders schwierig, daher stellen wir ihn wieder als leichte Übungsaufgabe.

Excercise 13. Beweisen Sie Theorem 12, indem Sie den Angreifer $\mathcal{A}$ beschreiben.

Das höchste Ziel, das wir bei der Konstruktion sicherer digitaler Signaturverfahren erreichen können, ist also, Angreifern mit *polynomiell beschränkter Laufzeit* höchstens eine *vernachlässigbar kleine Erfolgswahrscheinlichkeit* zu geben, die Sicherheit des Verfahrens zu brechen.

1.3 Erweiterung des Nachrichtenraumes

Wir möchten gerne Signaturverfahren konstruieren, die *Nachrichten beliebiger Länge* signieren können. Der Nachrichtenraum soll also die Menge $\{0,1\}^*$ aller endlichen Bit-Strings sein. Viele Konstruktionen, die wir im Folgenden kennen lernen werden, haben jedoch einen *endlichen Nachrichtenraum*, wie zum Beispiel

- die Menge $\{0,1\}^n$ aller Bitstrings der Länge n oder
- die Menge $\mathbb{Z}_p$, die wir durch die ganzen Zahlen $\{0,\dots,p-1\}$ darstellen.

Glücklicherweise ist es sehr einfach, ein Signaturverfahren mit endlichem Nachrichtenraum zu erweitern zu einem Signaturverfahren, mit dem Nachrichten beliebiger Länge signiert werden können. Dies geht mit Hilfe einer *kollisionsresistenten Hashfunktion*.

Kollisionsresistente Hashfunktionen.

Definition 14. Eine *Hashfunktion* H besteht aus zwei Polynomialzeit-Algorithmen $H = (\mathsf{Gen}_H, \mathsf{Eval}_H)$.

$\mathsf{Gen}_H(1^k)$ erhält als Eingabe den Sicherheitsparameter k und gibt einen Schlüssel t aus. Dieser Schlüssel t beschreibt eine Funktion

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

wobei $\mathcal{M}_t$ eine endliche Menge ist, die ebenfalls durch t festgelegt wird.

$\mathsf{Eval}_H(1^k, t, m)$ erhält als Eingabe den Sicherheitsparameter k, eine Funktionsbeschreibung t und eine Nachricht m und berechnet den Funktionswert $H_t(m)$.

Im Folgenden wird der Bezug zum Sicherheitsparameter k stets klar sein. Daher werden wir der Einfachheit halber nur H schreiben und meinen damit die Beschreibung t einer Funktion H_t, die durch $t \xleftarrow{\$} \mathsf{Gen}_H(1^k)$ erzeugt wurde. Ausserdem schreiben wir $H(m)$ als Kurzform für $H_t(m)$.

Definition 15. Wir sagen, dass $H = (\mathsf{Gen}_H, \mathsf{Eval}_H)$ *kollisionsresistent* ist, wenn für $t \xleftarrow{\$} \mathsf{Gen}_H(1^k)$ und für alle Polynomialzeit-Algorithmen $\mathcal{A}$ gilt, dass

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

für eine vernachlässigbare Funktion negl in k.

Remark 16. Hashfunktionen, die im obigen Sinne beweisbar kollisionsresistent sind, lassen sich von verschiedenen Komplexitätsannahmen, wie dem diskreten Logarithmusproblem, konstruieren. In der Praxis werden jedoch aus Effizienzgründen dedizierte Hashfunktionen wie zum Beispiel SHA-256 eingesetzt.

Die Kollisionsresistenz dieser Hashfunktionen lässt sich nicht bezüglich unserer asymptotischen Sicherheitsdefinitionen formulieren, da es für diese Hashfunktionen keinen Schlüssel t gibt, der in Abhängigkeit von einem Sicherheitsparameter generiert wird. Insbesondere haben diese Hashfunktionen eine konstante Ausgabelänge, sodass es natürlich einen Polynomialzeit-Algorithmus gibt, der Kollisionen findet.

Trotzdem scheint es nach aktuellem Kenntnisstand unmöglich, Kollisionen für diese Hashfunktionen zu finden. Daher kann jede Anwendung von kollisionsresistenten Hashfunktionen die wir kennenlernen werden in der Praxis auch mit einer solchen dedizierten Hashfunktionen instantiiert werden. Die Parametrisierung der Hashfunktion durch den Schlüssel t ist lediglich ein Artefakt unserer asymptotischen Sicherheitsdefinitionen.

Erweiterung des Nachrichtenraumes. Mit Hilfe einer Hashfunktion kann der Nachrichtenraum eines Signaturverfahrens erweitert werden. Sei $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vfy}')$ ein Signaturverfahren mit endlichem Nachrichtenraum $\mathcal{M}$ und sei $H : \{0,1\}^* \to \mathcal{M}$ eine Hashfunktion. Wir definieren ein Signaturverfahren $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ mit unendlichem Nachrichtenraum $\{0,1\}^*$ als:

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus Gen berechnet (pk, sk), indem er den Schlüsselerzeugungsalgorithmus $(pk, sk) \xleftarrow{\$} \mathsf{Gen}'(1^k)$ von Σ' laufen lässt.

$\mathsf{Sign}(sk, m)$**.** Um Nachricht $m \in \{0,1\}^*$ zu signieren, wird zunächst $H(m) \in \mathcal{M}$ berechnet und dann mittels $\sigma \xleftarrow{\$} \mathsf{Sign}'(sk, H(m))$ signiert.

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus berechnet $b = \mathsf{Vfy}'(pk, H(m), \sigma)$ und gibt b aus.

Die einzige Modifikation ist also, dass die Nachricht vor Signatur und Verifikation gehasht wird.

Theorem 17. *Wenn Σ' sicher ist im Sinne von EUF-CMA (bzw. EUF-naCMA) und H kollisionsresistent ist, dann ist Σ sicher im Sinne von EUF-CMA (bzw. EUF-naCMA).*

Excercise 18. Beweisen Sie Theorem 17.

Remark 19. Die obige Transformation führt neben den Annahmen, die notwendig sind für die Sicherheit des Signaturverfahrens Σ', eine zusätzliche Komplexitätsannahme ein, nämlich die Existenz kollisionsresistenter Hashfunktionen. Eine ähnliche Erweiterung des Nachrichtenraumes ist auch ohne zusätzliche Komplexitätsannahmen möglich, mittels *universal one-way*-Hashfunktionen. Wir gehen hierauf jedoch nicht weiter ein, eine Beschreibung findet sich zum Beispiel in [Kat10].

Kapitel 2

Einmalsignaturen

Einmalsignaturverfahren, oder kurz „Einmalsignaturen“, sind eine grundlegende und recht einfache Klasse von Signaturverfahren. Sie müssen nur relativ schwache Sicherheitseigenschaften erfüllen, denn es reicht aus wenn sie bei *einmaliger Verwendung* sicher sind. Einmalige Verwendung bedeutet, dass für jeden *public key* nur eine einzige Signatur ausgestellt wird.

Auf den ersten Blick erscheint das Konzept von Einmalsignaturen nicht besonders sinnvoll. Wofür soll es gut sein ein Signaturverfahren zu haben, mit dem man *nur eine einzige Signatur* ausstellen kann?

Interessanterweise sind Einmalsignaturen ein sehr starkes Werkzeug. Insbesondere können wir sie verwenden, um aus ihnen auf generische Art und Weise Mehrfach-Signaturverfahren zu konstruieren, mit denen wir dann eine praktisch unbeschränkte Anzahl von Signaturen ausstellen können. Die resultierenden Signaturen können starke Sicherheitseigenschaften haben, wie zum Beispiel EUF-CMA-Sicherheit.

Einmalsignaturen können von sehr schwachen Komplexitätsannahmen konstruiert werden, wie zum Beispiel der Schwierigkeit des diskreten Logarithmusproblems oder, noch allgemeiner, der Existenz von Einwegfunktionen. Zusammen mit der zuvor genannten generischen Konstruktion von Mehrfach-Signaturen aus Einmalsignaturen ist dies interessant, da es dabei hilft die minimal nötigen Komplexitätsannahmen für die Konstruktion digitaler Signaturen zu erforschen. Darüber hinaus sind Einmalsignaturen ein wichtiger Baustein für viele kryptographische Konstruktionen.

In diesem Kapitel passen wir zunächst die bekannten Sicherheitsdefinitionen EUF-CMA und EUF-naCMA auf Einmalsignaturen an. Danach betrachten wir eine allgemeine Konstruktion von Einmalsignaturen aus Einwegfunktionen und zwei spezielle Konstruktionen basierend auf konkreten Komplexitätsannahmen. Zum Schluss stellen wir einige Anwendungen von Einmalsignaturen vor. Dazu gehört eine generische Konstruktion, die es erlaubt aus einem beliebigen EUF-naCMA-sicheren Signaturverfahren ein EUF-CMA-sicheres Signaturverfahren zu bauen. Diese Transformation ist sehr hilfreich, und wird in der Literatur immer wieder benutzt. Danach beschreiben wir Baum-basierte Signaturen (*„Merkle Trees“*).

2.1 Sicherheitsdefinition

Falls wir einem Angreifer erlauben wollen (non-adaptive) chosen-message Signaturanfragen zu stellen, wie wir es im CMA und naCMA Angreifermodell tun, so müssen wir bei der Betrachtung von Einmalsignaturverfahren beachten, dass solche Verfahren nur dann Sicherheit

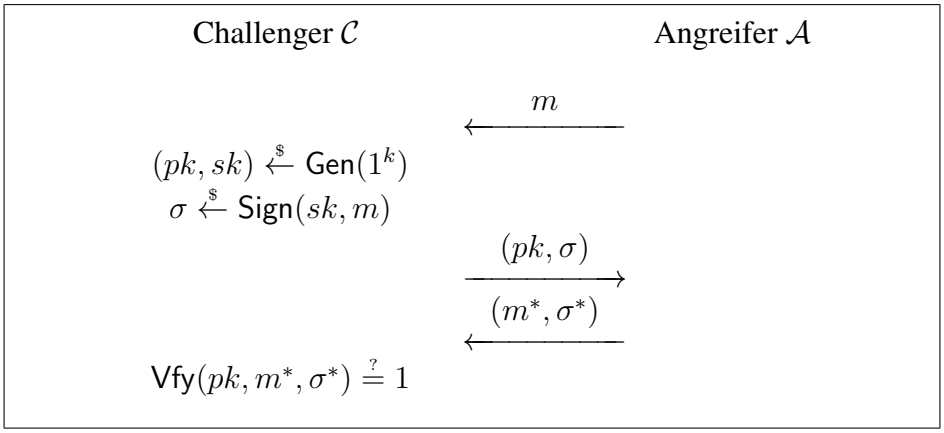

Abbildung 2.1: Das EUF-1-naCMA Sicherheitsexperiment.

gewährleisten müssen, wenn nur eine einzige Signatur ausgestellt wird.

Aus diesem Grunde werden wir die CMA und naCMA Angreifermodelle für die Betrachtung von Einmalsignaturen abschwächen. Anstatt die Anzahl der Nachrichten, für die der Angreifer eine Signatur anfragen kann, durch ein Polynom $q = q(k)$ zu beschränken, erlauben wir bei Einmalsignaturen *nur eine einzige* Signaturanfrage. Wir setzen also $q = 1$. Wenn wir dieses Angreifermodell mit dem Angreiferziel *existential forgery* (EUF) kombinieren, ergeben sich dadurch zwei neue Sicherheitsdefinitionen

- *existential unforgeability under one-time adaptive chosen-message attack* (EUF-1-CMA)
- *existential unforgeability under one-time non-adaptive chosen-message attack* (EUF-1-naCMA)

Wir werden uns in diesem Kapitel insbesondere für EUF-1-naCMA-Sicherheit interessieren, da dies ausreicht (wie wir später sehen werden), um daraus EUF-CMA-sichere Signaturen zu konstruieren. Daher beweisen wir in diesem Kapitel nur, dass Lamport's Einmalsignaturverfahren EUF-1-naCMA-sicher ist, unter der Annahme, dass die Funktion f eine Einwegfunktion ist.[1]

EUF-1-naCMA-Sicherheit. Das EUF-1-naCMA-Sicherheitsexperiment läuft wie folgt ab (siehe auch Abbildung 2.1):

1. Der Angreifer $\mathcal{A}$ wählt eine Nachricht m, für die er eine Signatur vom Challenger erhalten möchte.
2. Der Challenger $\mathcal{C}$ erzeugt ein Schlüsselpaar $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^k)$ sowie eine Signatur $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$, und sendet (pk, σ) an $\mathcal{A}$.
3. $\mathcal{A}$ gibt eine Nachricht m^* mit Signatur σ^* aus.

[1] Man kann auch zeigen, dass Lamport-Signaturen EUF-1-CMA-sicher sind. Der Beweis ist sehr ähnlich, daher eignet er sich gut als Übungsaufgabe.

Definition 20. Wir sagen, dass $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ ein EUF-1-naCMA-sicheres Einmalsignaturverfahren ist, falls für alle Polynomialzeit-Angreifer $\mathcal{A}$ im EUF-1-naCMA-Experiment gilt, dass

$$\Pr[\mathcal{A}^{\mathcal{C}} = (m^*, \sigma^*) : \mathsf{Vfy}(pk, m^*, \sigma^*) = 1 \wedge m^* \neq m] \leq \mathsf{negl}(k)$$

für eine vernachlässigbare Funktion negl im Sicherheitsparameter.

Excercise 21. Beschreiben Sie das EUF-1-CMA-Sicherheitsexperiment und geben Sie eine Definition an wann der Angreifer im Experiment gewinnt.

2.2 Einmalsignaturen von Einwegfunktionen

In diesem Kapitel beschreiben wir die Konstruktion von Einmalsignaturen aus Einwegfunktionen von Lamport [Lam79]. Diese Konstruktion ist besonders interessant, weil die Existenz von Einwegfunktionen eine der schwächsten Annahmen in der Kryptographie ist.

2.2.1 Einwegfunktionen

Die Grundidee von Einwegfunktionen ist:

- Für eine gegebene Funktion $f : \{0,1\}^* \to \{0,1\}^*$ und eine Eingabe $x \in \{0,1\}^*$ soll es „leicht“ sein den Funktionswert $y = f(x)$ zu berechnen. „Leicht“ bedeutet, wie üblich, dass die Laufzeit durch ein Polynom in der Länge von x beschränkt ist.
- Gegeben einen Wert $y = f(x)$ soll es jedoch „praktisch unmöglich“ sein $f^{-1}(y) = x$ zu berechnen, also ein Urbild von y bezüglich f zu finden. „Praktisch unmöglich“ bedeutet, es soll kein Polynomialzeit-Algorithmus existieren, der $f^{-1}(y)$ berechnet.

Eine wichtige Grundannahme in der Kryptographie ist, dass Einwegfunktionen existieren. Ob diese Annahme stimmt ist unbekannt. Trotzdem ist die Existenz von Einwegfunktionen eine der schwächsten Annahmen in der Kryptographie, und sie ist notwendig um viele interessante Dinge zu konstruieren, wie zum Beispiel Blockchiffren oder digitale Signaturen. Für viele andere kryptographische Primitive, wie zum Beispiel *public key*-Verschlüsselung oder identitätsbasierte Verschlüsselung, benötigt man (vermutlich) wesentlich stärkere Annahmen.

Definition von Einwegfunktionen. Eine abstrakte, allgemeine Definition von Einwegfunktionen anzugeben, die gleichzeitig alle wichtigen Beispiele von in der Kryptographie verwendeten Einwegfunktionen (wie kryptographische Hashfunktionen, der RSA-Funktion oder der diskreten Exponentialfunktion) konsistent abdeckt, ist garnicht so einfach — jedoch glücklicherweise auch nicht unbedingt notwendig.

Der Einfachheit halber werden wir daher in diesem Kapitel Funktionen betrachten, die sich als eine einzelne Funktion der Form

$$f : \{0,1\}^* \to \{0,1\}^*$$

beschreiben lassen. Wir betrachten also Funktionen deren Eingabe- und Ausgabemenge *alle* endlichen Bit-Strings umfasst. Weiterhin nehmen wir an, dass die Funktion *effizient berechenbar* ist, also insbesondere eine *Beschreibung konstanter Länge* hat, zum Beispiel in Form eines

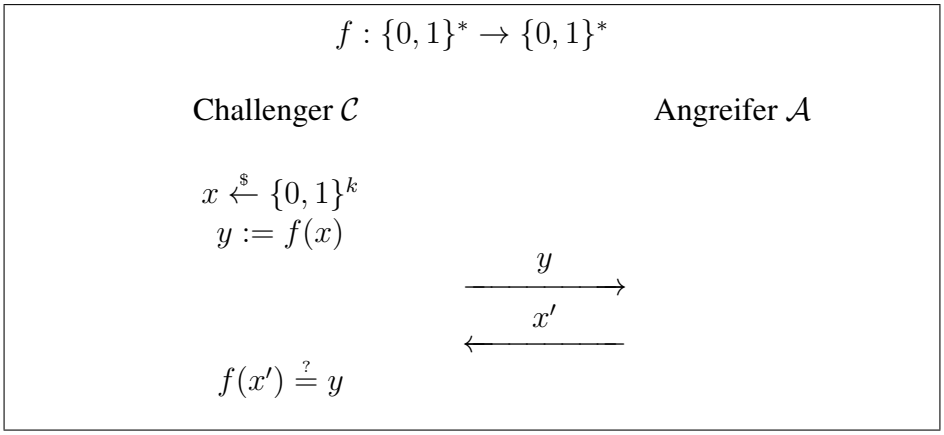

Abbildung 2.2: Das Sicherheitsexperiment für Einwegfunktionen.

Algorithmus, und dass für eine Eingabe x mit $|x| = k$ die Länge der Ausgabe $y = f(x)$ nach oben beschränkt ist durch ein Polynom $\mathsf{poly}(k)$ in k. Dies ist hilfreich, da es uns zahlreiche irrelevante technische Details erspart und somit die Betrachtung wesentlich vereinfacht. Gleichzeitig tritt durch die vereinfachte Formulierung auch die Grundidee der Konstruktionen, um die es hier eigentlich geht, mehr in den Vordergrund.

Sicherheitsexperiment für Einwegfunktionen. Das Sicherheitsexperiment mit Einwegfunktion f läuft wie folgt ab (siehe Abbildung 2.2).

1. Der Challenger $\mathcal{C}$ wählt $x \xleftarrow{\$} \{0,1\}^k$ gleichverteilt zufällig und berechnet $y = f(x)$.
2. Der Angreifer erhält die Beschreibung der Funktion f und y als Eingabe. Er „gewinnt“ das Spiel, wenn er $x' \in \{0,1\}^*$ ausgibt, sodass $f(x') = y$.

Definition 22. Wir sagen, dass f eine Einwegfunktion ist, falls f effizient berechenbar ist und für alle Polynomialzeit-Algorithmen $\mathcal{A}$ im Sicherheitsexperiment für Einwegfunktionen gilt, dass

$$\Pr[\mathcal{A}(1^k, y) = x' : f(x') = y] \leq \mathsf{negl}(k)$$

für eine vernachlässigbare Funktion negl.

Kandidaten für Einwegfunktionen. Wir wissen leider nicht, ob Einwegfunktionen existieren. Aber wir kennen einige Kandidaten für Funktionen, bei denen man vermutet, dass sie Beispiele für Einwegfunktionen sind:

Kryptographische Hashfunktionen. Eine (recht milde) Sicherheitsanforderung an eine kryptographische Hashfunktion ist, dass sie schwer zu invertieren ist. Für kryptographische Hashfunktionen gibt es viele Kandidaten, wie zum Beispiel MD5, SHA-1, RIPE-MD oder die aktuelle SHA-3 Hashfunktion.

Leider lassen sich Funktionen wie MD5 und die anderen genannten Beispiele nicht in der von uns gewählten asymptotischen Formulierung als Einwegfunktionen auffassen, da die Länge der Ausgabe dieser Funktionen konstant ist und somit natürlich ein

Polynomialzeit-Algorithmus existiert, der mit guter Wahrscheinlichkeit Urbilder findet. Trotzem ist es für diese Funktionen nach aktuellem Kenntnisstand praktisch unmöglich Urbilder zu berechnen, und diese Funktionen sind geeignet um die hier vorgestellten auf Einwegfunktionen basierenden Verfahren in der Praxis zu instantiieren.

Die diskrete Exponentialfunktion. Sei $\mathbb{G}$ eine endliche Gruppe mit Generator g und Ordnung p. Für manche solche Gruppen ist die Funktion $\mathbb{Z}_p \to \mathbb{G}$, $x \mapsto g^x$ ein Kandidat für eine Einwegfunktion, nämlich dann, wenn das diskrete Logarithmusproblem in der Gruppe nicht effizient lösbar ist. Dazu gehören (vermutlich) geeignete Untergruppen der multiplikativen Gruppe $\mathbb{Z}_p^*$ oder bestimmte Elliptische Kurven Gruppen.

Man kann die diskrete Exponentialfunktion zwar nicht *auf natürliche Weise* als Funktion $f : \{0,1\}^* \to \{0,1\}^*$ auffassen, jedoch kann man sich die diskrete Exponentialfunktion auch als eine Familie von Funktionen $(f_k)_{k\in\mathbb{N}}$ vorstellen. Teil der Beschreibung jeder Funktion f_k ist die Beschreibung einer Gruppe $\mathbb{G}$ mit Generator g und Ordnung p, wobei $p = p(k)$ von k abhängt.

Die RSA-Funktion. Auch bei der RSA-Funktion $\mathbb{Z}_N \to \mathbb{Z}_N$, $x \mapsto x^e \bmod N$ geht man davon aus, dass sie (für geeignet gewählte (N, e)) eine Einwegfunktion darstellt.

Auch die RSA-Funktion kann man sich als Familie von Funktionen $(f_k)_{k\in\mathbb{N}}$ vorstellen. Teil der Beschreibung jeder Funktion f_k sind zwei ganze Zahlen (N, e), welche die Funktion $f_k : x \mapsto x^e \bmod N$ definieren.

Die RSA-Funktion stellt sogar einen Kandidaten für eine *Trapdoor-Einwegpermutation* dar. Das bedeutet, dass die Funktion effizient und eindeutig invertierbar ist, wenn man eine passende „Falltür“ (*„Trapdoor“*) kennt. Im Falle von RSA dient die Faktorisierung des Modulus N als Trapdoor.

2.2.2 Lamport's Einmalsignaturverfahren

Im Jahr 1979 hat Lamport [Lam79] ein sehr einfaches und elegantes Einmalsignaturverfahren beschrieben, welches auf Einwegfunktionen basiert. In diesem Kapitel beschreiben wir Lamport's Signaturverfahren und analysieren seine Sicherheit.

Um ein Signaturverfahren zu beschreiben, müssen wir drei Algorithmen $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ angeben. Im Falle von Lamport's Verfahren benutzen diese Algorithmen eine Funktion $f : \{0,1\}^* \to \{0,1\}^*$, von der wir in der Sicherheitsanalyse annehmen werden, dass sie eine Einwegfunktion ist. Wir bezeichnen mit n die Länge der Nachrichten, die mit dem Verfahren signiert werden sollen. Dabei ist $n = n(k)$ ein Polynom im Sicherheitsparameter k.

$\mathsf{Gen}(1^k)$. Der Schlüsselerzeugungsalgorithmus Gen wählt $2n$ gleichverteilt zufällige Werte

$$x_{1,0}, x_{1,1}, \ldots, x_{n,0}, x_{n,1} \xleftarrow{\$} \{0,1\}^k$$

und berechnet

$$y_{i,j} := f(x_{i,j}) \qquad \forall i \in \{1,\ldots,n\}, \forall j \in \{0,1\}.$$

Die Schlüssel pk und sk sind definiert als

$$pk := (f, y_{1,0}, y_{1,1}, \ldots, y_{n,0}, y_{n,1}) \quad \text{und} \quad sk := (x_{1,0}, x_{1,1}, \ldots, x_{n,0}, x_{n,1}).$$

$\mathsf{Sign}(sk, m)$**.** Gegeben ist sk und eine Nachricht $m \in \{0,1\}^n$. Die Nachricht wird signiert, indem bestimmte Urbilder der im pk enthaltenen Elemente $y_{i,j}$ veröffentlicht werden. Die einzelnen Bits der Nachricht bestimmen, welches Urbild veröffentlicht wird.

Wir bezeichnen das i-te Bit von $m \in \{0,1\}^n$ mit $m_i \in \{0,1\}$, also $m = (m_1, \ldots, m_n) \in \{0,1\}^n$. Die Signatur σ für die Nachricht m besteht aus

$$\sigma = (x_{1,m_1}, \ldots, x_{n,m_n}).$$

$\mathsf{Vfy}(pk, m, \sigma)$**.** Zur Verifikation wird geprüft, ob die in σ enthaltenen Werte $x_{i,j}$ tatsächlich Urbilder der im pk enthaltenen Elemente $y_{i,j}$ sind.

Gegeben den öffentlichen Schlüssel pk, eine Nachricht $m = (m_1, \ldots, m_n) \in \{0,1\}^n$ und eine Signatur $\sigma = (x'_{1,m_1}, \ldots, x'_{n,m_n})$, wird geprüft ob

$$f(x'_{i,m_i}) = y_{i,m_i} \qquad \forall i \in \{1, \ldots, n\}.$$

Falls dies für alle $i \in \{1, \ldots, n\}$ erfüllt ist, so wird die Signatur akzeptiert, also 1 ausgegeben. Ansonsten wird 0 ausgegeben.

Die *Correctness* dieses Verfahrens ist offensichtlich. Im Folgenden werden wir beweisen, dass auch die *Soundness* Eigenschaft für eine geeignete Definition von „Sicherheit“ erfüllt ist, falls f eine Einwegfunktion ist.

Theorem 23. *Sei Σ das Lamport Einmalsignaturverfahren, instantiiert mit Einwegfunktion f und Nachrichtenlänge n. Für jeden PPT-Angreifer $\mathcal{A}$, der die EUF-1-naCMA-Sicherheit von Σ in Zeit $t_\mathcal{A}$ mit Erfolgswahrscheinlichkeit $\epsilon_\mathcal{A}$ bricht, existiert ein PPT-Angreifer $\mathcal{B}$, der die Einwegfunktion f in Zeit $t_\mathcal{B}$ mit Erfolgswahrscheinlichkeit $\epsilon_\mathcal{B}$ bricht, sodass*

$$t_\mathcal{A} \approx t_\mathcal{B} \qquad \textit{und} \qquad \epsilon_\mathcal{B} \geq \frac{\epsilon_\mathcal{A}}{n}.$$

Falls $\mathcal{A}$ ein Polynomialzeit-Angreifer mit nicht-vernachlässigbarer Erfolgswahrscheinlichkeit ist, so ist $t_\mathcal{A}$ nach oben beschränkt durch $t_\mathcal{A} \leq \mathsf{poly}(k)$, und $\epsilon_\mathcal{A}$ ist nach unten beschränkt durch $\epsilon_\mathcal{A} \geq 1/\mathsf{poly}(k)$ für ein Polynom poly im Sicherheitsparameter. Dann existiert auch ein Angreifer $\mathcal{B}$, der die Einwegfunktion f in polynomieller Zeit und mit nicht-vernachlässigbarer Wahrscheinlichkeit bricht. Wenn man annimmt, dass es so einen Angreifer $\mathcal{B}$ nicht geben kann, dann kann es also auch $\mathcal{A}$ nicht geben.

Die Beweisidee ist recht einfach. $\mathcal{B}$ benutzt $\mathcal{A}$ als „Subroutine“, indem er den Challenger für $\mathcal{A}$ simuliert. Die Nachricht m^*, für die der Angreifer die Signatur fälscht, muss sich an mindestens einer Stelle von der Nachricht m, für die der Angreifer eine Signatur anfragt, unterscheiden. Angreifer $\mathcal{B}$ rät diese Stelle, und bettet dort das Bild $y = f(x)$ der Einwegfunktion ein, für die er ein Urbild x' mit $f(x') = y$ berechnen soll. Den Rest des pk generiert er so, dass er eine gültige Signatur für m erstellen kann.

Beweis. Wir konstruieren $\mathcal{B}$. $\mathcal{B}$ spielt das Sicherheitsexperiment für Einwegfunktionen, und erhält als Eingabe eine Beschreibung der Funktion f und einen Wert $y = f(x)$, wobei $x \xleftarrow{\$} \{0,1\}^k$ gleichverteilt zufällig gewählt ist.

$\mathcal{B}$ benutzt Angreifer $\mathcal{A}$, indem er den EUF-1-naCMA-Challenger für $\mathcal{A}$ simuliert. Daher erhält $\mathcal{B}$ als Erstes von $\mathcal{A}$ eine Nachricht $m = (m_1, \ldots, m_n) \in \{0,1\}^n$, für die $\mathcal{A}$ eine Signatur sehen möchte.

Nun konstruiert $\mathcal{B}$ einen öffentlichen Schlüssel pk wie folgt:

1. $\mathcal{B}$ wählt einen zufälligen Index $\nu \xleftarrow{\$} \{1, \ldots, n\}$ und definiert $y_{\nu,1-m_\nu} := y$.

2. Um einen vollständigen pk erzeugen zu können, wählt $\mathcal{B}$ nun noch $2n-1$ zufällige Werte $x_{i,j}$ und berechnet $y_{i,j} := f(x_{i,j})$ für alle übrigen $(i,j) \in \{1,\ldots,n\} \times \{0,1\}$ mit $(i,j) \neq (\nu, 1-m_\nu)$. Der von $\mathcal{B}$ erzeugte pk ist

$$pk = (y_{1,0}, y_{1,1}, \ldots, y_{n,0}, y_{n,1}).$$

Man beachte, dass nun $\mathcal{B}$ fast alle Urbilder der Werte $y_{i,j}$ kennt, mit *einer einzigen* Ausnahme: Das Urbild von $y_{\nu,1-m_\nu}$ kennt er nicht, denn hier hat er die Challenge y eingebettet. Damit kann $\mathcal{B}$ nun *jede beliebige* Nachricht $m' = (m'_1, \ldots, m'_n)$ signieren, *solange* $m'_\nu = m_\nu$ ist, also das ν-te Bit der Nachricht m' gleich dem ν-ten Bit der Nachricht m ist.

$\mathcal{B}$ kann also insbesondere eine gültige Signatur σ für m simulieren, da er alle *secret key* Elemente kennt die nötig sind um

$$\sigma = (x_{1,m_1}, \ldots, x_{n,m_n})$$

zu bestimmen.

$\mathcal{A}$ erhält nun als Eingabe (pk, σ). Der pk ist perfekt ununterscheidbar von einem *public key* im „echten“ EUF-1-naCMA Experiment und σ ist eine gültige Signatur für m bezüglich pk. Daher wird $\mathcal{A}$ (nach Annahme) mit Wahrscheinlichkeit $\epsilon_\mathcal{A}$ eine Nachricht m^* zusammen mit einer gefälschten Signatur σ^* ausgeben, sodass $m^* \neq m$ und $\mathsf{Vfy}(pk, m^*, \sigma^*) = 1$.

$\mathcal{B}$ hofft nun darauf, dass $\mathcal{A}$ eine gültige Signatur σ^* fälscht für eine Nachricht $m^* = (m^*_1, \ldots, m^*_n) \in \{0,1\}^n$, sodass sich m^* an der ν-ten Stelle von m unterscheidet. Da sich m^* an mindestens einer Position von m unterscheidet, und $\mathcal{A}$ absolut keine Information über ν erhält, tritt dies mit Wahrscheinlichkeit mindestens $1/n$ ein. In diesem Falle lernt $\mathcal{B}$ ein Urbild von y, das er benutzen kann um das Sicherheitsexperiment für Einwegfunktionen zu gewinnen.

$\mathcal{B}$ ist also erfolgreich, wenn (i) $\mathcal{A}$ erfolgreich ist und (ii) sich m^* von m an der ν-ten Stelle unterscheidet, also $m^*_\nu \neq m_\nu$. Dabei tritt (i) nach Annahme mit Wahrscheinlichkeit $\epsilon_\mathcal{A}$ ein und (ii) mit Wahrscheinlichkeit mindestens $1/n$. Da der Angreifer keine Information über ν erhält sind beide Ereignisse von einander unabhängig. Daher ist

$$\epsilon_\mathcal{B} \geq \epsilon_\mathcal{A} \cdot 1/n.$$

Ausserdem haben wir $t_\mathcal{A} \approx t_\mathcal{B}$, denn die Laufzeit von $\mathcal{B}$ entspricht ungefähr der Laufzeit von $\mathcal{A}$, plus einem kleinen Overhead. □

Mit nahezu der gleichen Beweistechnik ist es leicht möglich direkt zu zeigen, dass Lamport-Signaturen auch sicher im Sinne von one-time *adaptive* chosen-message attacks (EUF-1-CMA) sind. Dies ist eine simple Erweiterung des hier vorgestellten Beweises, die sich gut als Übungsaufgabe eignet.

Remark 24. Wenn man sich den Beweis von Theorem 23 näher ansieht und etwas darüber nachdenkt, dann kann man sehen, dass er im Wesentlichen aus zwei wichtigen Teilen besteht:

1. *Simulation:* Zuerst muss der Algorithmus $\mathcal{B}$, der gerne den Signatur-Angreifer $\mathcal{A}$ benutzen möchte um ein „schwieriges“ Problem zu lösen (nämlich im Falle von Theorem 23 die Invertierung einer Einwegfunktion), dafür sorgen, dass $\mathcal{A}$ auch mit der gewünschten Erfolgswahrscheinlichkeit $\epsilon_\mathcal{A}$ eine gefälschte Signatur ausgibt. Daher muss die „Sicht“

von $\mathcal{A}$ ununterscheidbar vom echten Sicherheitsexperiment sein — ansonsten könnte es sein, dass $\mathcal{A}$ nicht mehr funktioniert, wenn sich seine Sicht auf das Experiment vom echten Sicherheitsexperiment unterscheidet.

Aus diesem Grunde war es wichtig, dass $\mathcal{B}$ den pk so berechnet hat, dass er *ganz genauso verteilt* ist wie in einem echten Sicherheitsexperiment. Auch die Signatur σ, die $\mathcal{A}$ von $\mathcal{B}$ erhalten hat, war für $\mathcal{A}$ *perfekt ununterscheidbar* von einer Signatur im echten Experiment.

Wenn man es schafft, den Angreifer $\mathcal{A}$ zu überzeugen, dass er tatsächlich im echten Sicherheitsexperiment teilnimmt, dann ist das schonmal die „halbe Miete“ im Sicherheitsbeweis, denn dann liefert er uns eine Fälschung (m^*, σ^*) mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{A}}$.

2. *Extraktion:* Die „zweite Hälfte der Miete“ ist nun, dass wir aus der gefälschten Signatur (m^*, σ^*) noch die Lösung für das schwierige Problem extrahieren müssen.

   Zum Beispiel im Falle von Lamport-Signaturen konnten wir hoffen, dass $\mathcal{A}$ uns mit guter Wahrscheinlichkeit das gesuchte Urbild von y als Teil der Signatur liefert.

Diese zwei Aufgaben sind der Kern aller Sicherheitsbeweise für digitale Signaturen. Auch neue Signaturverfahren werden in der Regel so konstruiert, dass man später im Sicherheitsbeweis einerseits den pk und ggfs. gültige Signaturen simulieren kann, und andererseits später die Lösung eines schwierigen Problems aus der Fälschung extrahieren kann.

Excercise 25. Beweisen Sie, dass Lamport's Einmalsignaturen EUF-1-CMA-sicher sind, unter der Annahme, dass f eine Einwegfunktion ist.

Remark 26. Winternitz-Signaturen [Mer88] sind eine Erweiterung von Lamport-Signaturen. Hier wird die Nachricht nicht als binärer String, sondern allgemein in w-adischer Darstellung mit $w \geq 2$ dargestellt. Dies erlaubt einen Trade-Off, durch den kürzere Signaturen erreicht werden können, auf Kosten von etwas mehr Berechnungen und einer zusätzlichen Prüfsumme. Siehe [Mer88] für Details.

2.3 Effiziente Einmalsignaturen von konkreten Komplexitätsannahmen

In diesem Kapitel werden wir zwei Einmalsignaturverfahren vorstellen, die auf konkreten Komplexitätsannahmen basieren. Diese Konstruktionen sind wesentlich effizienter als Lamport's generisches Verfahren. Wir werden sie später benutzen, um effiziente konkrete Signaturverfahren zu konstruieren.

Wir setzen grundlegende Vorkenntnisse über das diskrete Logarithmusproblem und das RSA-Problem voraus, eine umfassende Einführung findet sich in [KL07].

2.3.1 Einmalsignaturen basierend auf dem Diskreten Logarithmusproblem

Definition 27. Sei $\mathbb{G}$ eine endliche Gruppe mit Generator g und Ordnung $p \in \mathbb{N}$ (im Folgenden wird $\mathbb{G}$ stets abelsch, also kommutativ, sein und p meist eine Primzahl). Das *diskrete Logarith-*

musproblem in $\mathbb{G}$ ist: Gegeben ein zufälliges Gruppenelement $y \in \mathbb{G}$, finde eine ganze Zahl $x \in \mathbb{Z}_p$ sodass

$$g^x = y.$$

Wir nehmen an, dass Gruppen existieren, in welchen das diskrete Logarithmusproblem „schwer“ ist. Kandidaten für solche Gruppen sind zum Beispiel Untergruppen der multiplikativen Gruppe $\mathbb{Z}_q^*$ für eine natürliche Zahl $q \in \mathbb{N}$ (für q wird oft eine Primzahl gewählt, jedoch nicht notwendigerweise) oder bestimmte Elliptische Kurven Gruppen.

Basierend auf der Schwierigkeit des diskreten Logarithmusproblems lässt sich ein sehr einfaches Einmalsignaturverfahren konstruieren, welches eng mit dem Commitmentverfahren von Pedersen [Ped92] verwandt ist.

Sei im Folgenden also $\mathbb{G}$ eine endliche abelsche Gruppe mit Generator g und primer Ordnung p. Sei Σ das folgende Signaturverfahren mit Nachrichtenraum $\mathbb{Z}_p$.

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus wählt $x, \omega \xleftarrow{\$} \mathbb{Z}_p$ und berechnet $h := g^x$ und $c := g^\omega$. Der öffentliche Schlüssel ist $pk := (g, h, c)$, der geheime Schlüssel ist $sk := (x, \omega)$.

$\mathsf{Sign}(sk, m)$**.** Um die Nachricht $m \in \mathbb{Z}_p$ zu signieren, wird $\sigma \in \mathbb{Z}_p$ berechnet als

$$\sigma := \frac{\omega - m}{x}.$$

$\mathsf{Vfy}(pk, m, \sigma)$**.** Zur Verifikation wird geprüft, ob die Gleichung

$$c \stackrel{?}{=} g^m h^\sigma$$

erfüllt ist.

Die Correctness des Verfahrens kann durch einfaches Einsetzen gezeigt werden:

$$g^m h^\sigma = g^{m+x\sigma} = g^{m+x((\omega-m)/x)} = g^\omega = c.$$

Theorem 28. *Für jeden PPT-Angreifer $\mathcal{A}$, der die EUF-1-naCMA-Sicherheit von Σ in Zeit $t_\mathcal{A}$ mit Erfolgswahrscheinlichkeit $\epsilon_\mathcal{A}$ bricht, existiert ein PPT-Angreifer $\mathcal{B}$, der das diskrete Logarithmusproblem in $\mathbb{G}$ löst in Zeit $t_\mathcal{B} \approx t_\mathcal{A}$ mit Erfolgswahrscheinlichkeit $\epsilon_\mathcal{B} \geq \epsilon_\mathcal{A}$.*

Beweis. Angreifer $\mathcal{B}$ erhält als Eingabe den Generator g und ein zufälliges Gruppenelement h, dessen diskreten Logarithmus zur Basis g er berechnen soll. Er simuliert den Challenger für $\mathcal{A}$ wie folgt.

Im EUF-1-naCMA-Experiment gibt $\mathcal{A}$ zuerst eine Nachricht $m \in \mathbb{Z}_p$ aus, für die er eine Signatur sehen möchte. $\mathcal{B}$ wählt einen zufälligen Wert $\sigma \xleftarrow{\$} \mathbb{Z}_p$ und berechnet

$$c := g^m h^\sigma.$$

Dann sendet er den öffentlichen Schlüssel $pk = (g, h, c)$ und die Signatur σ an $\mathcal{A}$.

Der pk besteht aus dem Generator g und zwei gleichverteilt zufälligen Gruppenelementen (h, c) und ist somit ein korrekt verteilter öffentlicher Schlüssel. Der Wert σ ist eine gültige

Signatur für m bezüglich pk. Daher gibt $\mathcal{A}$ mit Wahrscheinlichkeit $\epsilon_{\mathcal{A}}$ eine Fälschung (m^*, σ^*) aus mit

$$c = g^{m^*} h^{\sigma^*}.$$

Diese Fälschung kann $\mathcal{B}$ benutzen, um den diskreten Logarithmus von h zur Basis g zu berechnen. Es gilt nämlich

$$g^{m^*} h^{\sigma^*} = g^m h^\sigma \iff g^{m^*+x\sigma^*} = g^{m+x\sigma} \iff m^* + x\sigma^* \equiv m + x\sigma \bmod p.$$

Weil $m \not\equiv m^* \bmod p$ ist muss auch $\sigma \not\equiv \sigma^* \bmod p$ sein. Daher kann x berechnet werden als

$$x := \frac{m - m^*}{\sigma^* - \sigma} \bmod p.$$

Die Laufzeit von $\mathcal{B}$ entspricht ungefähr der Laufzeit von $\mathcal{A}$, plus einem minimalen Overhead, und die Erfolgswahrscheinlichkeit von $\mathcal{B}$ ist mindestens so groß wie die Erfolgswahrscheinlichkeit von $\mathcal{A}$. □

2.3.2 Einmalsignaturen basierend auf der RSA-Annahme

Erinnerung und Einführung der Notation. Wir bezeichnen mit $\mathbb{Z}_N$ die Menge

$$\mathbb{Z}_N := \{0, \ldots, N-1\} \subset \mathbb{Z}.$$

$\mathbb{Z}_N$ ist eine Gruppe bezüglich der Addition modulo N, und ein Ring bezüglich der Addition und Multiplikation modulo N. Wir schreiben $\mathbb{Z}_N^*$, um die Menge der Einheiten modulo N zu bezeichnen, also $\mathbb{Z}_N^* := \{a \in \mathbb{Z}_N : \mathrm{ggT}(a, N) = 1\}$. $\mathbb{Z}_N^*$ ist eine Gruppe bezüglich der Multiplikation modulo N.

Die Gruppe $\mathbb{Z}_N^*$ hat Ordnung $\phi(N)$, wobei $\phi(N)$ die Eulersche Phi-Funktion ist. Falls $N = PQ$ das Produkt von zwei verschiedenen Primzahlen ist, dann ist $\phi(N) = (P-1) \cdot (Q-1)$. Für eine natürliche Zahl e mit $\mathrm{ggT}(e, \phi(N)) = 1$ und für $d \equiv e^{-1} \bmod \phi(N)$ gilt also

$$(x^e)^d \equiv x^{ed \bmod \phi(N)} \equiv x^1 \equiv x \bmod N$$

für alle $x \in \mathbb{Z}_N$.

Definition 29. Sei $N := PQ$ das Produkt von zwei Primzahlen. Sei $e \in \mathbb{N}$ sodass $e > 1$ und $\mathrm{ggT}(e, \phi(N)) = 1$ und sei $y \xleftarrow{\$} \mathbb{Z}_N$ eine zufällige Zahl modulo N. Das durch (N, e, y) gegebene *RSA-Problem* ist: Berechne $x \in \mathbb{Z}_N$, sodass

$$x^e \equiv y \bmod N.$$

Es ist bekannt, dass das RSA-Problem höchstens so schwer ist wie das Problem der Faktorisierung von N. Ob die Umkehrung auch gilt, also ob das RSA-Problem äquivalent zum Faktorisierungsproblem ist, ist unbekannt. Zumindest kennen wir keinen Algorithmus, der das RSA-Problem besser löst als durch Berechnung des geheimen RSA-Schlüssels d mit $d \equiv 1/e \bmod \phi(N)$. Den geheimen Schlüssel d zu berechnen ist genauso schwer wie N zu faktorisieren [RSA78, May04] — es könnte jedoch einen leichteren Weg geben e-te Wurzeln modulo N zu berechnen ohne dafür d kennen zu müssen.

RSA-basierte Einmalsignaturen. Das nachfolgend dargestellte Signaturverfahren Σ ist angelehnt an eine Konstruktion aus [HW09a]. Das Verfahren hat Nachrichtenraum $[0, 2^n - 1]$.

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus erzeugt einen RSA-Modulus $N = PQ$ sowie eine Primzahl $e > 2^n$ mit $\mathrm{ggT}(e, \phi(N)) = 1$ und berechnet $d := e^{-1} \bmod \phi(N)$. Außerdem werden zwei Zahlen $J, c \xleftarrow{\$} \mathbb{Z}_N$ zufällig gewählt. Der öffentliche Schlüssel ist $pk := (N, e, J, c)$, der geheime Schlüssel ist $sk := d$.

$\mathsf{Sign}(sk, m)$**.** Um eine Nachricht $m \in [0, 2^n - 1]$ zu signieren wird $\sigma \in \mathbb{Z}_N$ berechnet als

$$\sigma \equiv (c/J^m)^d \bmod N.$$

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus gibt 1 aus, wenn

$$c \equiv J^m \sigma^e \bmod N$$

gilt, und ansonsten 0.

Die *Correctness* des Verfahrens ergibt sich wieder durch Einsetzen:

$$c \equiv J^m \sigma^e \equiv J^m \cdot ((c/J^m)^d)^e \equiv J^m \cdot c/J^m \equiv c \bmod N.$$

Theorem 30. *Sei (N, e, y) eine Instanz des RSA-Problems sodass $e > 2^n$ eine Primzahl ist. Für jeden PPT-Angreifer $\mathcal{A}$, der die EUF-1-naCMA-Sicherheit von Σ in Zeit $t_\mathcal{A}$ mit Erfolgswahrscheinlichkeit $\epsilon_\mathcal{A}$ bricht, existiert ein PPT-Angreifer $\mathcal{B}$, der $x \in \mathbb{Z}_N$ berechnet mit $x^e \equiv y \bmod N$. Die Laufzeit von $\mathcal{B}$ ist $t_\mathcal{B} \approx t_\mathcal{A}$, die Erfolgswahrscheinlichkeit ist mindestens $\epsilon_\mathcal{B} \geq \epsilon_\mathcal{A}$.*

Als technisches Hilfsmittel für den Beweis benötigen wir das folgende Lemma aus [Sha83].

Lemma 31 („Shamir's Trick")**.** *Seien $J, S \in \mathbb{Z}_N$ und $e, f \in \mathbb{Z}$, sodass $\mathrm{ggT}(e, f) = 1$ und*

$$J^f \equiv S^e \bmod N.$$

Es gibt einen effizienten Algorithmus der, gegeben $N \in \mathbb{Z}$ und $(J, S, e, f) \in \mathbb{Z}_N^2 \times \mathbb{Z}^2$, $x \in \mathbb{Z}_N$ berechnet, sodass $x^e \equiv J \bmod N$.

Beweis. Der Algorithmus berechnet zuerst mit Hilfe des erweiterten Euklidischen Algorithmus zwei ganze Zahlen $(\alpha, \beta) \in \mathbb{Z}^2$, sodass

$$\alpha f + \beta e = 1.$$

Diese existieren, da $\mathrm{ggT}(e, f) = 1$ ist. Dann berechnet er

$$x \equiv S^\alpha \cdot J^\beta \bmod N.$$

Es bleibt zu zeigen, dass dieser Wert $x \in \mathbb{Z}_N$ tatsächlich die Gleichung $x^e \equiv J \bmod N$ erfüllt:

$$J^f \equiv S^e \iff J^{f\alpha} J^{e\beta} \equiv S^{e\alpha} J^{e\beta} \iff J^{f\alpha + e\beta} \equiv (S^\alpha J^\beta)^e \iff J \equiv x^e$$

wobei alle Kongruenzen modulo N sind. □

Nun können wir Theorem 30 beweisen.

Beweis von Theorem 30. Algorithmus $\mathcal{B}$ erhält als Eingabe eine Instanz (N, e, y) des RSA-Problems. Es startet den Angreifer, und erhält eine Nachricht $m \in [0, 2^n - 1]$. Den pk berechnet $\mathcal{B}$ als (N, e, J, c) mit

$$J := y \qquad \text{und} \qquad c := J^m \sigma^e \bmod N$$

für eine zufällige Zahl $\sigma \xleftarrow{\$} \mathbb{Z}_N$. Weil σ und x gleichverteilt zufällig über $\mathbb{Z}_N$ sind, sind auch c und J gleich verteilt zufällig. Damit ist pk von einem echten öffentlichen Schlüssel nicht zu unterscheiden. Ausserdem erfüllt σ die Verifikationsgleichung und ist somit eine gültige Signatur. Daher gibt $\mathcal{A}$ mit Wahrscheinlichkeit $\epsilon_{\mathcal{A}}$ eine Fälschung (m^*, σ^*) aus mit $m^* \neq m$ und

$$c \equiv J^{m^*} (\sigma^*)^e \bmod N.$$

Insgesamt gilt also die Gleichung

$$J^m \sigma^e \equiv c \equiv J^{m^*} (\sigma^*)^e \bmod N$$

welche sich umformen lässt zu den zwei Gleichungen

$$J^{m-m^*} \equiv (\sigma^*/\sigma)^e \qquad \text{und} \qquad J^{m^*-m} \equiv (\sigma/\sigma^*)^e,$$

falls $\sigma \in \mathbb{Z}_N^*$ und $\sigma^* \in \mathbb{Z}_N^*$.

Da $m \neq m^*$ gilt, ist ausserdem entweder $m - m^* \in [1, 2^n - 1]$ (nämlich dann, wenn $m > m^*$) oder $m^* - m \in [1, 2^n - 1]$ (nämlich dann, wenn $m^* > m$).

Fall 1: $m - m^* \in [1, 2^n - 1]$**.** Wenn wir nun $f := m - m^* \in [1, 2^n - 1]$ und $S := \sigma^*/\sigma \in \mathbb{Z}_N$ definieren,[2] dann gilt die Gleichung

$$J^f \equiv S^e \bmod N.$$

Dabei ist $\mathrm{ggT}(e, f) = \mathrm{ggT}(e, m - m^*) = 1$, denn e ist eine Primzahl und größer als 2^n. $\mathcal{B}$ kann also „Shamir's Trick" (Lemma 31) auf (N, J, S, e, f) anwenden um x zu berechnen, sodass $x^e \equiv J \bmod N$. Dies ist genau die gesuchte e-te Wurzel von y, denn es ist $J = y$.

Fall 2: $m^* - m \in [1, 2^n - 1]$**.** Dieser Fall ist identisch zu Fall 1, jedoch mit $f := m^* - m \in [1, 2^n - 1]$ und $S := \sigma/\sigma^* \in \mathbb{Z}_N$. □

2.4 Von EUF-naCMA-Sicherheit zu EUF-CMA-Sicherheit

Eine sehr wichtige Anwendung von Einmalsignaturen ist die „Verstärkung" der Sicherheit von Signaturverfahren. In diesem Kapitel werden wir eine Transformation beschreiben, die mit Hilfe eines EUF-1-naCMA-sicheren Einmalsignaturverfahrens ein EUF-naCMA-sicheres Signaturverfahren $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vfy}')$ in ein EUF-CMA-sicheres Signaturverfahren $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ transformiert.

[2] Wir sind hier etwas ungenau, denn σ muss nicht unbedingt invertierbar modulo N sein. Die Wahrscheinlichkeit, dass $\sigma \equiv 0 \bmod N$ ist, ist jedoch vernachlässigbar klein. Falls $\sigma \not\equiv 0 \bmod N$ und trotzdem nicht-invertierbar ist, dann können wir N faktorisieren durch Berechnung von $\mathrm{ggT}(N, \sigma)$ und so die gegebene RSA-Probleminstanz lösen.

Unser eigentliches Ziel bei der Konstruktion von Signaturen ist in der Regel EUF-CMA-Sicherheit. Dieses Ziel ist jedoch nicht leicht erreichbar. Die Transformation vereinfacht unsere Aufgabe, da es ausreicht eine EUF-1-naCMA-sichere Einmalsignatur und ein EUF-naCMA-sicheres Signaturverfahren zu konstruieren. Beides ist wesentlich einfacher zu erreichen als direkte EUF-CMA-Sicherheit, denn

- EUF-1-naCMA-sichere Einmalsignaturverfahren sind sehr leicht konstruierbar. Wir haben bereits einige einfache Konstruktionen von generischen Annahmen (der Existenz von Einwegfunktionen) und von recht schwachen, konkreten Komplexitätsannahmen gesehen.

- EUF-naCMA-sichere Signaturen sind wesentlich leichter zu konstruieren als EUF-CMA-sichere Signaturen. Der Vorteil bei der Konstruktion (mit Sicherheitsbeweis) ist hier, dass der Angreifer im Sicherheitsbeweis dem Challenger mitteilen muss für welche Nachrichten er eine Signatur sehen möchte *bevor* dieser den *public key* erzeugt. Daher kann der *public key* im Beweis so aufgesetzt werden, dass für alle vom Angreifer gewählten Nachrichten Signaturen erstellt werden können. Dies ist schon einmal die „halbe Miete“.

  Die andere Hälfte, nämlich dass wir im Beweis aus der gefälschten Signatur eine Lösung zu einem schwierigen Problem extrahieren können, muss auf eine andere Art und Weise sicher gestellt werden.

Um dann am Ende ein EUF-CMA-sicheres Verfahren zu erhalten, wenden wir einfach die Transformation auf das Einmalsignaturverfahren und das EUF-naCMA-sichere Verfahren an.

Die Idee der hier vorgestellten Transformation stammt von Even, Goldwasser und Micali [EGM96].

Die Transformation. Im Folgenden sei $\Sigma_1 = (\mathsf{Gen}^{(1)}, \mathsf{Sign}^{(1)}, \mathsf{Vfy}^{(1)})$ ein (Einmal-)Signaturverfahren und $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vfy}')$ ein Signaturverfahren. Wir beschreiben ein neues Signaturverfahren $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$, welches $\Sigma^{(1)}$ und Σ' als Bausteine benutzt.

$\mathsf{Gen}(1^k)$**.** Zur Schlüsselerzeugung wird ein Schlüsselpaar $(pk', sk') \xleftarrow{\$} \mathsf{Gen}'(1^k)$ mit dem Schlüsselerzeugungsalgorithmus von Σ' generiert. $pk := pk'$, $sk := sk'$.

$\mathsf{Sign}(sk, m)$**.** Eine Signatur für Nachricht m wird in zwei Schritten berechnet.

1. Zunächst wird ein temporäres Schlüsselpaar $(pk^{(1)}, sk^{(1)}) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^k)$ mit dem Schlüsselerzeugungsalgorithmus von $\Sigma^{(1)}$ erzeugt. Die Nachricht m wird mit dem temporären $sk^{(1)}$ signiert:

$$\sigma^{(1)} := \mathsf{Sign}^{(1)}(sk^{(1)}, m).$$

2. Dann wird $pk^{(1)}$ mit dem langlebigen sk signiert:

$$\sigma' := \mathsf{Sign}'(sk, pk^{(1)}).$$

Die Signatur σ für Nachricht m ist $\sigma := (pk^{(1)}, \sigma^{(1)}, \sigma')$.

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus prüft ob beide in $\sigma = (pk^{(1)}, \sigma^{(1)}, \sigma')$ enthaltenen Signaturen gültig sind. Die Signatur wird akzeptiert, also es wird 1 ausgegeben, wenn

$$\mathsf{Vfy}^{(1)}(pk^{(1)}, m, \sigma^{(1)}) = 1 \qquad \text{und} \qquad \mathsf{Vfy}'(pk, pk^{(1)}, \sigma') = 1.$$

Ansonsten wird 0 ausgegeben.

Die Idee der Transformation ist also zuerst jede Nachricht mit einem frisch generierten Schlüssel von Σ_1 zu signieren. Der frisch generierte Schlüssel wird dann mit Hilfe des langlebigen Schlüssels sk „zertifiziert".

Theorem 32. *Für jeden PPT-Angreifer $\mathcal{A}$, der die EUF-CMA-Sicherheit von Σ in Zeit $t_{\mathcal{A}}$ mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{A}}$ bricht, und dabei höchstens q Signaturanfragen stellt, existiert ein PPT-Angreifer $\mathcal{B}$, der in Zeit $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ läuft und*

- *entweder die EUF-1-naCMA-Sicherheit von $\Sigma^{(1)}$ bricht, mit einer Erfolgswahrscheinlichkeit von mindestens*
$$\epsilon^{(1)} \geq \frac{\epsilon_{\mathcal{A}}}{2q},$$
- *oder die EUF-naCMA-Sicherheit von Σ' bricht, mit einer Erfolgswahrscheinlichkeit von mindestens*
$$\epsilon' \geq \frac{\epsilon_{\mathcal{A}}}{2}.$$

Falls wir annehmen, dass sowohl Σ' als auch $\Sigma^{(1)}$ sicher sind, also ϵ' und $\epsilon^{(1)}$ vernachlässigbar klein sind für alle Polynomialzeit-Angreifer $\mathcal{B}$, so muss auch $\epsilon_{\mathcal{A}}$ vernachlässigbar klein sein.

Beweis. Jeder EUF-CMA-Angreifer $\mathcal{A}$ stellt eine Reihe von adaptiven Signatur-Anfragen $m_1, \ldots, m_q$, $q \geq 0$, auf welche er als Antwort Signaturen $\sigma_1, \ldots, \sigma_q$ erhält, wobei jede Signatur σ_i aus drei Komponenten $\sigma_i = (pk_i^{(1)}, \sigma_i^{(1)}, \sigma_i')$ besteht. Wir betrachten zwei verschiedene Ereignisse:

- Ereignis E_0 tritt ein, falls der Angreifer eine gültige Fälschung $(m^*, \sigma^*) = (m^*, (pk^{(1)*}, \sigma^{(1)*}, \sigma'^*))$ ausgibt, sodass
$$pk^{(1)*} = pk_i^{(1)}.$$
für mindestens ein $i \in \{1, \ldots, q\}$.

  Ereignis E_0 tritt also ein, wenn der Angreifer einen temporären *public key* $pk_i^{(1)}$ aus einer der Signaturen $\sigma_i \in \{\sigma_1, \ldots, \sigma_q\}$ wieder benutzt.

- Ereignis E_1 tritt ein, falls der Angreifer eine gültige Fälschung $(m^*, \sigma^*) = (m^*, (pk^{(1)*}, \sigma^{(1)*}, \sigma'^*))$ ausgibt, sodass
$$pk^{(1)*} \neq pk_i^{(1)}.$$
für alle $i \in \{1, \ldots, q\}$.

  Ereignis E_1 tritt also ein, wenn der Angreifer eine Signatur fälscht, die einen *neuen* temporären *public key* $pk^{(1)*}$ enthält.

Jeder erfolgreiche Angreifer ruft entweder Ereignis E_0 oder Ereignis E_1 hervor. Es gilt also

$$\epsilon_{\mathcal{A}} \leq \Pr[E_0] + \Pr[E_1].$$

Die obige Ungleichung impliziert außerdem, dass zumindest eine der beiden Ungleichungen

$$\Pr[E_0] \geq \frac{\epsilon_{\mathcal{A}}}{2} \quad \text{oder} \quad \Pr[E_1] \geq \frac{\epsilon_{\mathcal{A}}}{2} \tag{2.1}$$

erfüllt sein muss.

Angriff auf die EUF-1-naCMA-Sicherheit von $\Sigma^{(1)}$. Angreifer $\mathcal{B}$ versucht die EUF-1-naCMA-Sicherheit von $\Sigma^{(1)}$ zu brechen, indem er den EUF-CMA-Challenger für $\mathcal{A}$ simuliert. Dazu geht er wie folgt vor.

$\mathcal{B}$ generiert ein Schlüsselpaar $(pk, sk) \xleftarrow{\$} \mathsf{Gen}'(1^k)$ und rät einen Index $\nu \xleftarrow{\$} \{1, \ldots, q\}$. Angreifer $\mathcal{A}$ erhält pk als Eingabe. Falls $\mathcal{A}$ eine Signatur für Nachricht m_i anfragt, dann signiert $\mathcal{B}$ diese Nachricht so:

- Falls $i \neq \nu$, dann erzeugt $\mathcal{B}$ einen Schlüssel $(pk_i^{(1)}, sk_i^{(1)}) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^k)$ und benutzt diesen Schlüssel, um eine Signatur $\sigma_i = (pk_i^{(1)}, \sigma_i^{(1)}, \sigma_i')$ für m_i zu erstellen:

  $$\sigma_i^{(1)} := \mathsf{Sign}^{(1)}(sk_i^{(1)}, m_i) \qquad \text{und} \qquad \sigma_i' := \mathsf{Sign}'(sk, pk_i^{(1)}).$$

  Dies ist offensichtlich eine gültige Signatur für Nachricht m_i.

- Um Nachricht m_ν zu signieren geht $\mathcal{B}$ anders vor. In diesem Fall gibt er m_ν an seinen EUF-1-naCMA-Challenger $\mathcal{C}$ aus. Als Antwort erhält er vom Challenger einen *public key* $pk_\nu^{(1)}$ und eine gültige Signatur $\sigma_\nu^{(1)}$. $\mathcal{B}$ signiert nun noch $pk_\nu^{(1)}$, indem er

  $$\sigma_\nu' := \mathsf{Sign}'(sk, pk_\nu^{(1)})$$

  berechnet und gibt $\sigma_\nu = (pk_\nu^{(1)}, \sigma_\nu^{(1)}, \sigma_\nu')$ als Signatur von m_ν aus. Auch diese Signatur ist offensichtlich gültig.

$\mathcal{B}$ simuliert den EUF-CMA-Challenger für $\mathcal{A}$ perfekt. Wenn Ereignis E_0 eintritt, dann gibt $\mathcal{A}$ eine Nachricht $m^* \notin \{m_1, \ldots, m_q\}$ mit gültiger Signatur σ^* aus, sodass ein Index $i \in \{1, \ldots, q\}$ existiert mit

$$\sigma^* = (pk_i^{(1)}, \sigma^{(1)*}, \sigma_i'^*).$$

Die Signatur σ^* enthält also den i-ten temporären Schlüssel $pk_i^{(1)}$. Da der zufällige Index ν perfekt vor $\mathcal{A}$ verborgen ist, ist dies mit Wahrscheinlichkeit $1/q$ genau der öffentliche Schlüssel $pk_\nu^{(1)}$ den $\mathcal{B}$ vom EUF-1-naCMA-Challenger erhalten hat. Da $m^* \neq m_\nu$ gilt, kann $\mathcal{B}$ in diesem Falle das Tupel $(m^*, \sigma^{(1)*})$ als Fälschung an seinen Challenger $\mathcal{C}$ ausgeben und so das EUF-1-naCMA-Experiment mit einer Erfolgswahrscheinlichkeit von mindestens

$$\epsilon^{(1)} \geq \frac{\Pr[E_0]}{q} \tag{2.2}$$

gewinnen. Die Laufzeit von $\mathcal{B}$ entspricht im Wesentlichen der Laufzeit von $\mathcal{A}$, plus einem kleinen Overhead.

Angriff auf die EUF-naCMA-Sicherheit von Σ'. Angreifer $\mathcal{B}$ versucht die EUF-naCMA-Sicherheit von Σ' wie folgt zu brechen, indem er wieder den EUF-CMA-Challenger für $\mathcal{A}$ simuliert. Diesmal generiert $\mathcal{B}$ q Schlüsselpaare

$$(pk_i^{(1)}, sk_i^{(1)}) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^k), \qquad i \in \{1, \ldots, q\}.$$

Die Liste $(pk_1^{(1)}, \ldots, pk_q^{(1)})$ gibt $\mathcal{B}$ seinem EUF-naCMA-Challenger, um Signaturen für die „Nachrichten“ $pk_i^{(1)}$ zu erhalten. Als Antwort erhält $\mathcal{B}$ vom Challenger einen öffentlichen Schlüssel pk zusammen mit Signaturen $(\sigma'_1, \ldots, \sigma'_q)$ sodass für alle $i \in \{1, \ldots, q\}$ die Signatur σ'_i eine gültige Signatur für „Nachricht“ $pk_i^{(1)}$ ist.

Nun startet $\mathcal{B}$ den EUF-CMA-Angreifer $\mathcal{A}$ mit dem öffentlichen Schlüssel pk. Falls $\mathcal{A}$ eine Signatur für Nachricht m_i anfragt, dann signiert $\mathcal{B}$ diese Nachricht mit Hilfe des ihm bekannten temporären Schlüssels $sk_i^{(1)}$, indem er

$$\sigma_i^{(1)} := \mathsf{Sign}^{(1)}(sk_i^{(1)}, m_i)$$

berechnet. Dann gibt er $\sigma_i = (pk_i^{(1)}, \sigma_i^{(1)}, \sigma'_i)$ als Signatur für m_i aus. Dies ist eine gültige Signatur.

$\mathcal{B}$ simuliert den EUF-CMA-Challenger für $\mathcal{A}$ wieder perfekt. Wenn Ereignis E_1 eintritt, dann gibt $\mathcal{A}$ eine Nachricht $m^* \notin \{m_1, \ldots, m_q\}$ mit gültiger Signatur $\sigma^* = (pk^{(1)*}, \sigma^{(1)*}, \sigma'^*)$ aus, wobei σ^* einen neuen Wert $pk^{(1)*}$ zusammen mit gültiger Signatur σ'^* enthält. Ist dies der Fall, dann kann $\mathcal{B}$ das Tupel

$$(pk^{(1)*}, \sigma'^*)$$

ausgeben um das EUF-naCMA-Experiment mit Wahrscheinlichkeit mindestens

$$\epsilon' \geq \Pr[E_1]. \tag{2.3}$$

zu gewinnen. Die Laufzeit von $\mathcal{B}$ entspricht wieder im Wesentlichen der Laufzeit von $\mathcal{A}$, plus einem kleinen Overhead.

Insgesamt ergibt sich also, durch Einsetzen der Ungleichungen (2.2) und (2.3) in die Ungleichungen aus (2.1), dass zumindest eine der beiden Ungleichungen

$$\epsilon^{(1)} \geq \Pr[E_0] \geq \frac{\epsilon_{\mathcal{A}}}{2q} \quad \text{oder} \quad \epsilon' \geq \Pr[E_1] \geq \frac{\epsilon_{\mathcal{A}}}{2}$$

gelten muss. □

Excercise 33. Wenden Sie die obige Transformation auf die Signaturverfahren aus Kapitel 2.3.1 und Kapitel 2.3.2 and. Geben Sie eine vollständige Beschreibung des jeweils resultierenden EUF-1-CMA-sicheren Einmalsignaturverfahrens an.

2.5 Baum-basierte Signaturen

Leider lassen sich Einmalsignaturen nur ein einziges Mal verwenden, ohne dass sie unsicher werden. Dies ist jedoch für die meisten Anwendungen in der Praxis nicht ausreichend, da wir häufig einen *public key* gerne für viele Signaturen verwenden möchten.

Leider ist die Konstruktionen von Mehrfach-Signaturverfahren etwas schwieriger als die Konstruktion von Einmalsignaturen. Es gibt jedoch glücklicherweise eine generische Konstruktion, die aus einem gegebenen Einmalsignaturverfahren ein Mehrfachsignaturverfahren konstruiert, ohne dabei allzu ineffizient zu werden. Die Idee dazu geht auf Ralph Merkle zurück [Mer88], daher werden derartig konstruierte Verfahren auch *Merkle-Signaturen* oder *Merkle-Bäume („Merkle-Trees“)* genannt.

2.5.1 q-mal Signaturen

Aus einem Einmalsignaturverfahren $\Sigma^{(1)} = (\mathsf{Gen}^{(1)}, \mathsf{Sign}^{(1)}, \mathsf{Vfy}^{(1)})$ lässt sich leicht ein (sehr ineffizientes) q-mal Signaturverfahren konstruieren, indem man einfach q Verfahren parallel anwendet, wie wir in diesem Kapitel beschreiben. Wir erklären dieses Verfahren nur, um später die Idee von Merkle-Signaturen besser erklären zu können. Es ist kein Verfahren, das man sich allzu gut merken muss.

Das Signaturverfahren, das wir im Folgenden betrachten, ist *zustandsbehaftet („stateful“)*. Das bedeutet, dass der Signierer sich einen Zustand st speichern muss, der nach jeder erzeugten Signatur aktualisiert wird. Ein solches zufallsbehaftetes Verfahren ist natürlich nicht immer besonders handlich, insbesondere dann wenn man die Signaturerstellung verteilt einsetzen will (zum Beispiel auf mehreren Servern in einer Cloud) bereitet es Probleme, dass alle Signierer einen gemeinsamen Zustand teilen müssen. Daher werden in der Praxis zustandslose Verfahren eingesetzt.

$\mathsf{Gen}(1^k)$. Der öffentliche Schlüssel des q-mal Verfahrens besteht aus q öffentlichen Schlüsseln des Einmalsignaturverfahrens. Es wird ein Zähler $st := 1$ initialisiert. Der geheime Schlüssel besteht aus den entsprechenden q geheimen Schlüsseln und dem Zähler st:[3]

$$pk = (pk_1, \ldots, pk_q), \quad sk = (sk_1, \ldots, sk_q, st), \quad (pk_i, sk_i) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^k)\, \forall\, i \in \{1, \ldots, q\}.$$

$\mathsf{Sign}(sk, m)$. Sei $st = i$, sodass $1 \leq i \leq q$. Um die i-te Signatur für eine gegebene Nachricht m_i zu erzeugen, berechnet der Signierer

$$\sigma_i := \mathsf{Sign}^{(1)}(sk_i, m_i).$$

Die Signatur $\sigma = (\sigma_i, i)$ besteht aus der Signatur σ_i sowie einer Zahl $i \in \{1, \ldots, q\}$, die dem Verifizierer sagt, dass er den Schlüssel pk_i zur Verifikation benutzen soll. Zum Schluss wird der Zustand inkrementiert, $st := st + 1$.

$\mathsf{Vfy}(pk, m, \sigma)$. Wenn der Verifizierer eine Signatur $\sigma = (\sigma_i, i)$ erhält, dann prüft er, ob

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

ist. Falls ja, so akzeptiert er die Signatur und gibt 1 aus. Ansonsten gibt er 0 aus.

Das obige Verfahren ist also zustandsbehaftet. Der Zustand besteht aus einem Zähler, der mitzählt wieviele Signaturen bislang erstellt worden sind. Sinn und Zweck des Zählers ist, dass jeder geheime Schlüssel nur einmal benutzt wird. Wir erwähnen schonmal, dass dieser Zustand nicht immer unbedingt nötig ist. Wir werden später eine Technik erklären, wie man ihn manchmal (auch im hier beschriebenen Verfahren) vermeiden kann.

[3] Der Zustand st ist kein Geheimnis, wir müssen ihn nur irgendwo beim Signierer speichern. Da der sk sowieso gespeichert werden muss, bietet sich dieser auch als Speicherplatz für st an.

Excercise 34. Zeigen Sie, dass das obige Verfahren ein EUF-naCMA-sicheres (EUF-CMA-sicheres) q-mal Signaturverfahren ist, wenn $\Sigma^{(1)}$ EUF-1-naCMA-sicher (EUF-1-CMA-sicher) ist.

Das obige Verfahren hat einige große Nachteile. Insbesondere ist die Größe der öffentlichen und geheimen Schlüssel linear in der Anzahl der Signaturen, die jemals ausgestellt werden. Der Vorteil ist allerdings, dass die Größe einer Signatur σ recht klein ist, nämlich konstant.

$$|pk| = O(q), \qquad |sk| = O(q), \qquad |\sigma| = O(1).$$

2.5.2 Tausche kurze Signaturen gegen kleine öffentliche Schlüssel!

Mit Hilfe einer kollisionsresistenten Hashfunktion $H : \{0,1\}^* \to \{0,1\}^n$ lässt sich die Größe des öffentlichen Schlüssels pk jedoch leicht reduzieren. Die Idee ist, anstatt der Liste $(pk_1, \ldots, pk_q)$ aller q öffentlichen Schlüssel einfach eine Hashfunktion sowie den Hashwert dieser Liste als pk zu veröffentlichen. Also

$$pk := (H, y),$$

wobei $y = H(pk_1, \ldots, pk_q)$.

Damit ein Verifizierer die Gültigkeit einer gegebenen Signatur verifizieren kann, muss man nun allerdings die Liste in der Signatur mit senden, also besteht eine Signatur bei diesem Verfahren mit gehashtem öffentlichem Schlüssel aus $\sigma = (\sigma_i, i, (pk_1, \ldots, pk_q))$. Der Verifizierer in diesem modifizierten Verfahren prüft nun, ob

$$\mathsf{Vfy}^{(1)}(pk_i, \sigma_i, m_i) \stackrel{?}{=} 1 \qquad \text{und} \qquad y \stackrel{?}{=} H(pk_1, \ldots, pk_q).$$

Falls beides erfüllt ist, so gibt er 1 aus. Ansonsten 0.

Remark 35. Man kann beweisen, dass dieses Verfahren ein EUF-naCMA-sicheres (EUF-CMA-sicheres) q-mal Signaturverfahren ist, wenn $\Sigma^{(1)}$ EUF-1-naCMA-sicher (EUF-1-CMA-sicher) ist und die Hashfunktion kollisionsresistent. Wir lassen dies jedoch aus.

Diese „gehashte“ Variante des ursprünglichen q-mal Signaturverfahrens hat nun zwar einen öffentlichen Schlüssel konstanter Größe, jedoch dafür größere Signaturen.

$$|pk| = O(1), \qquad |sk| = O(q), \qquad |\sigma| = O(q).$$

Man kann also offensichtlich eine kurze Signaturgröße „eintauschen“ gegen kurze öffentliche Schlüssel. Geht das vielleicht noch etwas cleverer?

2.5.3 Clevere Kompression der öffentlichen Schlüssel: Merkle-Bäume

Die Idee von Merkle-Bäumen [Mer88] ist im Wesentlichen, die öffentlichen Schlüssel auf eine schlauere Art zu „komprimieren“ als wir es im vorigen Kapitel getan haben. Die geniale Idee ist dabei, dass die Hashfunktion nicht bloß einmalig verwendet wird, sondern auf geschickte Art mehrmals rekursiv. Im Folgenden nehmen wir an, dass $q = 2^t$ ist für eine natürliche Zahl t.

Betrachten wir also das Problem, eine Liste $(pk_1, \ldots, pk_{2^t})$ von öffentlichen Schlüsseln so geschickt zu komprimieren, dass wir am Ende einen kleinen öffentlichen Schlüssel pk erhalten, aber gleichzeitig die Größe der Signaturen dabei nicht allzu stark anwächst. Eine Möglichkeit, die Liste zu komprimieren ist durch die folgende Berechnung:

1. Zuerst wird der Hashwert $h_{t,i} := H(pk_i)$, $i \in \{1, \ldots, 2^t\}$, für alle Schlüssel in der Liste berechnet.

2. Dann wird rekursiv berechnet:

$$h_{j-1,i} := H(h_{j,2i-1}||h_{j,2i}) \quad \forall j \in \{t, \ldots, 1\}, i \in \{1, \ldots, 2^{j-1}\}$$

Diese etwas unübersichtliche Rechenvorschrift ist für den Fall $t = 3$ in Abbildung 2.3 dargestellt. Man beachte hier, dass wir einen binären Baum aufbauen, dessen Knoten die Hashwerte $h_{j,i}$ sind. Wir brauchen ein paar Begriffe über Bäume.

- Wir sagen, dass $h_{0,1}$ die *Wurzel* des Baumes ist.
- Der *Pfad* von einem Knoten $h_{j,i}$ zur Wurzel besteht aus allen Knoten, die auf der kürzesten Verbindung von $h_{j,i}$ zur Wurzel liegen. In Abbildung 2.3 ist der Pfad von $h_{3,2}$ zur Wurzel durch eine durchgezogene Linie dargestellt.
- Wir sagen, dass ein Knoten $h_{j,i}$ der *Vater* von $h_{j',i'}$ ist, falls $j+1 = j'$ und $i' \in \{2i-1, 2i\}$ gilt.
- Wir sagen dass zwei Knoten *Geschwister* sind, wenn sie einen gemeinsamen Vater haben. Zum Beispiel $h_{1,1}$ und $h_{1,2}$ sind Geschwister, da sie den gemeinsamen Vater $h_{0,1}$ haben.
- Der *Co-Pfad* von $h_{j,i}$ zur Wurzel besteht aus allen Geschwistern von einem Knoten, der auf dem Pfad von $h_{j,i}$ zur Wurzel liegt. In Abbildung 2.3 ist der Co-Pfad von $h_{3,2}$ zur Wurzel durch eine gestrichelte (nicht gepunktete) Linie dargestellt.

Der öffentliche Schlüssel des Baum-basierten Verfahrens besteht nun aus der Wurzel $pk := h_{0,1}$, des vollständigen binären Baumes, der durch die $h_{j,i}$-Hashwerte aufgespannt wird. Die Wurzel $h_{0,1}$ ist im Wesentlichen wieder eine komprimierte Darstellung der Liste $(pk_1, \ldots, pk_q)$, wie im vorigen Abschnitt, jedoch ist nun der Hashwert nicht einfach als $H(pk_1, \ldots, pk_q)$ berechnet, sondern etwas komplizierter. Es wird sich herausstellen, dass dies ziemlich clever ist.

Eine Signatur für die i-te Nachricht besteht nun aus aus dem i-ten Schlüssel pk_i , gemeinsam mit allen Werten die nötig sind um zu verifizieren, dass $h_{t,i} = H(pk_i)$ tatsächlich ein Kind von $h_{0,1}$ ist, indem man den direkten Pfad von $h_{t,i}$ hoch bis zur Wurzel $h_{0,1}$ des Baumes nachberechnet.

Um diesen Pfad nachzuberechnen brauchen wir zunächst den Startwert $h_{t,i}$, den man als Hashwert $h_{t,i} = pk_i$ berechnen kann. Die übrigen benötigten Werte sind genau die Werte, die auf dem „Co-Pfad“ von $h_{t,i}$ hoch bis zur Wurzel $h_{0,1}$ liegen.

Example 36. Betrachten wir als Beispiel den Wert $h_{3,2}$ in Abbildung 2.3. Dann besteht der Pfad von $h_{3,2}$ hoch bis zur Wurzel aus der Liste der Hashwerte $P = (h_{3,2}, h_{2,1}, h_{1,1}, h_{0,1})$. Der Pfad von $h_{3,2}$ nach $h_{0,1}$ kann wie folgt nachberechnet werden:

1. Der Wert $h_{3,2}$ kann durch $h_{3,2} = H(pk_2)$ berechnet werden.
2. Um den Wert $h_{2,1}$ nachzuberechnen, benötigen wir die beiden Werte $(h_{3,1}, h_{3,2})$. Der Wert $h_{3,2}$ ist schon bekannt, daher wird $h_{3,1}$ in der Signatur mit gesendet.

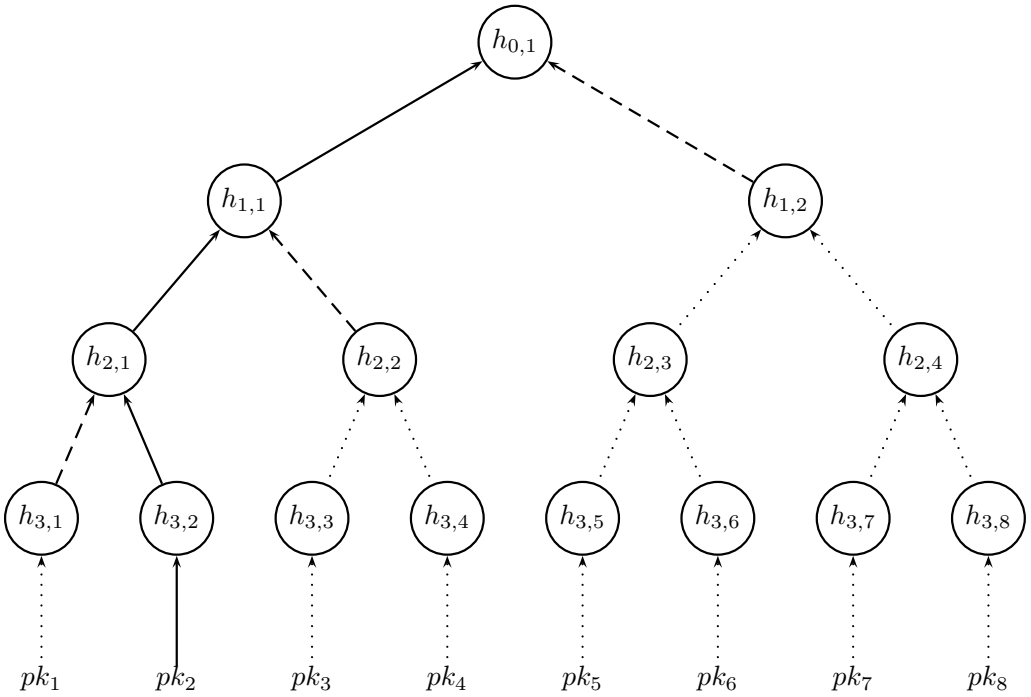

Abbildung 2.3: Merkle-Baum mit Tiefe 3.

3. Um den Wert $h_{1,1}$ nachzuberechnen, benötigen wir die beiden Werte $(h_{2,1}, h_{2,2})$. Wieder ist ein Wert bekannt, nämlich $h_{2,1}$, daher wird nur $h_{2,2}$ in der Signatur mit gesendet.

4. Um zum Schluss den Wert $h_{0,1}$ nachzuberechnen, benötigen wir die beiden Werte $(h_{1,1}, h_{1,2})$. Wieder ist ein Wert bekannt, nämlich $h_{1,1}$, daher wird nur $h_{1,2}$ in der Signatur mit gesendet.

Eine Signatur, für die pk_2 verwendet wurde, besteht also aus der Einmalsignatur $\sigma_2 = \mathsf{Sign}(sk_2, m)$, dem Index "2" der dem Verifizierer den verwendeten Pfad mitteilt, sowie den Werten

$$\sigma = (pk_2, h_{3,1}, h_{2,2}, h_{1,2})$$

Die Liste $(h_{3,1}, h_{2,2}, h_{1,2})$ ist genau die Liste der Elemente auf dem Co-Pfad von $h_{3,2}$ zur Wurzel.

Man kann beweisen, dass dieses Signaturverfahren ein EUF-naCMA-sicheres (bzw. EUF-CMA-sicheres) q-mal Signaturverfahren ist, falls das zugrunde liegende Einmalsignaturverfahren EUF-1-naCMA-sicher (bzw. EUF-1-naCMA-sicher) ist, und die Hashfunktion H kollisionsresistent ist.

Man beachte, dass die Größe der Signatur dieses Verfahrens durch die Tiefe t des Baumes beschränkt ist. Um also $q = 2^t$ Signaturen erzeugen zu können, muss man einen Baum der Tiefe t wählen. Die Größe des öffentlichen Schlüssels ist konstant, lediglich die Größe des geheimen Schlüssels ist immernoch linear in der Anzahl q der zu signierenden Nachrichten.

$$|pk| = O(1), \qquad |sk| = O(q), \qquad |\sigma| = O(\log q).$$

Das ist schonmal deutlich besser als die zuvor vorgestellten Signaturverfahren aus Kapitel 2.5.1 und 2.5.2. Diese ineffizienteren Verfahren kann man nun auch wieder vergessen – wir haben sie nur vorgestellt, weil dann die Idee hinter Merkle-Bäume leichter zu erklären ist.

2.5.4 Kurze Darstellung der geheimen Schlüssel

Ein Problem, das wir noch zu lösen haben, ist die Reduktion der Größe der geheimen Schlüssel. Kann man diese auf ähnliche Art komprimieren wie die öffentlichen Schlüssel? Die Antwort auf diese Frage lautet – vielleicht etwas überraschend – ja!

Pseudozufallsfunktionen

Als Werkzeug dafür benötigen wir eine Pseudozufallsfunktion (*„pseudo-random function", PRF*). Intuitiv ist eine Pseudozufallsfunktion eine Funktion, die ununterscheidbar von einer echt zufälligen Funktion ist. Sei also

$$\mathsf{PRF} : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^l$$

eine Funktion, die als Eingabe einen *Seed* $s \in \{0,1\}^k$ sowie einen Bit-String $\alpha \in \{0,1\}^n$ erhält, und einen l-Bit-String $\mathsf{PRF}(s,\alpha) \in \{0,1\}^k$ ausgibt. Sei

$$F : \{0,1\}^n \to \{0,1\}^l$$

eine zufällig gewählte Funktion aus allen möglichen Funktionen von $\{0,1\}^n$ nach $\{0,1\}^l$.

Wir betrachten Algorithmen, die eine Funktion in Form einer Black-Box gegeben haben. Das bedeutet, die Algorithmen dürfen nur Anfragen an die Black-Box stellen, und erhalten den Funktionswert zurück. Solche Algorithmen sollen nicht in der Lage sein die Funktion $\mathsf{PRF}(s,\cdot)$ von der Funktion $F(\cdot)$ zu unterscheiden, wenn der Seed $s \xleftarrow{\$} \{0,1\}^k$ der Pseudozufallsfunktion zufällig gewählt ist. Wir schreiben $\mathcal{A}^{\mathsf{PRF}(s,\cdot)}$ bzw. $\mathcal{A}^{F(\cdot)}$, um zu bezeichnen, dass ein Algorithmus $\mathcal{A}$ Black-Box Anfragen an die Funktion $\mathsf{PRF}(s,\cdot)$ bzw. $F(\cdot)$ stellen kann.

Definition 37. Wir sagen dass PRF eine *Pseudozufallsfunktion* ist, falls für alle Polynomialzeit-Algorithmen $\mathcal{A}$ und für zufällig gewählten Seed $s \xleftarrow{\$} \{0,1\}^k$ gilt,

$$\left|\Pr\left[\mathcal{A}^{\mathsf{PRF}(s,\cdot)}(1^k) = 1\right] - \Pr\left[\mathcal{A}^{F(\cdot)}(1^k) = 1\right]\right| \leq \mathsf{negl}(k)$$

für eine vernachlässigbare Funktion negl in k.

Komprimierung geheimer Schlüssel

Unser Problem ist, eine Liste von geheimen Schlüsseln $(sk_1, \ldots, sk_q)$ zu „komprimieren", wir wollen sie also kompakter darstellen. Diese Schlüssel wurden stets durch

$$(pk_i, sk_i) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^k) \quad \forall i \in \{1, \ldots, q\} \tag{2.4}$$

erzeugt.

Der Schlüsselerzeugungsalgorithmus $\mathsf{Gen}^{(1)}(1^k)$ ist dabei ein *probabilistischer* Algorithmus (dies ist natürlich notwendig um Sicherheit zu gewährleisten). Wir können uns jedoch jeden probabilistischen Algorithmus äquivalenterweise auch als deterministischen Algorithmus

$$\mathsf{Gen}^{(1)}(1^k) = \mathsf{Gen}^{(1)}_{\mathsf{det}}(1^k, r)$$

vorstellen, der einen zufälligen Bit-String r als zusätzliche Eingabe bekommt. Wir berechnen nun alle Schlüsselpaare mit diesem deterministischen Algorithmus, wobei wir den Zufallsstring r jeweils mit einer Pseudozufallsfunktion PRF erzeugen, und zwar als

$$(pk_i, sk_i) \xleftarrow{\$} \mathsf{Gen}^{(1)}_{\mathsf{det}}(1^k, \mathsf{PRF}(s, i)) \quad \forall i \in \{1, \ldots, q\}. \tag{2.5}$$

wobei $s \xleftarrow{\$} \{0,1\}^k$ ein zufällig gewähler Seed ist. Zur Erzeugung der Schlüsselpaare wird also die PRF an der Stelle i ausgewertet. Die Beobachtung ist nun, dass, aufgrund der Sicherheit der PRF und der Tatsache dass $s \xleftarrow{\$} \{0,1\}^k$ zufällig gewählt ist, die so erzeugten Schlüssel ununterscheidbar sind von

$$(pk_i, sk_i) \xleftarrow{\$} \mathsf{Gen}^{(1)}_{\mathsf{det}}(1^k, F(i)) \quad \forall i \in \{1, \ldots, q\},$$

wobei F eine echt zufällige Funktion ist. Und damit sind sie „genauso gut“ wie Schlüssel die wie in (2.4) erzeugt werden.

Das bedeutet, wir müssen im sk nicht mehr alle Schlüssel $sk = (sk_1, \ldots, sk_q)$ speichern. Statt dessen genügt es, wenn wir einen zufälligen Seed $s \xleftarrow{\$} \{0,1\}^k$ wählen, diesen im sk speichern, und uns alle Schlüsselpaare bei Bedarf mit Hilfe von s neu generieren, wie in (2.5) dargestellt.

In Kombination mit dem Merkle-Baum-basierten Signaturverfahren ergibt dies ein Signaturschema, dessen geheimer Schlüssel ebenfalls konstante Größe hat, also

$$|pk| = O(1), \qquad |sk| = O(1), \qquad |\sigma| = O(\log q).$$

Die in diesem Abschnitt vorgestellte, allgemeine Technik zur „Kompression“ von geheimen Schlüsseln ist für viele kryptographische Verfahren anwendbar, insbesondere Baum-basierte Signaturen und ähnliche Konstruktionen. Sie geht zurück auf Oded Goldreich [Gol87]. Diese Technik kann manchmal auch verwendet werden, um zustandsbehaftete Signaturverfahren komplett zustandslos zu machen. Sie ist allerdings im Falle der Hashfunktionen-basierten Merkle-Bäume nicht (bzw. nur eingeschränkt) anwendbar.

2.5.5 Effizientere Varianten

Auf einen großen Nachteil von allen hier vorgestellten Verfahren muss noch hingewiesen werden. In jedem Beispiel mussten *alle* Schlüsselpaare $(pk_1, sk_1), \ldots, (pk_q, sk_q)$ auf einmal berechnet werden. Entweder einmalig bei der Schlüsselerzeugung (im zustandsbehafteten Fall) oder, noch schlimmer, jedes Mal bei der Erzeugung einer Signatur (im zustandslosen Fall). Insbesondere wenn q sehr groß ist, ist dies natürlich extrem ineffizient.

Dieses Problem taucht nicht auf, wenn man den kompletten Baum (nicht bloß die unterste Ebene) mit Hilfe von Einmalsignaturen aufbaut, anstatt einer kollisionsresistenten Hashfunktion. Derartige Verfahren lassen sich mit der Technik von Goldreich [Gol87] sogar vollständig zustandslos konstruieren. Wir gehen hier nicht weiter darauf ein, und verweisen auf [Kat10] für Details.

Excercise 38. Sei $\Sigma^{(1)} = (\mathsf{Gen}^{(1)}, \mathsf{Sign}^{(1)}, \mathsf{Vfy}^{(1)})$ ein Einmalsignaturverfahren, und sei $q = 2^t$. Konstruieren Sie nach obigem Vorbild mit Hilfe von $\Sigma^{(1)}$ ein Baum-basiertes q-mal Signaturverfahren.

Kapitel 3

Chamäleon-Hashfunktionen

Auch wenn der Name es suggeriert: Chamäleon-Hashfunktionen sind keine Hashfunktionen, die ihre Farbe wechseln können. Sie haben aber eine sehr ähnliche Eigenschaft, die entfernt an die Vielseitigkeit eines Chamäleons erinnert. Sie wurden im Jahr 2000 von Krawczyk und Rabin [KR00] eingeführt, und sind mittlerweile ein wichtiger Baustein für viele kryptographische Konstruktionen.

In diesem Kapitel erklären wir zunächst die Idee von Chamäleon-Hashfunktionen, und geben zwei konkrete Konstruktionen an, die auf dem diskreten Logarithmusproblem (DL) und dem RSA-Problem basieren. Diese Konstruktionen sind sehr ähnlich zu den DL- und RSA-basierten Einmalsignaturen, die wir schon kennen. Danach beschreiben wir zwei interessante Anwendungen von Chamäleon-Hashfunktionen im Kontext von Signaturen. Als erste Anwendung stellen wir so genannte *abstreitbare Signaturen* vor. Danach zeigen wir, dass man aus einer Chamäleon-Hashfunktion immer ein Einmalsignaturverfahren konstruieren kann. Als letzte Anwendung zeigen wir, dass man Signaturverfahren konstruieren kann die *strong existential unforgeable* sind. Dies ist, wie die Bezeichnung schon verrät, eine stärkere Sicherheitseigenschaft als *existential unforgeability*, die in manchen Anwendungen notwendig ist.

Neben den hier vorgestellten Beispielen haben Chamäleon-Hashfunktionen viele weitere, wichtige Anwendungen in der Kryptographie, auf die wir hier leider nicht eingehen können. Insbesondere ist an dieser Stelle die Konstruktion von CCA-sicheren Verschlüsselungsverfahren aus so genannten identitätsbasierten Verschlüsselungsverfahren zu nennen [Zha07] (eine Alternative zur Einmalsignatur-basierten Konstruktion aus [CHK04]), und die Verwandtschaft zu bestimmten Identifikationsverfahren (Sigma-Protokollen) [BR08].

3.1 Motivation

Nehmen wir an, ein Kunde K möchte eine Ware einkaufen. Es gibt zwei konkurrierende Händler H_1 und H_2, die diese Ware in gleicher Qualität liefern können. Beide Händler veröffentlichen ihre Preise nicht, sondern verhandeln jedes Mal individuell mit dem Kunden. Am liebsten will ein Händler immer ein kleines bisschen günstiger sein als sein Konkurrent – jedoch weiß der eine Händler leider nicht, welchen Preis der andere Händler dem Kunden gerade genannt hat.

Der Kunde fragt per E-Mail zuerst bei H_1 nach einem Preis für die Ware. Dieser antwortet mit einer E-Mail, die einen echten Schnäppchenpreis von 100 € pro Stück anbietet. Damit die Nachricht authentisch bei K ankommt, ist sie von H_1 digital signiert.

Nun möchte der Kunde den Händler H_2 kontaktieren, und ihn um einen günstigeren Preis bitten. Falls der Kunde einfach so behauptet, dass H_1 nur 100 € pro Stück haben möchte, so würde ihm H_2 das nicht glauben – der Preis ist wirklich sehr günstig (aber immer noch profitabel für den Verkäufer). Da die Nachricht von H_1 digital signiert ist, kann K sie einfach weiterleiten. Die Signatur überzeugt H_2, dass H_1 tatsächlich einen Preis von nur 100 € angeboten hat – und unterbietet ihn, indem er nur 99 € pro Stück verlangt.

In diesem Falle ist eine Signatur einerseits nützlich (denn K kann sicher sein, dass die Nachricht von H_1 nicht verändert wurde), aber auch schädlich, denn K kann die Signatur benutzen, um H_2 davon zu überzeugen, dass H_1 tatsächlich einen unglaublich günstigen Preis angeboten hat. H_1 kann das nicht abstreiten, denn die Signatur ist ja öffentlich verifizierbar.

Es scheint also, als seien Authentizität und Abstreitbarkeit zwei gegensätzliche Eigenschaften, die sich gegenseitig ausschließen. Muss das so sein? Können wir ein Signaturverfahren konstruieren, dass

1. dem Empfänger K erlaubt die Authentizität einer empfangenen Nachricht m zu verifizieren,

2. dem Sender H_1 aber *gleichzeitig* erlaubt gegenüber Dritten (wie zum Beispiel H_2) plausibel abzustreiten, dass eine gewisse Nachricht m gesendet wurde, selbst wenn K dies behauptet und eine gültige Signatur von H_1 über m vorlegt?

Die Antwort lautet – vielleicht etwas überraschend – ja! Signaturverfahren, die eine solche Eigenschaft haben, werden *Chamäleon-Signaturverfahren* genannt [KR00]. Das Werkzeug, das wir zur Konstruktion solcher Verfahren brauchen, sind Chamäleon-Hashfunktionen.

3.2 Definition von Chamäleon-Hashfunktionen

Eine Chamäleon-Hashfunktion besteht aus zwei Algorithmen $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$.

$\mathsf{Gen}_{\mathsf{ch}}(1^k)$**.** Der Erzeugungsalgorithmus erhält als Eingabe den Sicherheitsparameter. Er gibt (ch, τ) aus, wobei ch eine Beschreibung einer Funktion

$$\mathsf{ch} : \mathcal{M} \times \mathcal{R} \to \mathcal{N}$$

ist. Dabei sind $\mathcal{M}, \mathcal{N}, \mathcal{R}$ Mengen, die von der konkreten Konstruktion der Chamäleon-Hashfunktion abhängen. Der Wert τ ist eine Trapdoor („Falltür“) für ch.

$\mathsf{TrapColl}_{\mathsf{ch}}(\tau, m, r, m^*)$**.** Mit Hilfe der Trapdoor τ kann der Trapdoor-Kollisionsalgorithmus $\mathsf{TrapColl}_{\mathsf{ch}}$ Kollisionen für die Funktion finden. Der Algorithmus erhält als Eingabe die Trapdoor τ und $(m, r, m^*) \in \mathcal{M} \times \mathcal{R} \times \mathcal{M}$. Er berechnet einen Wert r^*, sodass

$$\mathsf{ch}(m, r) = \mathsf{ch}(m^*, r^*).$$

Als einzige Sicherheitsanforderung an eine Chamäleon-Hashfunktion stellen wir, dass sie kollisionsresistent ist. Kein effizienter Algorithmus, der die Trapdoor *nicht* kennt, soll in der Lage sein für eine gegebene Funktion ch Kollisionen zu finden.

Definition 39. Wir sagen, dass eine Chamäleon-Hashfunktion $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$ *kollisionsresistent* ist, wenn

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

für alle PPT Algorithmen $\mathcal{A}$.

Man beachte, dass $\mathcal{A}$ nicht die Trapdoor τ von ch als Eingabe erhält – ansonsten könnte er natürlich sehr leicht Kollisionen finden.

3.3 Beispiele für Chamäleon-Hashfunktionen

3.3.1 Chamäleon-Hashfunktion basierend auf dem Diskreten Logarithmusproblem

Wir verwenden die gleiche Definition des diskreten Logarithmusproblems und die gleiche Notation wie in Kapitel 2.3.1. Sei im Folgenden also $\mathbb{G}$ eine endliche abelsche Gruppe mit Generator g und primer Ordnung p. Sei $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$ die folgende Chamäleon-Hashfunktion

$$\mathsf{ch} : \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{G}.$$

$\mathsf{Gen}_{\mathsf{ch}}(1^k)$**.** Der Erzeugungsalgorithmus wählt $x \xleftarrow{\$} \mathbb{Z}_p^*$ und berechnet $h := g^x$. Die Beschreibung der Chamäleon-Hashfunktion besteht aus $\mathsf{ch} := (g,h)$. Gegeben $(m,r) \in \mathbb{Z}_p \times \mathbb{Z}_p$ wird der Funktionswert berechnet als

$$\mathsf{ch}(m,r) = g^m h^r.$$

Die Falltür ist $\tau := x$.

$\mathsf{TrapColl}_{\mathsf{ch}}(\tau, m, r, m^*)$**.** Gegeben (m,r,m^*) wird der gesuchte Wert r^* berechnet als Lösung der Gleichung

$$m + xr \equiv m^* + xr^* \bmod p \iff r^* \equiv \frac{m-m^*}{x} + r \bmod p.$$

Dann gilt natürlich

$$g^m h^r = g^{m^*} h^{r^*}.$$

Theorem 40. *Für jeden PPT-Angreifer $\mathcal{A}$, der als Eingabe (g,h) erhält, in Zeit $t_{\mathcal{A}}$ läuft und mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{A}}$ vier Werte $(m,r,m^*,r^*) \in \mathbb{Z}_p^4$ berechnet, sodass $(m,r) \neq (m^*,r^*)$ und*

$$g^m h^r = g^{m^*} h^{r^*}$$

gilt, existiert ein PPT-Angreifer $\mathcal{B}$, der das diskrete Logarithmusproblem in $\mathbb{G}$ löst in Zeit $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{B}} \geq \epsilon_{\mathcal{A}}$.

Excercise 41. Beweisen Sie Theorem 40. Tipp: Der Beweis ist sehr ähnlich zum Sicherheitsbeweis des Einmalsignaturverfahrens aus Kapitel 2.3.1.

3.3.2 Chamäleon-Hashfunktion basierend auf der RSA-Annahme

Wir verwenden die gleiche Definition des RSA-Problems und die gleiche Notation wie in Kapitel 2.3.2. Sei $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$ die folgende Chamäleon-Hashfunktion

$$\mathsf{ch} : [0, 2^n - 1] \times \mathbb{Z}_N \setminus \{0\} \to \mathbb{Z}_N.$$

$\mathsf{Gen}_{\mathsf{ch}}(1^k)$. Sei $n \in \mathbb{N}$. Der Erzeugungsalgorithmus erzeugt einen RSA-Modulus $N = PQ$ sowie eine Primzahl $e > 2^n$ mit $\mathrm{ggT}(e, \phi(N)) = 1$ und berechnet $d := e^{-1} \bmod \phi(N)$. Außerdem wird eine Zahl $J \xleftarrow{\$} \mathbb{Z}_N$ zufällig gewählt. Die Beschreibung der Funktion besteht aus (N, e, J). Gegeben $(m, r) \in \{0,1\}^n \times \mathbb{Z}_N \setminus \{0\}$ wird der Funktionswert berechnet als

$$\mathsf{ch}(m, r) = J^m r^e.$$

Die Falltür ist der geheime RSA-Schlüssel $\tau := d$.

$\mathsf{TrapColl}_{\mathsf{ch}}(\tau, m, r, m^*)$. Gegeben $(m, r, m^*) \in [0, 2^n - 1] \times \mathbb{Z}_N \setminus \{0\} \times [0, 2^n - 1]$, wird der gesuchte Wert r^* berechnet als Lösung der Gleichung

$$J^m r^e \equiv J^{m^*} (r^*)^e \bmod N \iff r^* \equiv \left(J^{m-m^*} \cdot r^e\right)^d \bmod N.$$

Falls dieser Wert nicht existiert, weil J^{m^*} nicht invertierbar ist, wird ein Fehlersymbol $\perp$ ausgegeben. (Dies passiert jedoch „fast niemals“. Warum?)

Diese Chamäleon-Hashfunktion ist kollisionsresistent unter der RSA-Annahme. Das folgende Theorem besagt, dass wir aus einem Algorithmus, der die Kollisionsresistenz bricht, einen Algorithmus zum Lösen des RSA-Problems konstruieren können.

Theorem 42. *Sei (N, e, y) eine Instanz des RSA-Problems, sodass $e > 2^n$ eine Primzahl ist. Für jeden PPT-Angreifer $\mathcal{A}$, der als Eingabe (N, e, J) erhält, in Zeit $t_{\mathcal{A}}$ läuft und mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{A}}$ vier Werte*

$$(m, r, m^*, r^*) \in [0, 2^n - 1] \times \mathbb{Z}_N \setminus \{0\} \times [0, 2^n - 1] \times \mathbb{Z}_N$$

berechnet, sodass $(m, r) \neq (m^*, r^*)$ und

$$J^m r^e \equiv J^{m^*} (r^*)^e \bmod N$$

gilt, existiert ein PPT-Angreifer $\mathcal{B}$, der, gegeben (N, e, y) mit $y \xleftarrow{\$} \mathbb{Z}_N$, $x \in \mathbb{Z}_N$ berechnet mit $x^e \equiv y \bmod N$. Der Angreifer $\mathcal{B}$ läuft in Zeit $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{B}} \geq \epsilon_{\mathcal{A}}$.

Excercise 43. Beweisen Sie Theorem 42. Tipp: Der Beweis ist sehr ähnlich zum Sicherheitsbeweis des Einmalsignaturverfahrens aus Kapitel 2.3.2.

3.4 Chamäleon-Signaturen

Bei Chamäleon-Signaturverfahren wird eine Nachricht etwas anders signiert als bei normalen digitalen Signaturen. Bei der Signaturerstellung wie auch bei der Verifikation wird zusätzlich

eine Chamäleon-Hashfunktion *des Empfängers* verwendet. Daher bekommen sowohl der Signaturalgorithmus als auch der Verifikationsalgorithmus eine zusätzliche Eingabe, nämlich die Chamäleon-Hashfunktion des Empfängers.

Anstatt zuerst eine allgemeine Definition eines Chamäleon-Signaturverfahrens anzugeben, beschreiben wir direkt ein Verfahren. Sei im Folgenden

$$\mathsf{ch} : \mathcal{M} \times \mathcal{R} \to \mathcal{N}$$

eine vom Empfänger durch $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^k)$ generierte Chamäleon-Hashfunktion.[1] Sei $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vfy}')$ ein Signaturverfahren. Wir definieren ein Chamäleon-Signaturverfahren wie folgt.

$\mathsf{Gen}(1^k)$**:** Der Schlüsselerzeugungsalgorithmus erhält als Eingabe den Sicherheitsparameter. Er berechnet das Schlüsselpaar durch $(pk, sk) \xleftarrow{\$} \mathsf{Gen}'(1^k)$ und gibt (pk, sk) aus.

$\mathsf{Sign}(sk, m, \mathsf{ch})$**:** Der Signaturalgorithmus wählt $r \xleftarrow{\$} \mathcal{R}$ und berechnet zuerst $m' := \mathsf{ch}(m, r)$ und dann $\sigma' := \mathsf{Sign}'(sk, m')$. Die Signatur besteht aus

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

$\mathsf{Vfy}(pk, m, \sigma, \mathsf{ch})$**:** Der Verifikationsalgorithmus erhält $\sigma = (\sigma', r)$, und prüft ob

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

Falls dies erfüllt ist, gibt er 1 aus. Ansonsten 0.

Wir müssen nun zunächst zeigen, dass dieses Verfahren durch die Verwendung der Chamäleon-Hashfunktion nicht unsicherer wird – wenn die Chamäleon-Hashfunktion kollisionsresistent ist. Wir betrachten dazu Angreifer, die *nicht* die Trapdoor τ von ch kennen – ansonsten wird das Verfahren trivialerweise unsicher.

Das EUF-CMA Sicherheitsexperiment für ein Chamäleon-Signaturverfahren mit Angreifer $\mathcal{A}$, Challenger $\mathcal{C}$, Signaturverfahren $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ und Chamäleon-Hashfunktion $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$ ist identisch zum EUF-CMA-Spiel, außer dass der Angreifer als zusätzliche Eingabe die Chamäleon-Hashfunktion des Empfängers erhält. Der Vollständigkeit halber nochmal das gesamte Experiment (siehe auch Abbildung 3.1):

1. Der Challenger $\mathcal{C}$ generiert ein Schlüsselpaar des Senders $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^k)$ und eine Chamäleon Hashfunktion des Empfängers $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^k)$. Der Angreifer erhält (pk, ch).

2. Nun darf der Angreifer $\mathcal{A}$ beliebige Nachrichten $m_1, \ldots, m_q$ vom Challenger signieren lassen. Dazu sendet er Nachricht m_i an den Challenger. Dieser berechnet $\sigma_i = \mathsf{Sign}(sk, m_i, \mathsf{ch})$ und antwortet mit σ_i. Dieser Schritt kann vom Angreifer beliebig oft wiederholt werden. Wenn wir Angreifer mit polynomiell beschränkter Laufzeit betrachten werden, ist $q = q(k)$ ein Polynom im Sicherheitsparameter.

[1] Ein Empfänger könnte ch auch bösartig generieren, sodass er später plausibel belegen kann, dass er *keine* Kollision für ch finden kann. Wir nehmen zunächst mal an, dass dies nicht der Fall ist, und argumentieren später, dass man das sicher stellen kann.

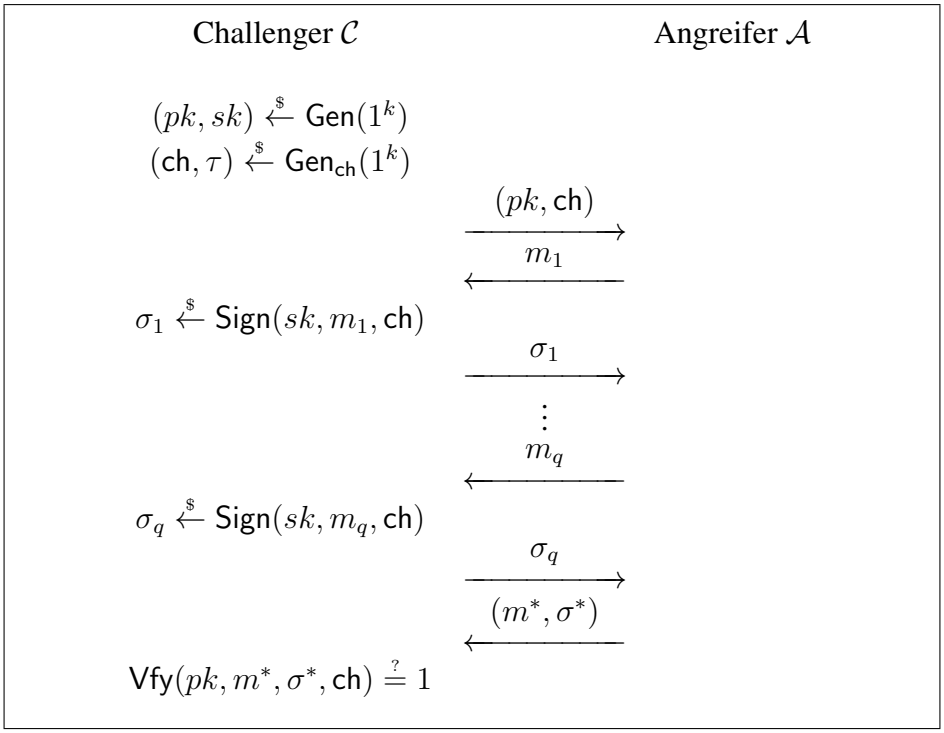

Abbildung 3.1: Das EUF-CMA Sicherheitsexperiment für Chamäleon-Signaturen.

3. Am Ende gibt $\mathcal{A}$ eine Nachricht m^* mit Signatur σ^* aus. Er „gewinnt“ das Spiel, wenn

$$\mathsf{Vfy}(pk, m^*, \sigma^*, \mathsf{ch}) = 1 \qquad \text{und} \qquad m^* \notin \{m_1, \ldots, m_q\}.$$

$\mathcal{A}$ gewinnt also, wenn σ^* eine gültige Signatur für m^* ist und den Challenger $\mathcal{C}$ nicht nach einer Signatur für m^* gefragt hat.

Remark 44. Trotz EUF-CMA-Sicherheit ist dies ein relativ schwaches Sicherheitsexperiment für Chamäleon-Signaturen, da der Angreifer nur Signaturen erhalten kann, die für einen Dritten, den ehrlichen Empfänger, bestimmt sind. Insbesondere darf er nicht selbst eine Chamäleon-Hashfunktion (möglicherweise „bösartig“) generieren, und eine Signatur vom Challenger bezüglich dieser Chamäleon-Hashfunktion erfragen. Dies könnte ihm jedoch dabei helfen eine Signatur zu fälschen, die auch von einem Dritten, dem ehrlichen Empfänger, akzeptiert wird. Wir betrachten dieses Sicherheitsmodell, welches auch in [KR00] benutzt wurde, der Einfachheit halber.

Theorem 45. *Für jeden PPT-Angreifer $\mathcal{A}(pk, \mathsf{ch})$, der die EUF-CMA-Sicherheit von Σ bricht in Zeit $t_{\mathcal{A}}$ mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{A}}$, existiert ein PPT-Angreifer $\mathcal{B}$, der in Zeit $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ läuft und*

- *entweder die Kollisionsresistenz von* ch *bricht mit einer Erfolgswahrscheinlichkeit von mindestens*

$$\epsilon_{\mathsf{ch}} \geq \frac{\epsilon_{\mathcal{A}}}{2},$$

- *oder die EUF-naCMA-Sicherheit von Σ' bricht mit einer Erfolgswahrscheinlichkeit von mindestens*

$$\epsilon' \geq \frac{\epsilon_{\mathcal{A}}}{2}.$$

Remark 46. Man beachte, dass das Theorem sogar die EUF-CMA-Sicherheit von Σ behauptet, selbst wenn Σ' nur EUF-naCMA-sicher ist.

Beweis. Jeder EUF-CMA-Angreifer $\mathcal{A}$ stellt eine Reihe von adaptiven Signatur-Anfragen $m_1, \ldots, m_q$, $q \geq 0$, auf welche er als Antwort Signaturen $\sigma_1, \ldots, \sigma_q$ erhält, wobei jede Signatur σ_i aus zwei Komponenten (σ'_i, r_i) besteht. Wir betrachten zwei verschiedene Ereignisse:

- Wir sagen, dass Ereignis E_0 eintritt, falls der Angreifer $(m^*, \sigma^*) = (m^*, (\sigma'^*, r^*))$ ausgibt, sodass
$$\mathsf{ch}(m^*, r^*) = \mathsf{ch}(m_i, r_i)$$
für mindestens ein $i \in \{1, \ldots, q\}$.
- Wir sagen, dass Ereignis E_1 eintritt, falls der Angreifer $(m^*, \sigma^*) = (m^*, (\sigma'^*, r^*))$ ausgibt, sodass
$$\mathsf{ch}(m^*, r^*) \neq \mathsf{ch}(m_i, r_i)$$
für alle $i \in \{1, \ldots, q\}$.

Jeder erfolgreiche Angreifer ruft entweder Ereignis E_0 oder Ereignis E_1 hervor. Es gilt also

$$\epsilon_{\mathcal{A}} \leq \Pr[E_0] + \Pr[E_1].$$

Die obige Ungleichung impliziert außerdem, dass zumindest eine der beiden Ungleichungen

$$\Pr[E_0] \geq \frac{\epsilon_{\mathcal{A}}}{2} \quad \text{oder} \quad \Pr[E_1] \geq \frac{\epsilon_{\mathcal{A}}}{2} \tag{3.1}$$

erfüllt sein muss.

Angriff auf die Chamäleon-Hashfunktion. Angreifer $\mathcal{B}$ versucht die Kollisionsresistenz der Chamäleon-Hashfunktion wie folgt zu brechen. $\mathcal{B}$ erhält als Eingabe ch und generiert ein Schlüsselpaar $(pk, sk) \xleftarrow{\$} \mathsf{Gen}'(1^k)$. Dann startet er $\mathcal{A}$ mit Eingabe pk. $\mathcal{B}$ kann das EUF-CMA-Experiment simulieren, da er den geheimen Schlüssel sk kennt und so alle Signatur-Anfragen von $\mathcal{A}$ beantworten kann.

Mit Wahrscheinlichkeit $\epsilon_{\mathcal{A}}$ wird $\mathcal{A}$ eine Fälschung $(m^*, \sigma^*) = (m^*, (\sigma'^*, r^*))$ produzieren. Falls dies eintritt, so prüft $\mathcal{B}$ ob Ereignis E_0 eintritt. $\mathcal{B}$ prüft also, ob es ein $i \in \{1, \ldots, q\}$ gibt, sodass $\mathsf{ch}(m^*, r^*) = \mathsf{ch}(m_i, r_i)$. Falls ja, so gibt $\mathcal{B}$ (m^*, r^*, m_i, r_i) aus.

Offensichtlich kann $\mathcal{B}$ die Fälschung von $\mathcal{A}$ benutzen, um die Kollisionsresistenz der Chamäleon-Hashfunktion zu brechen, wenn E_0 eintritt. $\mathcal{B}$ ist also erfolgreich mit einer Wahrscheinlichkeit von mindestens

$$\epsilon_{\mathsf{ch}} \geq \Pr[E_0]. \tag{3.2}$$

Angriff auf Σ'. $\mathcal{B}$ versucht die EUF-naCMA-Sicherheit von Σ' wie folgt zu brechen. Er generiert zunächst eine Chamäleon-Hashfunktion $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^k)$. Dann wählt $\mathcal{B}$ q zufällige Werte $(\tilde{m}_i, \tilde{r}_i) \xleftarrow{\$} \mathcal{M} \times \mathcal{R}$ und berechnet $y_i = \mathsf{ch}(\tilde{m}_i, \tilde{r}_i)$ für alle $i \in \{1, \ldots, q\}$.

Die Liste $y_1, \ldots, y_q$ gibt $\mathcal{B}$ als Nachrichten an seinen EUF-naCMA-Challenger aus. Als Antwort erhält er einen öffentlichen Schlüssel pk sowie Signaturen $\sigma'_1, \ldots, \sigma'_q$, sodass für jedes $i \in \{1, \ldots, q\}$ der Wert σ'_i eine gültige Signatur für y_i bezüglich pk ist. Nun startet $\mathcal{B}$ den Angreifer $\mathcal{A}$ mit Eingabe pk.

Die i-te Signaturanfrage m_i von $\mathcal{A}$ wird durch $\mathcal{B}$ so beantwortet:

1. Mit Hilfe der Trapdoor τ der Chamäleon-Hashfunktion berechnet $\mathcal{B}$ durch

$$r'_i = \mathsf{TrapColl}_{\mathsf{ch}}(\tau, \tilde{m}_i, \tilde{r}_i, m_i)$$

einen Wert r'_i sodass $y_i = \mathsf{ch}(m_i, r'_i)$ ist.

2. Dann beantwortet $\mathcal{B}$ die Anfrage von $\mathcal{A}$ mit Hilfe der vom Challenger erhaltenen Signatur σ'_i für y_i. $\mathcal{B}$ gibt also $\sigma_i = (\sigma'_i, r_i)$ an $\mathcal{A}$ zurück. Dies ist eine gültige Signatur für m_i.

Mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{A}}$ gibt $\mathcal{A}$ eine Fälschung $(m^*, \sigma^*) = (m^*, (\sigma'^*, r^*))$ aus. Wir schreiben $y^* := \mathsf{ch}(m^*, r^*)$. Falls Ereignis E_1 eintritt, so ist $y^* \neq y_i$ für alle $i \in \{1, \ldots, q\}$. Dann kann $\mathcal{B}$ das Tupel (y^*, σ'^*) als gültige Fälschung an seinen Challenger ausgeben. Wenn E_1 eintritt dann kann $\mathcal{B}$ gegen seinen Challenger gewinnen. Somit gilt

$$\epsilon' \geq \Pr[E_1]. \tag{3.3}$$

Insgesamt ergibt sich also, durch Einsetzen der Ungleichungen (3.2) und (3.3) in die Ungleichungen aus (3.1), dass zumindest eine der beiden Ungleichungen

$$\epsilon_{\mathsf{ch}} \geq \Pr[E_0] \geq \frac{\epsilon_{\mathcal{A}}}{2} \quad \text{oder} \quad \epsilon' \geq \Pr[E_1] \geq \frac{\epsilon_{\mathcal{A}}}{2}$$

gelten muss. □

Wir wissen also nun, dass die zusätzliche Chamäleon-Hashfunktion das Verfahren nicht unsicher macht. Im Gegenteil: Es wird sogar sicherer, da EUF-naCMA-Sicherheit von Σ' ausreicht, um EUF-CMA-Sicherheit von Σ zu zeigen.

Abstreitbarkeit. Es bleibt zu überlegen, ob ein Empfänger K, der eine Signatur von Sender H_1 erhält, einen Dritten H_2 davon überzeugen kann, dass H_1 eine ganz bestimmte Nachricht signiert hat.

Wenn H_1 das obige Chamäleon-Signaturverfahren verwendet, dann benutzt er zur Signaturerstellung die Chamäleon-Hashfunktion ch des Empfängers K. Für diese Chamäleon-Hashfunktion kann K mit Hilfe der Trapdoor beliebige Kollisionen finden – also einen gegebenen Hashwert zu *beliebigen Nachrichten* öffnen.

Jede dieser Nachrichten könnte, aus Sicht eines Dritten H_2, die Nachricht sein, die H_1 ursprünglich signiert hat – denn K als Empfänger der Signatur hat ja die Chamäleon-Hashfunktion, die von H_1 zur Signaturerstellung benutzt wurde, selbst generiert. Somit kennt K also auch die Trapdoor, und könnte damit beliebige Kollisionen finden – er kann also nicht glaubwürdig behaupten, dass H_1 eine *ganz bestimmte* Nachricht signiert hat. Ganz allein K weiß, welche Nachricht H_1 tatsächlich signiert hat.

Bösartig generierte Chamäleon-Hashfunktion. Es könnte natürlich sein, dass ein bösartiger Empfänger seine Chamäleon-Hashfunktion so generiert, dass er später plausibel erklären kann, dass er die Trapdoor *nicht* kennt. Bislang haben wir uns mit der Annahme ausgeholfen, dass der Empfänger die Chamäleon-Hashfunktion ehrlich durch $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^k)$ generiert hat.

Wir können den Empfänger jedoch *„zwingen“*, dass er die Trapdoor τ stets kennen muss. Dies geht zum Beispiel, indem der Empfänger zusammen mit der Beschreibung ch seiner

Chamäleon-Hashfunktion einen „Beweis“ veröffentlicht, dass er die dazu gehörige Trapdoor kennt. Dies geht mit kryptographischen Methoden, wie zum Beispiel (nicht-interaktiven) *Zero-Knowledge Proofs of Knowledge*. Wir erklären an dieser Stelle nicht, was das genau ist – es sei nur gesagt, dass das Problem eine Lösung hat.

3.5 Chamäleon-Hashfunktionen sind Einmalsignaturverfahren

Wir haben bereits gesehen, dass einige Konstruktionen von Chamäleon-Hashfunktionen sehr ähnlich sind zu den uns schon bekannten Einmalsignaturverfahren. Dies gibt Anlass zur Vermutung, dass beide Primitive miteinander verwandt sind.

Tatsächlich ist es so, dass Chamäleon-Hashfunktionen Einmalsignaturverfahren implizieren. Das bedeutet, aus jeder Chamäleon-Hashfunktion kann man, auf generische Art und Weise, ein Einmalsignaturverfahren bauen. Das Gegenteil stimmt jedoch nicht unbedingt. Nicht aus jedem Einmalsignaturverfahren muss man eine Chamäleon-Hashfunktion konstruieren können. Die Konstruktion wurde 2010 in [Moh10] publiziert.

Sei $(\mathsf{Gen}_{\mathsf{ch}}, \mathsf{TrapColl}_{\mathsf{ch}})$ eine Chamäleon-Hashfunktion. Wir konstruieren ein Einmalsignaturverfahren $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ wie folgt.

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus generiert eine Chamäleon-Hashfunktion

$$\mathsf{ch} : \mathcal{M} \times \mathcal{R} \to \mathcal{N}$$

mit Trapdoor τ, indem er $(\mathsf{ch}, \tau) \xleftarrow{\$} \mathsf{Gen}_{\mathsf{ch}}(1^k)$ ausführt. Dann wählt er $(\tilde{m}, \tilde{r}) \xleftarrow{\$} \mathcal{M} \times \mathcal{R}$, und berechnet $c := \mathsf{ch}(\tilde{m}, \tilde{r})$. Der öffentliche Schlüssel ist $pk := (\mathsf{ch}, c)$, der geheime Schlüssel ist $sk := (\tau, \tilde{m}, \tilde{r})$.

$\mathsf{Sign}(sk, m)$**.** Um eine Nachricht $m \in \mathcal{M}$ zu signieren, wird eine passende Randomness r berechnet, sodass $\mathsf{ch}(m, r) = c$. Dies geht mit Hilfe von $sk = (\tau, \tilde{m}, \tilde{r})$ durch Berechnung von

$$r := \mathsf{TrapColl}_{\mathsf{ch}}(\tau, \tilde{m}, \tilde{r}, m).$$

Die Signatur ist $\sigma := r$.

$\mathsf{Vfy}(pk, m, \sigma)$**.** Gegeben eine Signatur $\sigma = r \in \mathcal{R}$, prüft der Verifikationsalgorithmus, ob

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

gilt. Falls ja, so wird 1 ausgegeben, ansonsten 0.

Theorem 47. *Für jeden PPT-Angreifer $\mathcal{A}$, der die EUF-1-naCMA-Sicherheit von Σ in Zeit $t_{\mathcal{A}}$ mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{A}}$ bricht, existiert ein PPT-Angreifer $\mathcal{B}$, der die Kollisionsresistenz der Chamäleon-Hashfunktion bricht in Zeit $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{B}} \geq \epsilon_{\mathcal{A}}$.*

Excercise 48. Beweisen Sie Theorem 47. Tipp: Nutzen Sie aus, dass Sie im EUF-1-naCMA-Experiment den pk erst generieren müssen, *nachdem* Sie vom Angreifer die gewählte Nachricht m erhalten haben, und dass der Angreifer eine Lösung (m^*, σ^*) der Gleichung

$$c = \mathsf{ch}(m^*, \sigma^*)$$

berechnen muss.

Excercise 49. Konstruieren Sie ein EUF-1-CMA-sicheres Einmalsignaturverfahren aus Chamäleon-Hashfunktionen. *Tipp:* Hier können Sie die Transformation aus Kapitel 2.4 anwenden.

Remark 50. Die auf dem diskreten Logarithmusproblem (DLP) und dem RSA-Problem basierenden Einmalsignaturverfahren, die wir in Kapitel 2 beschrieben haben, sind genau die Einmalsignaturverfahren, die sich ergeben, wenn man die obige Transformation auf die auf dem DLP- und RSA-basierten Chamäleon-Hashfunktionen anwendet.

3.6 Strong Existential Unforgeability durch Chamäleon-Hashing

Die stärkste bislang vorgestellte Sicherheitsdefinition für digitale Signaturen ist EUF-CMA. Für manche Anwendungen benötigt man jedoch noch stärkere Signaturverfahren.

Bei EUF-CMA-Sicherheit (beziehungsweise eigentlich generell bei EUF-Sicherheit, kombiniert mit beliebigen Angreifermodellen wie NMA,naCMA,CMA,...) fordern wir, dass kein effizienter Angreifer in der Lage ist eine gültige Signatur für eine *neue* Nachricht $m^* \notin \{m_1, \ldots, m_q\}$ zu erzeugen. Dies modelliert das Ziel, dass kein Angreifer in der Lage ist Signaturen für neue Nachrichten zu erzeugen, die nicht vom ehrlichen Signierer signiert wurden.

Einen Angriff, den diese Sicherheitsdefinition nicht abdeckt, ist, dass der Angreifer es schaffen könnte für eine Nachricht $m^* \in (m_1, \ldots, m_q)$ eine *neue* Signatur zu erstellen, die nicht vom ehrlichen Signierer erzeugt wurde.

Dieser Angriff klingt zunächst sinnlos, denn der Angreifer kennt ja schon eine Signatur für m^*, warum sollte er eine Neue erstellen wollen? Tatsächlich gibt es in der Kryptographie aber Anwendungen, für die man solche Angriffe ausschließen muss. Das vielleicht wichtigste Beispiel sind manche Konstruktionen von starken *public key*-Verschlüsselungsverfahren (IND-CCA sicher), die als Baustein ein (Einmal-) Signaturverfahren, oder alternativ eine Chamäleon-Hashfunktion, benutzen. Diese Konstruktionen sind nicht trivial, daher können wir hier leider nicht auf die Details eingehen –Wichtig ist nur zu sagen, dass EUF-CMA-Sicherheit manchmal nicht ausreicht, und man stärkere Sicherheitseigenschaften braucht.

Ein Verfahren, bei dem man nicht mal dann eine neue Signatur generieren kann, selbst wenn man schon eine andere Signatur für die vom Angreifer gewählte Nachricht m^* kennt, nennt man *strong existential unforgeable*.

Das sEUF-CMA Sicherheitsexperiment. Das sEUF-CMA Sicherheitsexperiment mit Angreifer $\mathcal{A}$, Challenger $\mathcal{C}$ und Signaturverfahren $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ läuft genauso ab wie das EUF-CMA Sicherheitsexperiment (siehe auch Abbildung 3.2):

1. Der Challenger $\mathcal{C}$ generiert ein Schlüsselpaar $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^k)$. Der Angreifer erhält pk.

2. Nun darf der Angreifer $\mathcal{A}$ beliebige Nachrichten $m_1, \ldots, m_q$ vom Challenger signieren lassen.

   Dazu sendet er Nachricht m_i an den Challenger. Dieser berechnet $\sigma_i = \mathsf{Sign}(sk, m_i)$ und antwortet mit σ_i.

   Dieser Schritt kann vom Angreifer beliebig oft wiederholt werden. Wenn wir Angreifer mit polynomiell beschränkter Laufzeit betrachten werden, ist $q = q(k)$ üblicherweise ein Polynom im Sicherheitsparameter.

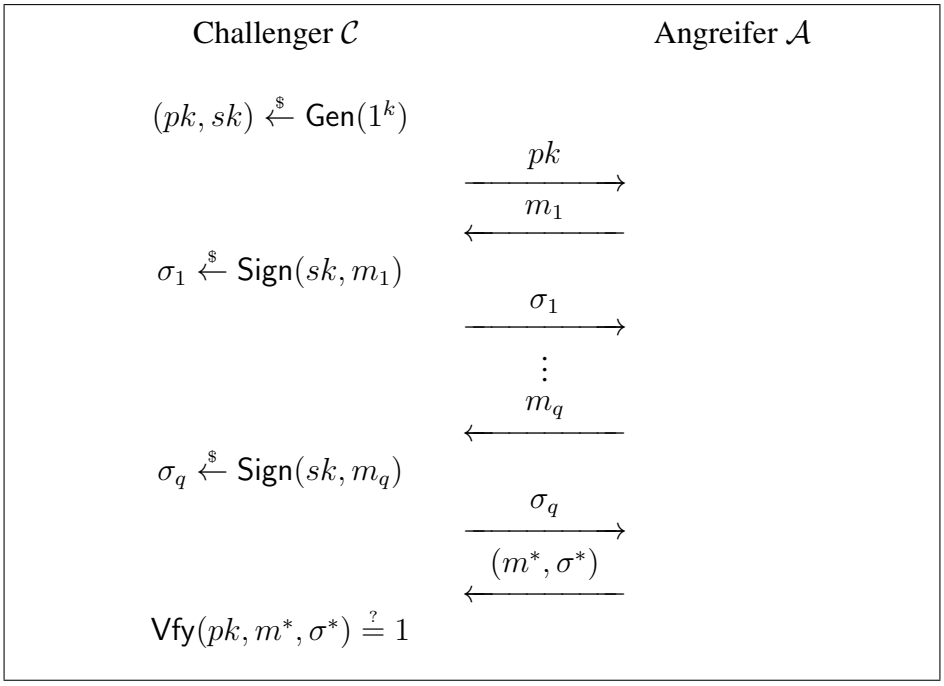

Abbildung 3.2: Das sEUF-CMA Sicherheitsexperiment ist identisch zum EUF-CMA-Experiment.

3. Am Ende gibt $\mathcal{A}$ eine Nachricht m^* mit Signatur σ^* aus.

Definition 51. Wir sagen, dass $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ sicher ist im Sinne von sEUF-CMA, falls für alle PPT-Angreifer $\mathcal{A}$ im sEUF-CMA-Experiment gilt, dass

$$\Pr[\mathcal{A}^{\mathcal{C}}(pk) = (m^*, \sigma^*) : \mathsf{Vfy}(pk, m^*, \sigma^*) = 1 \wedge (m^*, \sigma^*) \notin \{(m_1, \sigma_1), \ldots, (m_q, \sigma_q)\}] \leq \mathsf{negl}(k)$$

für eine vernachlässigbare Funktion negl im Sicherheitsparameter.

Der Unterschied zwischen den Definitionen von EUF-CMA-Sicherheit und sEUF-CMA-Sicherheit ist subtil. Während wir bei EUF-CMA-Sicherheit lediglich fordern, dass

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

ist, gibt es bei sEUF-CMA die stärkere Forderung dass

$$(m^*, \sigma^*) \notin \{(m_1, \sigma_1), \ldots, (m_q, \sigma_q)\}.$$

Die obige Definition von sEUF-CMA-Sicherheit lässt sich auf natürliche Weise adaptieren zu sEUF-naCMA, sEUF-1-CMA und sEUF-1-naCMA.

Excercise 52. Geben Sie die Definitionen von sEUF-1-naCMA- und sEUF-1-CMA-Sicherheit in Form von Sicherheitsexperimenten an.

Excercise 53. Zeigen Sie, dass Einmalsignaturverfahren, die durch Anwendung der Transformation aus Kapitel 3.5 aus Chamäleon-Hashfunktionen erstellt wurden, sEUF-1-naCMA-sicher sind.

Excercise 54. Konstruieren Sie ein sEUF-1-CMA-sicheres Einmalsignaturverfahren aus Chamäleon-Hashfunktionen. *Tipp:* Wenn Sie zur Lösung von Übungsaufgabe 49 die Transformation aus Kapitel 2.4 verwendet haben, dann können Sie zeigen, dass das so konstruierte Verfahren bereits sEUF-1-CMA-sicher ist.

Strong Existential Unforgeability durch Chamäleon-Hashing. Interessanterweise ist es sehr leicht möglich, ein EUF-naCMA-sicheres Signaturverfahren Σ' in ein sEUF-CMA-sicheres Signaturverfahren zu transformieren. Dies geht durch geschickte Anwendung eines sEUF-1-CMA-sicheren Einmalsignaturverfahrens. Ein solches Verfahren wurde in Übungsaufgabe 54 basierend auf Chamäleon-Hashfunktionen konstruiert.

Die Transformation. Im Folgenden bezeichne $\Sigma^{(1)} = (\mathsf{Gen}^{(1)}, \mathsf{Sign}^{(1)}, \mathsf{Vfy}^{(1)})$ ein sEUF-1-CMA-sicheres Einmalsignaturverfahren und $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vfy}')$ ein EUF-naCMA-sicheres Signaturverfahren. Wir beschreiben ein neues Signaturverfahren $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$, welches $\Sigma^{(1)}$ und Σ' als Bausteine benutzt. Die Konstruktion erinnert stark an die Transformation von EUF-naCMA-sicheren Signaturverfahren zu EUF-CMA-sicheren Signaturen aus Kapitel 2.

$\mathsf{Gen}(1^k)$**.** Zur Schlüsselerzeugung wird ein Schlüsselpaar $(pk', sk') \xleftarrow{\$} \mathsf{Gen}'(1^k)$ mit dem Schlüsselerzeugungsalgorithmus von Σ' generiert.

Der öffentliche Schlüssel ist $pk = pk'$, der geheime Schlüssel ist $sk = sk'$.

$\mathsf{Sign}(sk, m)$**.** Eine Signatur für Nachricht m wird in zwei Schritten berechnet.

1. Zunächst wird ein Schlüsselpaar $(pk^{(1)}, sk^{(1)}) \xleftarrow{\$} \mathsf{Gen}^{(1)}(1^k)$ für das Einmalsignaturverfahren generiert.
2. Der Wert $pk^{(1)}$ wird dann mit dem Signaturverfahren Σ' signiert:
$$\sigma' := \mathsf{Sign}'(sk', pk^{(1)}).$$
3. Dann wird mit Hilfe von $sk^{(1)}$ eine Signatur $\sigma^{(1)} = \mathsf{Sign}^{(1)}(sk^{(1)}, (m, \sigma'))$ über (m, σ') berechnet.

Die Signatur σ für Nachricht m ist $\sigma := (\sigma', pk^{(1)}, \sigma^{(1)})$.

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus erhält $\sigma := (\sigma', pk^{(1)}, \sigma^{(1)})$ und m, und gibt 1 aus wenn
$$\mathsf{Vfy}'(pk, pk^{(1)}, \sigma') = 1 \qquad \text{und} \qquad \mathsf{Vfy}^{(1)}(pk^{(1)}, (m, \sigma'), \sigma^{(1)}) = 1.$$
Ansonsten wird 0 ausgegeben.

Die Idee der Transformation ist also, zusätzlich zur Nachricht auch die Signatur σ' mit zu signieren.

Theorem 55. *Für jeden PPT-Angreifer $\mathcal{A}$, der die sEUF-CMA-Sicherheit von Σ bricht in Zeit $t_{\mathcal{A}}$ mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{A}}$, existiert ein PPT-Angreifer $\mathcal{B}$, der in Zeit $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ läuft und*

- *entweder die Kollisionsresistenz von* ch *bricht mit einer Erfolgswahrscheinlichkeit von mindestens*
$$\epsilon_{\mathsf{ch}} \geq \frac{\epsilon_{\mathcal{A}}}{2},$$
- *oder die EUF-naCMA-Sicherheit von Σ' bricht mit einer Erfolgswahrscheinlichkeit von mindestens*
$$\epsilon' \geq \frac{\epsilon_{\mathcal{A}}}{2}.$$

Der Beweis dieses Theorems ist sehr ähnlich zu den Beweisen von Theorem 32 und Theorem 45. Daher geben wir ihn hier nicht an, sondern überlassen ihn als (etwas aufwendigere) Übungsaufgabe.

Excercise 56. Beweisen Sie Theorem 55.

Die Konstruktion, die wir in diesem Kapitel vorgestellt haben, stammt im Grunde von Steinfeld, Pieprzyk und Wang [SPW07], wir haben sie jedoch etwas anders beschrieben. Während wir aus der Chamäleon-Hashfunktion zunächst ein sEUF-1-CMA-sicheres Signaturverfahren konstruiert, und dann dieses Verfahren für die generische Transformation benutzt haben, wurden in [SPW07] direkt Chamäleon-Hashfunktionen verwendet. Eine etwas schwächere generische Transformation, die nur für manche Signaturverfahren (so genannte „separierbare Signaturen“) funktioniert, wurde in [BSW06] vorgestellt.

Kapitel 4

RSA-basierte Signaturverfahren

Bislang haben wir in dieser Vorlesung Einmalsignaturverfahren und q-mal Signaturverfahren basierend auf Merkle-Bäumen kennengelernt, wobei q polynomiell beschränkt war. In diesem Kapitel beginnen wir mit der Betrachtung von Signaturverfahren, die eine exponentielle Anzahl von Nachrichten signieren können. Eine wichtige Klasse solcher Verfahren basiert auf Varianten der RSA-Annahme.

Wir betrachten in diesem Kapitel zunächst das „Lehrbuch"-RSA Signaturverfahren, welches einige Nachteile (wie zum Beispiel Homomorphie, die im Kontext von Signaturverfahren nicht unbedingt vorteilhaft sein muss) mit sich bringt. Danach beschreiben und analysieren wir die folgenden RSA-basierten Signaturverfahren:

- Das RSA Full-Domain Hash Verfahren [BR96]. Die Sicherheitsanalyse dieses Verfahrens erfolgt in einem idealisierten Modell, dem so genannten Random Oracle Modell.
- Das Signaturverfahren von Gennaro, Halevi und Rabin [GHR99]. Dieses ist das erste „Hash-and-Sign" Signaturvefahren mit einem EUF-CMA-Sicherheitsbeweis, es basiert jedoch auf einer stärkeren Komplexitätsannahme, der so genannten *Strong-RSA-Annahme*.
- Das RSA-basierte Signaturverfahren von Hohenberger und Waters [HW09b]. Dieses Verfahren ist das Erste, welches beweisbar EUF-CMA-sicher ist unter der RSA-Annahme. Es ist jedoch leider relativ ineffizient. Die Konstruktion eines effizienten RSA-basierten Signaturverfahrens mit Beweis im Standardmodell (also ohne Random Oracles und ähnliche Idealisierungen) ist ein wichtiges offenes Problem.

4.1 „Lehrbuch"-RSA Signaturen

Sei $N \in \mathbb{N}$ eine natürliche Zahl. Wir verwenden die in Kapitel 2.3.2 eingeführte Notation und Definitionen von $\mathbb{Z}_N$, $\mathbb{Z}_N^*$ und $\phi(N)$ sowie die Formulierung des RSA-Problems aus Definition 29.

Das „Lehrbuch"-Signaturverfahren funktioniert so:

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus erzeugt einen RSA-Modulus $N = PQ$, wobei P und Q zwei zufällig gewählte Primzahlen sind. Dann wird eine Zahl $e \in \mathbb{N}$ mit $e \neq 1$ und $\mathrm{ggT}(e, \phi(N)) = 1$ gewählt, und der Wert $d := e^{-1} \bmod \phi(N)$ berechnet.

Der öffentliche Schlüssel ist $pk := (N, e)$, der geheime Schlüssel ist $sk := d$.

$\mathsf{Sign}(sk, m)$**.** Um eine Nachricht $m \in \mathbb{Z}_N$ zu signieren wird $\sigma \in \mathbb{Z}_N$ berechnet als

$$\sigma \equiv m^d \bmod N.$$

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus gibt 1 aus, wenn

$$m \equiv \sigma^e \bmod N$$

gilt, und ansonsten 0.

Correctness. Die Correctness des Verfahrens ergibt sich daraus, dass $d \equiv e^{-1} \bmod \phi(N)$ ist und daher

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

gilt.

Sicherheit von „Lehrbuch"-RSA Signaturen. Das „Lehrbuch"-RSA Signaturverfahren ist sehr einfach und eignet sich daher gut um das Prinzip digitaler Signaturen in Lehrbüchern zu erklären. Es ist für viele praktische Anwendungen jedoch zu schwach.

EUF-Sicherheit. Es ist recht leicht zu sehen, dass das „Lehrbuch"-RSA Signaturverfahren nicht EUF-NMA-sicher ist, und somit auch nicht EUF-CMA-sicher.

Ein Angreifer kann einfach eine beliebige Signatur $\sigma^* \in \mathbb{Z}_N$ wählen, und sich die passende Nachricht m^* dazu berechnen als $m^* \equiv (\sigma^*)^e \bmod N$. Offensichtlich ist (m^*, σ^*) eine gültige existentielle Fälschung. Der Angreifer muss dafür noch nicht einmal eine *chosen-message* Anfrage an den Challenger stellen.

UUF-CMA-Sicherheit. „Lehrbuch"-RSA Signaturen sind auch nicht UUF-CMA-sicher.

Ein Angreifer erhält als Eingabe vom Challenger eine Nachricht m^*. Sein Ziel ist die Berechnung einer Signatur σ^* mit $(\sigma^*)^e \equiv m^* \bmod N$. Er geht dazu so vor:

1. Der Angreifer wählt einen zufälligen Wert $x \xleftarrow{\$} \mathbb{Z}_N^* \setminus \{1\}$ und berechnet $y \equiv x^e \bmod N$.
2. Dann berechnet der Angreifer $m_1 := m^* \cdot y \bmod N$, und fragt den Challenger nach einer Signatur für m_1. Weil $x \neq 1 \bmod N$ gewählt wurde, ist auch $y \neq 1 \bmod N$. Daher ist $m_1 \neq m^*$. Als Antwort erhält der Angreifer daher einen Wert σ_1 mit $\sigma_1^e \equiv m_1 \bmod N$.
3. Zum Schluss berechnet der Angreifer $\sigma^* \equiv \sigma_1 \cdot x^{-1} \bmod N$, und gibt σ^* aus. Dies ist möglich, weil $x \in \mathbb{Z}_N^*$ invertierbar ist. Ausserdem ist dies eine gültige Signatur für m^*, denn

$$(\sigma^*)^e \equiv (\sigma_1 \cdot x^{-1})^e \equiv \sigma_1^e \cdot (x^e)^{-1} \equiv m_1 \cdot y^{-1} \equiv m^* \cdot y \cdot y^{-1} \equiv m^* \bmod N.$$

Die Tatsache, dass „Lehrbuch"-RSA Signaturen multiplikativ homomorph sind, erlaubt also diesen UUF-CMA-Angriff.

UUF-NMA-Sicherheit. Man kann durch eine einfache Reduktion zeigen, dass „Lehrbuch"-RSA Signaturen UUF-NMA-sicher sind unter der Annahme, dass das RSA-Problem „schwer" ist. Dies ist jedoch ein sehr schwaches Sicherheitsziel, und für praktische Anwendungen in der Regel nicht ausreichend.

Excercise 57. Zeigen Sie, dass das „Lehrbuch"-RSA Signaturverfahren UUF-NMA-sicher ist, unter der Annahme, dass das RSA-Problem „schwer" ist.

4.2 RSA Full-Domain Hash und das Random Oracle Modell

In diesem Kapitel beschreiben wir das RSA-basierte *Full-Domain Hash* Signaturverfahren (RSA-FDH), ein sehr wichtiges Verfahren. Es ist eine Erweiterung des „Lehrbuch"-RSA Verfahrens, bei der auf die zu signierende Nachricht zunächst eine Hashfunktion angewendet wird, und dann der Hash der Nachricht signiert wird. Wenn man eine geeignete (jedoch sehr starke) Anforderung an die Sicherheit der Hashfunktion stellt, dann kann man zeigen, dass dies die Schwachpunkte des „Lehrbuch"-RSA Verfahrens beseitigt.

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus erzeugt einen RSA-Modulus $N = PQ$, wobei P und Q zwei zufällig gewählte Primzahlen sind. Weiterhin wird eine Zahl $e \in \mathbb{N}$ mit $e \neq 1$ und $\mathrm{ggT}(e, \phi(N)) = 1$ gewählt, und der Wert $d := e^{-1} \bmod \phi(N)$ berechnet. Zum Schluss wird eine Hashfunktion

$$H : \{0,1\}^* \to \mathbb{Z}_N$$

ausgewählt. Der öffentliche Schlüssel ist $pk := (N, e, H)$, der geheime Schlüssel ist $sk := d$.

$\mathsf{Sign}(sk, m)$**.** Um eine Nachricht $m \in \{0,1\}^*$ zu signieren wird $\sigma \in \mathbb{Z}_N$ berechnet als

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

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus gibt 1 aus, wenn

$$H(m) \equiv \sigma^e \bmod N$$

gilt, und ansonsten 0.

Correctness. Die *Correctness* des Verfahrens ergibt sich wieder durch Einsetzen:

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

da $d \cdot e \equiv 1 \bmod \phi(N)$ ist.

Soundness. Um die EUF-CMA-Sicherheit des RSA-FDH Verfahrens beweisen zu können, müssen wir neben der RSA-Annahme noch eine geeignete Annahme über die Sicherheit der Hashfunktion H treffen. Offensichtlich müssen wir zumindest mal annehmen, dass H kollisionsresistent ist, ansonsten könnte ein Angreifer zunächst eine Kollision m, m^* mit $H(m) = H(m^*)$ für H finden und diese dann benutzen um das EUF-CMA-Experiment zu gewinnen.

Weiterhin darf H natürlich nicht die Identitätsabbildung sein (welche ja offensichtlich kollisionsresistent ist), denn ansonsten wäre das resultierende Full-Domain-Hash Verfahren ja identisch zum „Lehrbuch"-RSA-Verfahren, und wäre damit aufgrund der Homomorphie nicht mehr beweisbar sicher. Die Hashfunktion muss also diese Homomorphie irgendwie zerstören, was jedoch eine Sicherheitseigenschaft ist, die nicht so leicht klar definierbar ist.

Wir treffen im Folgenden eine sehr starke Annahme.

4.2.1 Das Random Oracle Modell

Sei $H : \mathcal{D} \to \mathcal{R}$ eine Hashfunktion.[1] Im Random Oracle Modell [BR93] wird angenommen, dass diese Hashfunktion *ideal* ist. Das bedeutet intuitiv:

- Ein Angreifer kann mit der Hashfunktion *nichts anderes tun als sie auszuwerten*, insbesondere kann er aus einem gegebenen Code der Hashfunktion keinen Nutzen ziehen, der ihm bei einem Angriff hilft, und
- die Hashfunktion gibt für jede Eingabe $m \in \mathcal{D}$ ein eindeutiges, aber *gleichverteilt zufälliges* Element $H(m) \in \mathcal{R}$ zurück.

Um diese Intuition zu formalisieren, modellieren wir die Funktion H als ein „Orakel", das *Random Oracle*, welches wie folgt funktioniert.

Aus Sicht des Angreifers ist das Random Oracle eine *Black-Box*. Um die Hashfunktion auszuwerten, also zu einem Wert $m \in \mathcal{D}$ den Hashwert $H(m) \in \mathcal{R}$ zu berechnen, stellt der Angreifer eine Anfrage an das Random Oracle. Die Anfrage besteht aus einem Wert $m \in \mathcal{D}$. Jede Anfrage wird mit einem gleichverteilt zufälligen Element $y = H(m) \in \mathcal{R}$ beantwortet. Die Antworten des Random Oracles sind jedoch *konsistent*: Wenn der Angreifer mehrmals den gleichen Wert m anfragt, dann erhält er jedes Mal den gleichen Hashwert $y = H(m)$ als Antwort.

Um ein konkretes Beispiel zu geben kann man sich hier vorstellen, dass das Random Oracle intern eine Liste

$$\mathcal{L} \subseteq \mathcal{D} \times \mathcal{R}$$

führt. Jeder Eintrag $(m, y) \in \mathcal{L}$ ordnet dem Wert $m \in \mathcal{D}$ seinen Hashwert $y \in \mathcal{R}$ zu.

- Vor der ersten Anfrage des Angreifers ist die Liste leer.
- Jedes Mal wenn der Angreifer einen Wert m anfragt, dann prüft das Random Oracle, ob es einen Eintrag $(m, y) \in \mathcal{L}$ gibt, dessen erste Koordinate dem angefragten Wert m entspricht.
  - Falls ja, so gibt das Random Oracle den Wert y zurück.
  - Falls nicht, so

[1] Im Falle des RSA-FDH Verfahrens ist $\mathcal{D} = \{0,1\}^*$ und $\mathcal{R} = \mathbb{Z}_N$.

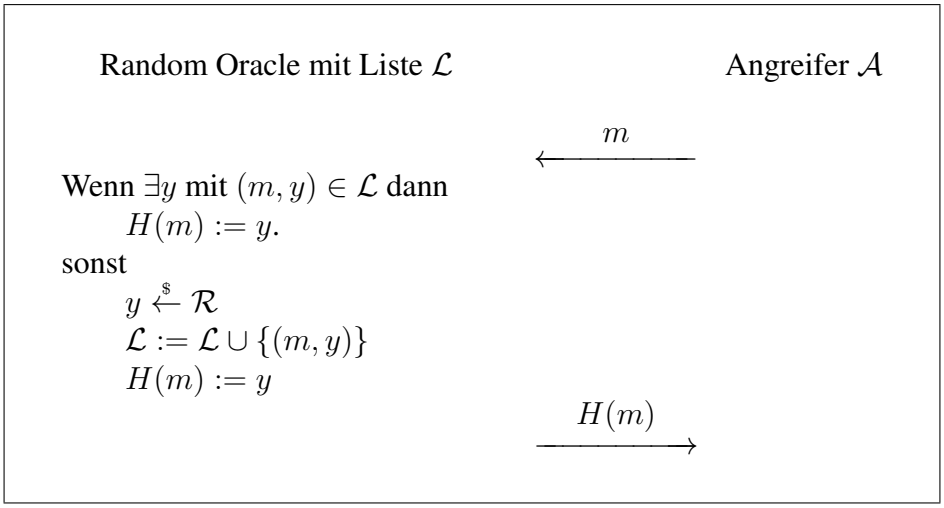

Abbildung 4.1: Interaktion zwischen Angreifer und Random Oracle. Der Angreifer darf beliebig viele Anfragen m an das Random Oracle stellen. Zu Beginn, d.h. bevor der Angreifer die erste Anfrage stellt, ist die Liste $\mathcal{L}$ leer.

- wählt das Random Oracle einen neuen Wert $y \xleftarrow{\$} \mathcal{R}$,
- trägt den Wert (m, y) in die Liste $\mathcal{L}$ ein,
- und gibt dann den Wert y zurück.

Remark 58. Das Random Oracle Modell ist eine *Idealisierung* einer konkreten Hashfunktion, die manchmal angenommen wird, um Sicherheitsbeweise durchzuführen. In der Praxis würde ein Kryptosystem stets mit einer konkreten Hashfunktion, wie zum Beispiel SHA-256, instantiiert werden. Die Idealisierung wird *lediglich für den Sicherheitsbeweis* angenommen.

Remark 59. Wenn ein Verfahren mit Sicherheitsbeweis im Random Oracle Modell in der Praxis eingesetzt wird, und das Random Oracle dabei mit einer konkreten Hashfunktion instantiiert wird, dann muss man sich bewusst sein, dass der Sicherheitsbeweis dann nicht mehr für das Verfahren gilt – denn es könnte sein, dass die konkrete Hashfunktion das Random Oracle nicht hinreichend gut „approximiert“. Die Heuristik, dass eine Hashfunktion wie SHA-256 „aus Sicht des Angreifers genauso gut wie ein Random Oracle“ ist, nennt man die *Random Oracle Heuristik*.

Remark 60. Es ist bekannt [CGH98], dass man *künstliche* („unrealistische“) Kryptosysteme konstruieren kann, die im Random Oracle Modell beweisbar sicher sind, die aber völlig unsicher sind, sobald das Random Oracle *mit einer beliebigen Hashfunktion* instantiiert wird. Die weckt natürlich starke Zweifel an der Random Oracle Heuristik. Allerdings sind diese Verfahren sehr künstlich und gezielt so konstruiert, dass sie nur genau dann sicher sind, wenn die Hashfunktion ein Random Oracle ist. Es ist kein einziges „realistisches“ Verfahren mit einer ähnlichen Eigenschaft bekannt – eine Konstruktion eines solchen Verfahrens (falls eines existiert) wäre sehr interessant, und ist ein offenes Forschungsproblem.

Remark 61. Das Random Oracle Modell kann als Hilfsmittel angesehen werden um Sicherheitsbeweise für Kryptosysteme anzugeben, für die wir sonst keine formalen Sicherheitsargumente finden können. Es ist im Allgemeinen wesentlich leichter einen Sicherheitsbeweis im Random Oracle Modell zu finden, als einen Beweis im Standardmodell (also ohne die Random Oracle Heuristik oder ähnliche Idealisierungen).

Zumeist sind kryptographische Verfahren mit Sicherheitsbeweis im Random Oracle Modell wesentlich effizienter als Verfahren, die einen Standardmodell-Beweis haben. Die Konstruktion solcher praktischer Verfahren ist auch genau die Idee, die Bellare und Rogaway bei der „Erfindung“ des Random Oracle Modells im Sinn hatten – wie schon der Titel ihres Artikels [BR93], in dem das Random Oracle Modell eingeführt wurde, verrät: *„Random oracles are practical: a paradigm for designing efficient protocols“*.

Der Gedanke dahinter ist: Einen Sicherheitsbeweis für ein Kryptosystem im Random Oracle Modell zu haben ist immerhin schonmal besser als überhaupt keinen Sicherheitsbeweis.

Remark 62. Das Random Oracle Modell stellt oftmals auch eine Hilfe dar zur Konstruktion neuer kryptographischer Bausteine. So hatten zum Beispiel die ersten identitätsbasierten Verschlüsselungsverfahren [BF01] oder die ersten kurzen Signaturen über bilinearen Gruppen [BLS01, BLS04] (die wir in einem späteren Kapitel betrachten werden) nur Beweise im Random Oracle Modell.

Eine erste Konstruktion eines neuen Bausteins anzugeben, auch wenn der Sicherheitsbeweis zunächst nur im Random Oracle Modell ist, kann dabei helfen zu lernen, wie man den gesuchten neuen Baustein *prinzipiell* konstruieren könnte. Ein zweiter Schritt ist dann die Suche nach einer ausgefeilteren Konstruktion, möglicherweise einer Variante der Ersten, die einen Sicherheitsbeweis im Standardmodell hat.

Remark 63. Es gibt auch kryptographische Bausteine, von denen man weiß, dass sie *nur* im Random Oracle Modell beweisbar existieren. Ein Beispiel dafür sind Verschlüsselungsverfahren, die sicher gegen so genannte adaptive Korruption von Empfängern sind *(„non-commiting encryption“)* [Nie02].

4.2.2 Sicherheitsbeweis von RSA-FDH im Random Oracle Modell

Theorem 64. *Sei* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ *das RSA-FDH Signaturverfahren mit Hashfunktion* H*. Wenn* H *als Random Oracle modelliert wird, dann existiert für jeden PPT-Angreifer* $\mathcal{A}$*, der die* EUF-CMA*-Sicherheit von* Σ *in Zeit* $t_{\mathcal{A}}$ *mit Erfolgswahrscheinlichkeit* $\epsilon_{\mathcal{A}}$ *bricht, und dabei* q_H *Anfragen an das Random Oracle stellt, ein PPT-Angreifer* $\mathcal{B}$*, der das RSA-Problem löst in Zeit* $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ *mit Erfolgswahrscheinlichkeit mindestens*

$$\epsilon_B \geq \frac{\epsilon_{\mathcal{A}} - 1/N}{q_H}.$$

Beweis. Die Grundlage des Sicherheitsbeweises ist die Beobachtung, dass der Angreifer das Random Oracle fragen muss, um einen Hashwert zu berechnen. Das Random Oracle wird vom Challenger implementiert. Die Beweisidee ist, eine gegebene RSA-Challenge auf geschickte Art und Weise so in die Antworten des Random Oracles einzubetten, dass man den Signatur-Angreifer benutzen kann, um das RSA-Problem zu lösen.

Ein erfolgreicher Angreifer erhält als Eingabe einen *public key* (N, e), darf Signatur-Anfragen stellen, und gibt am Ende (m^*, σ^*) aus, sodass $\sigma^* = H(m^*)^{1/e}$. Wir betrachten wieder zwei Ereignisse, die der Angreifer im Verlaufe des Experimentes hervorrufen kann:

- Wir sagen, dass Ereignis E_0 eintritt, falls der Angreifer $\mathcal{A}$ erfolgreich eine gültige Signatur $(m^*, \sigma^*) = (m^*, H(m^*)^{1/e})$ ausgibt und $\mathcal{A}$ im Verlaufe des Experimentes *niemals* das Random Oracle nach dem Hashwert $H(m^*)$ gefragt hat.

- Wir sagen, dass Ereignis E_1 eintritt, falls der Angreifer $\mathcal{A}$ $(m^*, \sigma^*) = (m^*, H(m^*)^{1/e})$ ausgibt und $\mathcal{A}$ im Verlaufe des Experimentes *irgendwann einmal* das Random Oracle nach dem Hashwert $H(m^*)$ gefragt hat.

Jeder erfolgreiche Angreifer ruft entweder Ereignis E_0 oder Ereignis E_1 hervor, also gilt

$$\epsilon_{\mathcal{A}} \leq \Pr[E_0] + \Pr[E_1].$$

Diese Ungleichung lässt sich umformen zu

$$\Pr[E_1] \geq \epsilon_{\mathcal{A}} - \Pr[E_0]. \tag{4.1}$$

Angreifer, die niemals $H(m^*)$ beim Random Oracle anfragen. Es ist recht leicht zu sehen, dass ein solcher Angreifer nur eine sehr kleine Erfolgswahrscheinlichkeit hat. Das Random Oracle wählt jeden Hashwert gleichverteilt zufällig aus $\mathbb{Z}_N$. Insbesondere ist also der Hashwert $H(m^*)$ gleichverteilt zufällig über $\mathbb{Z}_N$, und der Angreifer erhält keine Information über $H(m^*)$.

Da die Abbildung $h \mapsto h^{1/e} \bmod N$ eine Bijektion über $\mathbb{Z}_N$ ist, ist also auch der Wert $H(m^*)^{1/e}$ gleichverteilt über $\mathbb{Z}_N$. Der Angreifer kann diesen Wert nur raten, und somit ist seine Erfolgswahrscheinlichkeit in diesem Falle (und damit die Wahrscheinlichkeit von Ereignis E_0) höchstens

$$\Pr[E_0] \leq \frac{1}{N}. \tag{4.2}$$

Angreifer, die irgendwann einmal $H(m^*)$ beim Random Oracle anfragen. Wir zeigen, dass wir aus einem Angreifer $\mathcal{A}$, der Ereignis E_1 hervorruft, einen Angreifer $\mathcal{B}$ konstruieren können, der das RSA-Problem löst. Wir betrachten also einen Algorithmus $\mathcal{B}$, der als Eingabe eine RSA-Challenge (N, e, y) erhält. Algorithmus $\mathcal{B}$ lässt $\mathcal{A}$ als „Subroutine“ laufen, indem er den Challenger und das Random Oracle für $\mathcal{A}$ implementiert.

Den öffentlichen Schlüssel definiert $\mathcal{B}$ als $pk := (N, e)$. $\mathcal{B}$ rät einen Index $\nu \xleftarrow{\$} \{1, \ldots, q_H\}$ gleichverteilt zufällig, und startet $\mathcal{A}$ mit Eingabe pk.

Wir nehmen im Folgenden an, dass der Angreifer vor jeder Anfrage einer Signatur für eine Nachricht m_i *zuerst* den Hashwert $H(m_i)$ vom Random Oracle erfragt. (Alternativ kann man sich vorstellen, dass der Challenger die entsprechende Random Oracle Anfrage „im Namen von $\mathcal{A}$“ stellt, bevor er die Signaturanfrage beantwortet).

Wenn $\mathcal{A}$ seine j-te Random Oracle Anfrage mit Nachricht $\tilde{m}_j$ stellt, so beantwortet $\mathcal{B}$ sie wie folgt:

- Falls $j \neq \nu$, so wählt $\mathcal{B}$ einen Wert $x_j \xleftarrow{\$} \mathbb{Z}_N$ zufällig, berechnet $y_j := x_j^e \bmod N$, und gibt y_j als Hashwert $H(\tilde{m}_j) := y_j$ von $\tilde{m}_j$ zurück. Den Wert x_j „merkt“ sich $\mathcal{B}$, für den Fall, dass $\mathcal{A}$ später eine Signatur für Nachricht $\tilde{m}_j$ anfragt. Da x_j gleichverteilt ist, ist auch y_j gleichverteilt. Die Antwort des Random Oracles ist also genauso verteilt wie im echten Experiment.

- Falls $j = \nu$, so definiert $\mathcal{B}$ $H(\tilde{m}_\nu) := y$, wobei y aus der RSA-Challenge (N, e, y) stammt. Da der Wert y aus der RSA-Challenge ebenfalls gleichverteilt zufällig ist, ist auch in diesem Falle die Antwort des Random Oracles korrekt verteilt.

Man beachte, dass $\mathcal{B}$ für jede Nachricht $\tilde{m}_j$, $j \in \{1, \ldots, q\} \setminus \{\nu\}$, eine Signatur berechnen kann, indem er x_j ausgibt, denn es gilt

$$x_j^e \equiv y_j \equiv H(\tilde{m}_j) \bmod N \quad \iff \quad x_j \equiv H(\tilde{m}_j)^{1/e} \bmod N.$$

Nur für die ν-te Nachricht $\tilde{m}_\nu$ kann $\mathcal{B}$ keine Signatur berechnen.

Falls Ereignis E_1 eintritt, so wird $\mathcal{A}$ irgendwann ein gültiges Nachrichten-Signatur-Paar (m^*, σ^*) mit $\sigma^* \equiv H(m^*)^{1/e} \bmod N$ ausgeben, sodass $\mathcal{A}$ zwar zuvor den Hashwert $H(m^*)$ angefragt hat, aber keine Signatur für m^*. $\mathcal{B}$„hofft“ nun, dass m^* genau die ν-te Nachricht sein wird, also

$$m^* = \tilde{m}_\nu.$$

Da der Index ν gleichverteilt zufällig gewählt wurde, tritt dies mit Wahrscheinlichkeit $1/q_H$ ein. Ist dies der Fall, so kann $\mathcal{B}$ alle Signatur-Anfragen für $\mathcal{A}$ korrekt beantworten, und somit wird $\mathcal{A}$ die gesuchte e-te Wurzel von y für $\mathcal{B}$ berechnen:

$$H(m^*)^{1/e} \equiv H(\tilde{m}_\nu)^{1/e} \equiv y^{1/e} \bmod N.$$

Algorithmus $\mathcal{B}$ kann also das RSA-Problem lösen mit einer Erfolgswahrscheinlichkeit von mindestens

$$\epsilon_{\mathcal{B}} \geq \Pr[E_1]/q_h. \tag{4.3}$$

Durch Einsetzen der Ungleichungen 4.2 und 4.3 in Ungleichung 4.1 erhalten wir

$$\epsilon_B \cdot q_H \geq \Pr[E_1] \geq \epsilon_{\mathcal{A}} - \Pr[E_0] \geq \epsilon_{\mathcal{A}} - 1/N \iff \epsilon_B \geq \frac{\epsilon_{\mathcal{A}} - 1/N}{q_H}.$$

□

Remark 65. Im Beweis von Theorem 64 haben wir die Eigenschaft des Random Oracles, dass Hashwerte gleichverteilt zufällig sind, ausgenutzt. Wir haben das Random Oracle so „programmiert“, dass wir mit guter Wahrscheinlichkeit $1/q_H$ sowohl alle Signaturanfragen des Angreifers beantworten können („Simulation“), als auch eine Lösung des RSA-Problems aus der gefälschten Signatur extrahieren können („Extraktion“). Diese Beweistechnik nennt man auch „Programmierung des Random Oracles“. Sie wird häufig in Sicherheitsbeweisen im Random Oracle Modell angewandt.

Remark 66. Wir mussten im Beweis die Annahme treffen, dass die Hashfunktion H ein Random Oracle ist. Dies ist eine sehr starke Annahme. Eine naheliegende Frage ist nun: *Ist das Random Oracle Modell unbedingt notwendig, um die Sicherheit von RSA-FDH zu zeigen? Oder kann man einen Sicherheitsbeweis für RSA-FDH im Standardmodell finden?* Es gibt einige Ergebnisse, die besagen, dass dies unter bestimmten Bedingungen nicht möglich ist [DOP05, DHT12]. Es ist jedoch nicht ausgeschlossen, dass es einen Beweis gibt, der diese Unmöglichkeitsergebnisse geschickt umgeht. Dies ist ein wichtiges offenes Problem.

Remark 67. Im Beweis wurde die Erfolgswahrscheinlichkeit des RSA-Angreifers $\mathcal{B}$ von unten beschränkt durch

$$\epsilon_{\mathcal{B}} \geq \frac{\epsilon_{\mathcal{A}} - 1/N}{q_H}.$$

Man sieht, dass die Erfolgswahrscheinlichkeit von $\mathcal{B}$ kleiner wird, wenn q_H größer wird. Also: *Je mehr Hash-Anfragen der Signatur-Angreifer $\mathcal{A}$ stellt, umso kleiner wird die Erfolgswahrscheinlichkeit von $\mathcal{B}$.* Dabei kann q_H sehr groß werden, da Hashfunktionen üblicherweise sehr effizient ausgewertet werden können.

Auf der Eurocrypt 2012 haben Kakvi und Kiltz [KK12] einen verbesserten Sicherheitsbeweis vorgestellt, der unabhängig von q_H ist. Dazu muss in [KK12] jedoch eine stärkere Komplexitätsannahme getroffen werden, die so genannte *Phi-Hiding* Annahme.

Der Beweis in [KK12] ist sehr ähnlich zu dem hier vorgestellten Beweis. Daher empfiehlt es sich, dieses Papier einmal zu lesen.

4.3 Gennaro-Halevi-Rabin Signaturen

Das im vorigen Kapitel vorgestellte RSA-FDH Signaturverfahren ist sehr effizient, und damit in der Praxis gut einsetzbar. Leider ist es jedoch nur im Random Oracle Modell beweisbar EUF-CMA-sicher. Ein *praktisches* Signaturverfahren, das auf der RSA-Annahme basiert, und einen Sicherheitsbeweis im Standardmodell hat, ist ein wichtiges offenes Problem.[2]

Eine wichtige Klasse von *praktischen* Signaturverfahren basiert auf der so genannten *strong*-RSA-Annahme, welche eng mit der klassischen RSA-Annahme verwandt ist. Es gibt zahlreiche Signaturverfahren, die auf Grundlage der Strong-RSA-Annahme sicher bewiesen werden können [GHR99, CS99, Fis03, Sch11].

In diesem Kapitel beschreiben wir das Signaturverfahren von Gennaro, Halevi und Rabin [GHR99] (im Folgenden „GHR-Verfahren“ genannt). Dieses Verfahren ist der einfachste Vertreter dieser Klasse.

4.3.1 Die Strong-RSA-Annahme

Definition 68. Sei $N := PQ$ das Produkt von zwei Primzahlen und sei $y \xleftarrow{\$} \mathbb{Z}_N$ eine zufällige Zahl modulo N. Das durch (N, y) gegebene *Strong-RSA-Problem* ist: Berechne $x \in \mathbb{Z}_N$ und $e \in \mathbb{N}$, $e > 1$, sodass

$$x^e \equiv y \bmod N.$$

Die Strong-RSA-Annahme besagt, dass das Strong-RSA-Problem „schwer“ ist.

Das Strong-RSA-Problem ist also nahezu identisch zum RSA-Problem (Definition 29), jedoch ist beim klassischen RSA-Problem der Exponent e vorgegeben, während beim Strong-RSA-Problem der Exponent e vom Angreifer selbst gewählt werden darf.

Remark 69. Die Strong-RSA-Annahme ist eine stärkere Annahme als die RSA-Annahme, daher ist die Bezeichnung treffend. Im Gegensatz dazu ist das Strong-RSA-Problem jedoch *leichter* als das RSA-Problem, daher ist die Bezeichnung in diesem Fall nicht so richtig passend. Sie hat sich aber trotzdem durchgesetzt, da sie gut zur dazugehörigen Annahme passt.

[2] Es ist bislang nur ein recht ineffizientes RSA-basiertes Verfahren bekannt, welches beweisbar EUF-CMA-sicher ist. Dieses werden wir in Kapitel 4.4 vorstellen.

4.3.2 GHR Signaturen

Wir nehmen im Folgenden eine Hashfunktion $h : \{0,1\}^* \to \mathbb{P}$ an, die Bit-Strings (Nachrichten) auf die Menge der Primzahlen abbildet. Aus technischen Gründen nehmen wir weiterhin an, dass diese Primzahlen stets größer sind als der RSA-Modulus N der im Folgenden verwendet wird. Wir skizzieren später verschiedene Möglichkeiten, wie eine solche Funktion h konstruiert werden kann.

Betrachten wir das folgende Signaturverfahren $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$:

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus erzeugt einen RSA-Modulus $N = PQ$, wobei P und Q zwei zufällig gewählte Primzahlen sind. Außerdem wird $s \xleftarrow{\$} \mathbb{Z}_N$ gewählt.

Die Schlüssel sind $pk := (N, s, h)$ und $sk := \phi(N) = (P-1)(Q-1)$.

$\mathsf{Sign}(sk, m)$**.** Um eine Nachricht $m \in \{0,1\}^n$ zu signieren, wird $d := 1/h(m) \bmod \phi(N)$ berechnet. Die Signatur ist

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

also eine $h(m)$-te Wurzel von s. Beachten Sie, dass $1/h(m)$ nur dann existiert, wenn $\mathrm{ggT}(h(m), \phi(N)) = 1$. Dies ist jedoch dadurch garantiert, dass h auf Primzahlen abbildet, die größer als N sind.

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus gibt 1 aus, wenn

$$\sigma^{h(m)} \equiv s \bmod N.$$

gilt, also σ eine $h(m)$-te Wurzel von s ist. Ansonsten wird 0 ausgegeben.

Theorem 70. *Für jeden PPT-Angreifer $\mathcal{A}$ der die* EUF-naCMA*-Sicherheit von Σ in Zeit $t_{\mathcal{A}}$ mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{A}}$ bricht, existiert ein PPT-Angreifer $\mathcal{B}$, der in Zeit $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ läuft und*

- *entweder die Kollisionsresistenz von h bricht mit Erfolgswahrscheinlichkeit*

$$\epsilon_{\mathsf{coll}} \geq \epsilon_{\mathcal{A}}/2,$$

- *oder das Strong-RSA-Problem löst mit Erfolgswahrscheinlichkeit*

$$\epsilon_{\mathsf{sRSA}} \geq \epsilon_{\mathcal{A}}/2.$$

Beweis. Sei $m_1, \ldots, m_q$ die Liste der Nachrichten, für die $\mathcal{A}$ im EUF-naCMA-Experiment Signaturen angefragt hat. Wir betrachten wieder zwei Ereignisse.

- Ereignis E_0 tritt ein, wenn $\mathcal{A}$ eine gültige Fälschung (m^*, σ^*) ausgibt, und es existiert ein Index i mit $h(m^*) = h(m_i)$.
- Ereignis E_1 tritt ein, wenn $\mathcal{A}$ eine gültige Fälschung (m^*, σ^*) ausgibt, und es gilt $h(m^*) \neq h(m_i)$ für alle $i \in \{1, \ldots, q\}$.

Jeder erfolgreiche Angreifer ruft entweder Ereignis E_0 oder Ereignis E_1 hervor, also gilt wieder

$$\epsilon_{\mathcal{A}} \leq \Pr[E_0] + \Pr[E_1],$$

und somit muss auch entweder $\Pr[E_0] \geq \epsilon_A/2$ oder $\Pr[E_1] \geq \epsilon_A/2$ oder beides gelten.

Ereignis E_0. Falls Ereignis E_0 eintritt, können wir die Kollisionsresistenz der Funktion h brechen. Dieser Teil des Beweises ist recht simpel, und sehr ähnlich zu zuvor bereits vorgestellten Beweisen, welche mit der Kollisionsresistenz einer Funktion argumentieren. Daher lassen wir ihn aus.

Ereignis E_1. Falls Ereignis E_1 eintritt, so kann $\mathcal{B}$ das Strong-RSA-Problem wie folgt lösen. $\mathcal{B}$ erhält als Eingabe eine Instanz (N, y) des Strong-RSA-Problems, und muss (x, e) ausgeben sodass $e > 1$ und $x^e \equiv y \bmod N$.

Zu Beginn des Experiments erhält $\mathcal{B}$ vom Angreifer $\mathcal{A}$ eine Liste von Nachrichten $m_1, \ldots, m_q$. Dann berechnet $\mathcal{B}$ den Wert $s \in \mathbb{Z}_N$ als

$$s := y^{\prod_{i \in \{1,\ldots,q\}} h(m_i)} \bmod N.$$

Der öffentliche Schlüssel ist $pk = (N, s)$, und hat die korrekte Verteilung.

Die j-te Signatur für Nachricht m_j bestimmt $\mathcal{B}$ durch Berechnung von

$$\sigma_j := y^{\prod_{i \in \{1,\ldots,q\} \setminus \{j\}} h(m_i)} \bmod N.$$

Der Wert σ_j ist eine gültige Signatur für Nachricht m_j, denn die Verifikationsgleichung ist erfüllt:

$$\sigma_j^{h(m_j)} \equiv \left(y^{\prod_{i \in \{1,\ldots,q\} \setminus \{j\}} h(m_i)}\right)^{h(m_j)} \equiv y^{\prod_{i \in \{1,\ldots,q\}} h(m_i)} \equiv s \bmod N.$$

Mit Wahrscheinlichkeit $\Pr[E_1]$ gibt $\mathcal{A}$ eine Fälschung (m^*, σ^*) aus, sodass gilt

- $h(m^*) \neq h(m_i)$ für alle $i \in \{1, \ldots, q\}$, und
- $(\sigma^*)^{h(m^*)} \equiv s \bmod N$.

Daher erhält $\mathcal{B}$ von $\mathcal{A}$ eine Lösung σ^* der Gleichung

$$(\sigma^*)^{h(m^*)} \equiv s \equiv y^{\prod_{i \in \{1,\ldots,q\}} h(m_i)} \bmod N, \tag{4.4}$$

bei der $\mathrm{ggT}(h(m^*), \prod_{i \in \{1,\ldots,q\}} h(m_i)) = 1$ gilt, weil die Funktion h auf Primzahlen abbildet und $h(m^*) \neq h(m_i)$ für alle $i \in \{1, \ldots, q\}$ gilt.

Um daraus eine Lösung für das Strong-RSA-Problem zu extrahieren, berechnet $\mathcal{B}$ mit Hilfe von „Shamir's Trick" (Lemma 31) aus Gleichung 4.4 den Wert x, sodass $x^{h(m^*)} \equiv y \bmod N$, und gibt $(x, h(m^*))$ als Lösung aus. □

EUF-CMA-sichere Signaturen basierend auf der Strong-RSA-Annahme. Wir haben bewiesen, dass das GHR-Verfahren EUF-naCMA-sicher ist. Unser eigentliches Ziel ist jedoch EUF-CMA-Sicherheit.

Hier können wir die Transformation aus Kapitel 2.4 auf das GHR-Verfahren und die RSA-basierten Einmalsignaturen aus Kapitel 2.3.2 (oder alternativ die RSA-basierte Chamäleon-Hashfunktion aus Kapitel 3.3.2) anwenden. Da die RSA-Annahme von der Strong-RSA-Annahme impliziert wird, erhalten wir somit ein EUF-CMA-sicheres Signaturverfahren basierend auf der Strong-RSA-Annahme.

4.3.3 Hashfunktionen, die auf Primzahlen abbilden

Heuristische Konstruktion. Sei $H : \{0,1\}^* \to \{0,1\}^\ell$ eine kollisionsresistente Hashfunktion. Ein möglicher Ansatz, um aus dieser Hashfunktion H die oben benötigte Funktion h zu konstruieren, ist die Funktion h zu definieren als

$$h(m) := H(m||\gamma),$$

wobei $\gamma \in \mathbb{N}$ die kleinste natürliche Zahl ist, sodass der Wert $H(m||\gamma)$, aufgefasst als natürliche Zahl im Intervall $[0, 2^\ell - 1]$, eine Primzahl ist. Bei der Auswertung von h wird γ einfach so lange inkrementiert, bis $H(m||\gamma)$ prim ist.

Wenn die Hashwerte $H(m||\gamma)$ einigermaßen gleichmäßig über dem Intervall $[0, 2^\ell - 1]$ verteilt sind, dann erwartet man (nach dem Primzahlsatz) ungefähr $\log 2^\ell = \ell$ Versuche, um ein γ zu finden, sodass $H(m||\gamma)$ prim ist. Die Auswertung von h ist also einigermaßen effizient (im asymptotischen Sinne), falls die Auswertung von H effizient ist. Diese Konstruktion ist natürlich heuristisch, und hängt stark von den Eigenschaften der Funktion H ab.

Excercise 71. Geben Sie eine Hashfunktion $H : \{0,1\}^* \to [0, 2^\ell - 1]$ an, die kollisionsresistent ist, aber für die der Wert $H(m||\gamma) \in [0, 2^\ell - 1]$ niemals eine Primzahl ist. Sie können als Baustein eine kollisionsresistente Hashfunktion $H' : \{0,1\}^* \to B$ mit geeigneter Bildmenge B verwenden.

Konstruktion basierend auf einer Chamäleon-Hashfunktion. Sei $N' \in \mathbb{N}$ eine natürliche Zahl, und

$$\mathsf{ch} : \{0,1\}^n \times \mathbb{Z}_{N'} \to [0, N' - 1]$$

die RSA-basierte Chamäleon-Hashfunktion aus Kapitel 3.3.2. Diese Chamäleon-Hashfunktion hat die hilfreichen Eigenschaften, dass sie

- auf die *natürlichen Zahlen* im Intervall $[0, N' - 1]$ abbildet, und
- für eine beliebige Nachricht m und einen gleichverteilt zufälligen Wert $r \xleftarrow{\$} \mathbb{Z}_{N'}$ ist der Hash-Wert $\mathsf{ch}(m, r)$ *gleichverteilt* über $[0, N' - 1]$.

Basierend auf dieser Chamäleon-Hashfunktion lässt sich eine (randomisierte) Hashfunktion h, die auf Primzahlen abbildet, konstruieren. Um den Hashwert $h(m)$ einer Nachricht m zu berechnen, geht man wie folgt vor:

1. Wähle so lange zufällige Werte $(m', r') \xleftarrow{\$} \{0,1\}^n \times \mathbb{Z}_{N'}$, bis $p = \mathsf{ch}(m', r') \in [0, N' - 1]$ eine Primzahl ist (dies geht relativ schnell, nach dem Primzahlsatz sind ungefähr $\log N'$ Versuche nötig).
2. Berechne durch $r = \mathsf{TrapColl}(\tau, m', r', m)$ eine Randomness r, sodass $\mathsf{ch}(m, r) = p$.

Die Randomness r wird dann zusammen mit der Signatur an den Empfänger gesandt. Die Kollisionsresistenz dieser Funktion h folgt aus der Kollisionsresistenz der Chamäleon-Hashfunktion.

Ein weiterer Vorteil ist auch, dass man nun die Chamäleon-Eigenschaft von h auch gleich mitbenutzen kann, um EUF-CMA-Sicherheit des resultierenden Signaturverfahrens zu beweisen. Eine zweite Chamäleon-Hashfunktion zur Anwendung der Transformation aus Kapitel 2.4 ist somit nicht nötig.

4.4 Hohenberger-Waters Signaturen

Ein beweisbar EUF-CMA-sicheres Signaturverfahren das auf der Schwierigkeit des RSA-Problems basiert war sehr lange ein offenes Problem. Im Jahr 2009 haben Hohenberger und Waters [HW09b] das erste solche Verfahren vorgestellt.

Die Konstruktion von Hohenberger und Waters kann man auch als eine clevere Erweiterung des GHR-Verfahrens (Kapitel 4.3) ansehen. Diese basiert auf drei Beobachtungen, die wir in den folgenden Kapiteln beschreiben:

- Man kann zeigen dass GHR-Signaturen sicher sind unter der (klassischen) RSA-Annahme, wenn man ein schwächeres Sicherheitsziel betrachtet, nämlich „selektive Sicherheit“ (SUF-naCMA).
- Es gibt eine *generische Transformation*, mit der man aus einem SUF-naCMA-sicheren Verfahren ein EUF-naCMA-sicheres Verfahren konstruieren kann.
- Diese Transformation lässt sich besonders effizient auf das GHR-Verfahren anwenden.

4.4.1 Selektive Sicherheit von GHR-Signaturen

Ein natürlicher Ansatz, um ein RSA-basiertes Signaturverfahren zu konstruieren, ist sich zuerst mal ein bekanntes Strong-RSA-basiertes Verfahren, wie das GHR-Verfahren, anzusehen. Warum reicht die RSA-Annahme im Beweis nicht aus?

Wie immer, wenn wir die Sicherheit eines Signaturverfahrens beweisen wollen, mussten wir im Sicherheitsbeweis der GHR-Signaturen auf zwei Dinge achten:

Simulation. Wir (bzw. der Algorithmus $\mathcal{B}$, den wir im Beweis von Theorem 70 konstruiert haben) müssen in der Lage sein, im Beweis einen gültigen pk und gültige Signaturen für die Nachrichten zu „simulieren“.

Hier konnten wir ausnutzen, dass der Angreifer im EUF-naCMA-Experiment die Nachrichten, für die er Signaturen sehen möchte, zu Beginn des Experiments ausgeben muss – *bevor* er den pk sieht. Daher konnten wir den Wert $s \in \mathbb{Z}_N$, der im pk steht, so aufsetzen, dass wir „genau die richtigen“ $h(m_i)$-ten Wurzeln aus s berechnen konnten:

$$s := y^{\prod_i h(m_i)} \bmod N$$

Dieser Schritt könnte im Beweis des GHR-Verfahrens genauso funktionieren, wenn wir statt der Strong-RSA die RSA-Annahme treffen. Hier scheint also nicht das Problem zu liegen.

Extraktion. Wenn uns der Angreifer eine Fälschung (m^*, σ^*) mit

$$s \equiv (\sigma^*)^{h(m^*)} \bmod N$$

liefert, dann konnten wir (falls $\mathcal{A}$ keine Kollision für h gefunden hat) mit Hilfe von Shamir's Trick eine *neue* $h(m^*)$-te Wurzel von y berechnen, und so das Strong-RSA-Problem lösen.

Leider hatten wir (d.h. der Simulator im Beweis) keinen Einfluß auf den Wert $h(m^*)$, denn m^* wird im EUF-naCMA-Experiment vom Angreifer gewählt. Um das Strong-RSA-Problem zu lösen ist jedoch ausreichend *irgendeine* nichttriviale Wurzel von y zu berechnen – wir müssen nichts über $h(m^*)$ wissen, ausser dass $h(m^*) \neq h(m_i)$ für alle $i \in \{1, \ldots, q\}$ gelten muss. Beim RSA-Problem ist das anders, denn wir müssen eine *ganz bestimmte* e-te Wurzel ziehen. Nämlich für genau den Wert e, den uns die gegebene Instanz (N, e, y) des RSA-Problems vorgibt.

Hier liegt also das Problem. Der Angreifer im EUF-naCMA-Experiment gewinnt, wenn er eine gültige Fälschung (m^*, σ^*) für eine *beliebige neue Nachricht* m^* ausgibt. Daher ist es schwierig für uns, den Angreifer dazu zu bewegen, dass er uns „genau die richtige" e-te Wurzel berechnet.

Selektive Sicherheit. Können wir einen Sicherheitsbeweis für GHR-Signaturen unter der (klassischen) RSA-Annahme angeben, wenn wir den Angreifer im Experiment dazu zwingen eine Fälschung für eine *bestimmte* Nachricht m^* auszugeben?

Wir schwächen das EUF-naCMA-Experiment etwas ab, und betrachten im Folgenden das *selective unforgeability under non-adaptive chosen-message attack* (SUF-naCMA) Sicherheitsexperiment, welches mit einem Angreifer $\mathcal{A}$, Challenger $\mathcal{C}$ und Signaturverfahren $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ wie folgt abläuft (siehe auch Abbildung 4.2):

1. Der Angreifer legt sich zu Beginn auf die Nachricht m^* fest, für die er eine Signatur fälschen wird.

2. Dann gibt der Angreifer q Nachrichten $(m_1, \ldots, m_q)$ aus, sodass $m_i \neq m^*$ für alle $i \in \{1, \ldots, q\}$. Dies sind die Nachrichten, für die $\mathcal{A}$ eine Signatur erfragt.

3. Der Challenger $\mathcal{C}$ generiert ein Schlüsselpaar $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^k)$, und berechnet für jedes $i \in \{1, \ldots, q\}$ eine Signatur $\sigma_i := \mathsf{Sign}(sk, m_i)$. Der Angreifer erhält pk und die Signaturen $(\sigma_1, \ldots, \sigma_q)$.

4. Am Ende gibt $\mathcal{A}$ eine Signatur σ^* aus. Er „gewinnt" das Spiel, wenn

$$\mathsf{Vfy}(pk, m^*, \sigma^*) = 1.$$

   $\mathcal{A}$ gewinnt also, wenn σ^* eine gültige Signatur für m^* ist.

Definition 72. Wir sagen, dass $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ sicher ist im Sinne von SUF-naCMA, falls für alle PPT-Angreifer $\mathcal{A}$ im SUF-naCMA-Experiment gilt, dass

$$\Pr[\mathcal{A}^{\mathcal{C}} = \sigma^* : \mathsf{Vfy}(pk, m^*, \sigma^*) = 1] \leq \mathsf{negl}(k)$$

für eine vernachlässigbare Funktion negl im Sicherheitsparameter.

Wir zeigen im Folgenden, dass man die SUF-naCMA-Sicherheit von GHR-Signaturen unter der RSA-Annahme beweisen kann, wenn man die Funktion h geeignet definiert.

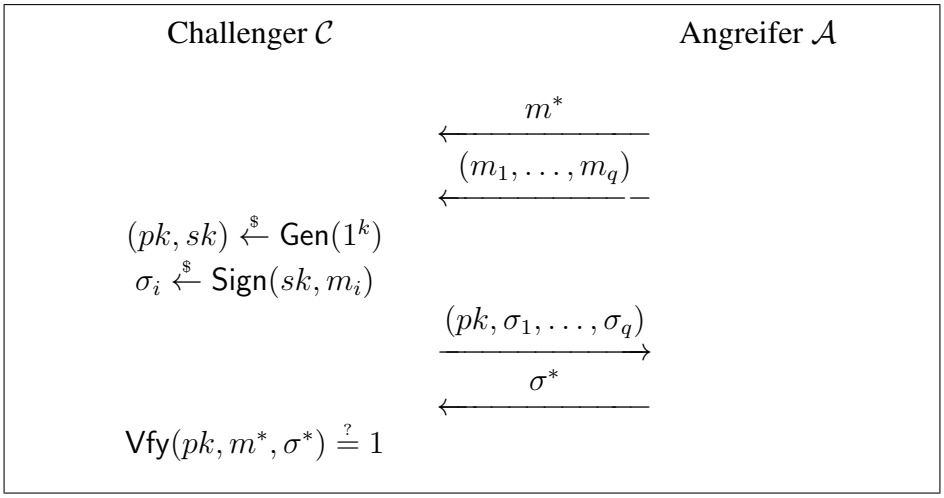

Abbildung 4.2: Das SUF-naCMA Sicherheitsexperiment.

Definition der Funktion h. Sei

$$\mathsf{PRF} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^\ell$$

eine Pseudozufallsfunktion (vgl. Definition 37) mit Schlüsselraum $\{0,1\}^k$, die Bit-Strings beliebiger Länge auf ℓ-Bit-Strings abbildet. Wir definieren Funktion h, die Bit-Strings auf Primzahlen abbildet, wie folgt.

- Zur Erzeugung der Funktionsbeschreibung wird ein Schlüssel $\kappa \xleftarrow{\$} \{0,1\}^k$ (der Seed) für die PRF gewählt, sowie ein zufälliger Bit-String $\alpha \xleftarrow{\$} \{0,1\}^\ell$. Die Funktion h wird beschrieben durch die Funktion PRF sowie die Werte (κ, α).
- Die Funktion
  $$h : \{0,1\}^n \to \mathbb{P}_\ell,$$
  welche n-Bit-Strings auf ℓ-Bit Primzahlen abbildet, wird definiert als
  $$h(m) := \mathsf{PRF}(\kappa, m||\gamma) \oplus \alpha,$$
  wobei $\gamma \in \mathbb{N}$ die kleinste natürliche Zahl ist, sodass der Wert $\mathsf{PRF}(\kappa, m||\gamma) \oplus \alpha \in \{0,1\}^\ell$, aufgefasst als natürliche Zahl im Intervall $[0, 2^\ell - 1]$, eine Primzahl ist (so ähnlich wie bei der heuristischen Konstruktion aus Kapitel 4.3.3).

Bei der Auswertung von h wird γ so lange inkrementiert, bis $\mathsf{PRF}(\kappa, m||\gamma) \oplus \alpha$ prim ist. Nach dem Primzahlsatz erwartet man dass dies recht schnell geht, man braucht ungefähr $\log 2^\ell = \ell$ Versuche.

GHR-Signaturen mit spezieller Funktion h. Wenn man diese Funktion h in das GHR-Signaturverfahren einsetzt, ergibt sich das folgende Signaturverfahren.

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus erzeugt einen RSA-Modulus $N = PQ$, wobei P und Q zwei zufällig gewählte Primzahlen sind. Außerdem wird $s \xleftarrow{\$} \mathbb{Z}_N$ gewählt.

Um die Funktion h zu beschreiben, werden zusätzlich $\kappa \xleftarrow{\$} \{0,1\}^k$ und $\alpha \xleftarrow{\$} \{0,1\}^\ell$ gewählt.

Die Schlüssel sind $pk := (N, s, (\mathsf{PRF}, \kappa, \alpha))$ und $sk := \phi(N) = (P-1)(Q-1)$.

$\mathsf{Sign}(sk, m)$**.** Um eine Nachricht $m \in \{0,1\}^n$ zu signieren, wird $d := 1/h(m) \bmod \phi(N)$ berechnet. Die Signatur ist

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

also eine $h(m)$-te Wurzel von s.

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus gibt 1 aus, wenn

$$\sigma^{h(m)} \equiv s \bmod N.$$

gilt, also σ eine $h(m)$-te Wurzel von s ist. Ansonsten wird 0 ausgegeben.

Dieses Verfahren kann unter der RSA-Annahme SUF-naCMA-sicher beweisen werden. Man muss sich jedoch auf Instanzen des RSA-Problems (N, e, y) beschränken, bei denen e eine Primzahl ist.

Beweisidee. Sei $\mathcal{B}$ ein Algorithmus, der einen SUF-naCMA-Angreifer $\mathcal{A}$ benutzt, um das RSA-Problem zu lösen. $\mathcal{B}$ erhält als Eingabe eine Instanz (N, e, y) des RSA-Problems, wobei wir im Folgenden annehmen, dass e eine *zufällige* ℓ-Bit Primzahl ist.

Die Grundidee des Sicherheitsbeweises ist, dass $\mathcal{B}$ die Funktion h geschickt so „programmiert", dass $h(m^*) = e$ gilt. Wenn dies gelingt, dann kann $\mathcal{B}$ die gleiche Strategie für Simulation und Extraktion wie im Strong-RSA-basierten Sicherheitsbeweis (Beweis von Theorem 70) anwenden. Da der Angreifer $\mathcal{A}$ im SUF-naCMA-Experiment eine Signatur für Nachricht m^* fälschen muss, und $h(m^*) = e$ gilt, wird $\mathcal{A}$ für $\mathcal{B}$ genau die gesuchte e-te Wurzel aus y berechnen.

$\mathcal{B}$ definiert daher die Funktion h, indem er zuerst einen zufälligen Seed $\kappa \xleftarrow{\$} \{0,1\}^k$ wählt, sowie einen zufälligen Wert $\gamma \xleftarrow{\$} \{1, \ldots, \ell\}$. Der Wert $\alpha \in \{0,1\}^\ell$ wird definiert als

$$\alpha := \mathsf{PRF}(k, m^* || \gamma) \oplus e.$$

Wenn man nun annimmt, dass γ die kleinste natürliche Zahl ist, sodass $\mathsf{PRF}(k, m^*) \oplus \alpha$ prim ist, dann ist

$$h(m^*) = \mathsf{PRF}(k, m^* || \gamma) \oplus \alpha = e.$$

Durch geeignete Wahl von α konnte $\mathcal{B}$ also die Funktion h genau so programmieren, dass $h(m^*) = e$ gilt.

4.4.2 Von SUF-naCMA-Sicherheit zu EUF-naCMA-Sicherheit

Unser Ziel ist die Konstruktion eines EUF-naCMA-sicheren Signaturverfahrens basierend auf der RSA-Annahme. Bislang haben wir ein RSA-basiertes Signaturverfahren, welches jedoch leider nur SUF-naCMA-sicher ist. In diesem Kapitel beschreiben wir eine generische Transformation, mit der man aus einem SUF-naCMA-sicheren Verfahren ein EUF-naCMA-sicheres Verfahren konstruieren kann. Explizit beschrieben wurde diese Transformation in [BK10], die Idee dazu stammt jedoch aus [HW09b].

Präfixe. Bevor wir die Konstruktion der Transformation angeben, müssen wir noch „Präfixe“ von Bit-Strings definieren und ein technisches Lemma angeben, das wir im Beweis benutzen werden.

Definition 73. Sei $m \in \{0,1\}^n$ ein Bit-String. Wir schreiben $m_{|w}$, und sagen dass $m_{|w}$ das w-te Präfix von m ist, wenn

- $|m_{|w}| = w$ gilt, also die Länge von $m_{|w}$ genau w ist, und
- $m_{|w}$ identisch ist zu den ersten w Bits von m.

Example 74. Sei $n = 4$ und $m = 1011 \in \{0,1\}^4$. Dann ist $m_{|1} = 1$, $m_{|2} = 10$, $m_{|3} = 101$ und $m_{|4} = 1011$.

Im Beweis von Theorem 77 benutzen wir das folgende Lemma.

Lemma 75. *Sei $m_1, \ldots, m_q$ eine Liste von q beliebigen Nachrichten, wobei $m_i \in \{0,1\}^n$ für alle $i \in \{1, \ldots, q\}$. Sei $m^* \in \{0,1\}^n$, sodass $m^* \neq m_i$ für alle $i \in \{1, \ldots, q\}$.*

1. *Es existiert ein* kürzestes *Präfix $m^*_{|w}$ von m^*, das* kein *Präfix von irgendeiner Nachricht m_i aus der Liste $m_1, \ldots, m_q$ ist.*
2. *Selbst wenn wir nur $(m_1, \ldots, m_q)$ gegeben haben, aber nicht m^*, können wir dieses Präfix korrekt bestimmen, mit einer Erfolgswahrscheinlichkeit von mindestens $1/(qn)$.*

Excercise 76. Beweisen Sie Lemma 75.

Die Transformation. Sei $\widetilde{\Sigma} = (\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Sign}}, \widetilde{\mathsf{Vfy}})$ ein (SUF-naCMA-sicheres) Signaturverfahren. Wir konstruieren daraus ein neues, (EUF-naCMA-sicheres) Signaturverfahren $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$.

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus erzeugt ein Schlüsselpaar durch $(pk, sk) \xleftarrow{\$} \widetilde{\mathsf{Gen}}(1^k)$.

$\mathsf{Sign}(sk, m)$**.** Um eine Nachricht $m \in \{0,1\}^n$ zu signieren, werden alle Präfixe von m mit $\widetilde{\mathsf{Sign}}$ signiert:

$$\tilde{\sigma}^{(w)} := \widetilde{\mathsf{Sign}}(sk, m_{|w}) \qquad \forall w \in \{1, \ldots, n\}.$$

Die Signatur für m besteht aus $\sigma := (\tilde{\sigma}^{(1)}, \ldots, \tilde{\sigma}^{(n)})$.

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus erhält $\sigma := (\tilde{\sigma}^{(1)}, \ldots, \tilde{\sigma}^{(n)})$ und prüft, ob der Wert $\tilde{\sigma}^{(w)}$ eine gültige Signatur für $m_{|w}$ ist, für alle $w \in \{1, \ldots, n\}$:

$$\widetilde{\mathsf{Vfy}}(pk, m_{|w}, \tilde{\sigma}^{(w)}) \stackrel{?}{=} 1 \qquad \forall w \in \{1, \ldots, n\}.$$

Die Grundidee der Transformation ist also sehr einfach: Nicht nur die Nachricht $m = m_{|n}$ wird signiert, sondern auch alle Präfixe von m.

Wir können zeigen, dass dieses Verfahren Σ tatsächlich EUF-naCMA-sicher ist, falls das zugrundeliegende Verfahren $\widetilde{\Sigma}$ SUF-naCMA-sicher ist. Dies wird wieder durch Widerspruch beweisen.

Theorem 77. *Für jeden PPT-Angreifer $\mathcal{A}$, der die EUF-naCMA-Sicherheit von Σ bricht in Zeit $t_{\mathcal{A}}$ mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{A}}$, existiert ein PPT-Angreifer $\mathcal{B}$, der die SUF-naCMA-Sicherheit von $\widetilde{\Sigma}$ bricht in Zeit $t_{\mathcal{B}}$ mit Erfolgswahrscheinlichkeit*

$$\epsilon_{\mathcal{B}} \geq \frac{\epsilon_{\mathcal{A}}}{qn},$$

wobei n die Nachrichtenlänge und q die Anzahl der Chosen-Message Signatur-Anfragen von $\mathcal{A}$ ist.

Beweis von Theorem 77. $\mathcal{B}$ startet $\mathcal{A}$, und erhält eine Liste von Nachrichten $(m_1, \ldots, m_q)$. Basierend auf dieser Liste rät $\mathcal{B}$ das kürzestes Präfix $m^*_{|w}$ von m^*, sodass $m^*_{|w}$ kein Präfix von irgendeiner Nachricht m_i ist, $i \in \{1, \ldots, q\}$, und legt sich gegenüber dem Challenger darauf fest eine Signatur für die Nachricht $\tilde{m}^* := m^*_{|w}$ zu fälschen.

Nun erfragt $\mathcal{B}$ noch Signaturen für alle Präfixe der Nachrichten in $(m_1, \ldots, m_q)$. Dazu definiert $\mathcal{B}$ eine neue Liste von Nachrichten $(\tilde{m}_1, \ldots, \tilde{m}_{qn})$ als

$$\begin{aligned}
(\tilde{m}_1, \ldots, \tilde{m}_n) &:= (m_{1|1}, \ldots, m_{1|n}) \\
(\tilde{m}_{n+1}, \ldots, \tilde{m}_{2n}) &:= (m_{2|1}, \ldots, m_{2|n}) \\
&\vdots \\
(\tilde{m}_{(q-1)n+1}, \ldots, \tilde{m}_{qn}) &:= (m_{q|1}, \ldots, m_{q|n}).
\end{aligned}$$

Die Liste $(\tilde{m}_1, \ldots, \tilde{m}_{qn})$ besteht also aus allen möglichen Präfixen aller Nachrichten in der Liste $(m_1, \ldots, m_q)$. Dann erfragt $\mathcal{B}$ Signaturen für alle Nachrichten in $(\tilde{m}_1, \ldots, \tilde{m}_{qn})$ beim Challenger.

Vom Challenger erhält $\mathcal{B}$ den öffentlichen Schlüssel pk sowie eine Liste von Signaturen $(\tilde{\sigma}_1, \ldots, \tilde{\sigma}_{qn})$. Nun definiert $\mathcal{B}$

$$\begin{aligned}
\sigma_1 &:= (\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n) \\
\sigma_2 &:= (\tilde{\sigma}_{n+1}, \ldots, \tilde{\sigma}_{2n}) \\
&\vdots \\
\sigma_q &:= (\tilde{\sigma}_{(q-1)n+1}, \ldots, \tilde{\sigma}_{qn}).
\end{aligned}$$

Somit besteht eine Signatur σ_i aus einer Liste von Signaturen über alle Präfixe der Nachricht m_i, und ist damit eine gültige Signatur für Nachricht m_i bezüglich des Signaturverfahrens Σ. Den pk und die Liste $(\sigma_1, \ldots, \sigma_q)$ leitet $\mathcal{B}$ an $\mathcal{A}$ weiter.

Mit Wahrscheinlichkeit $\epsilon_{\mathcal{A}}$ wird der Angreifer $\mathcal{A}$ daher eine Signaturfälschung (m^*, σ^*) ausgeben, wobei $\sigma^* = (\tilde{\sigma}^*_1, \ldots, \tilde{\sigma}^*_n)$ Signaturen über jedes Präfix von m^* enthält. Insbesondere enthält σ^* eine Signatur über das kürzeste Präfix, welches kein Präfix irgendeiner Nachricht m_i ist.

Falls $\mathcal{B}$ nun richtig geraten hat, so ist $m^*_{|w}$ das kürzeste Präfix von m^*, welches kein Präfix von irgendeiner Nachricht m_i ist, $i \in \{1, \ldots, q\}$. In diesem Fall kann $\mathcal{B}$ die Signatur $\tilde{\sigma}^*_w$ als gültige Fälschung ausgeben. Laut Lemma 75 rät $\mathcal{B}$ mit einer Wahrscheinlichkeit von mindestens $1/(qn)$ korrekt. Somit ist die Erfolgswahrscheinlichkeit von $\mathcal{B}$ mindestens

$$\epsilon_{\mathcal{B}} \geq \frac{\epsilon_{\mathcal{A}}}{qn}.$$

□

Die in diesem Kapitel vorgestellte Transformation ist sehr attraktiv, denn sie ist *generisch*. Das bedeutet, sie funktioniert mit beliebigen SUF-naCMA-sicheren Signaturverfahren. Sie eröffnet damit einen neuen Weg zur Konstruktion von EUF-CMA-sicheren Signaturverfahren:

$$\text{SUF-naCMA} \overset{\text{Dieses Kapitel}}{\Longrightarrow} \text{EUF-naCMA} \overset{\text{Kapitel 2.4}}{\Longrightarrow} \text{EUF-CMA}.$$

Einen Nachteil gibt es jedoch noch: Wenn man die generische Transformation in diesem Kapitel anwendet, dann werden die Signaturen sehr groß. Um eine n-Bit Nachricht $m \in \{0,1\}^n$ zu signieren, muss jedes Präfix der Nachricht m einzeln signiert werden. Daher wächst die Größe der Signaturen um den Faktor n.

Im Falle von GHR-Signaturen ist es jedoch glücklicherweise (und interessanterweise) möglich, diesen Faktor n zu vermeiden. Dies erklären wir im nächsten Kapitel.

4.4.3 Clevere „Kompression“ von GHR-Signaturen

Wenn man die Transformation aus dem vorigen Kapitel naiv auf GHR-Signaturen anwendet, dann werden die Signaturen recht groß. Sei $pk = (N, s, h)$ ein GHR-*public key*. Eine Signatur über eine n-Bit Nachricht $m \in \{0,1\}^n$ besteht aus n Werten:

$$\sigma' = (s^{1/h(m_{|1})}, \ldots, s^{1/h(m_{|n})}) \in \mathbb{Z}_N^n.$$

Im Falle von GHR-Signaturen geht es jedoch wesentlich effizienter. Wir berechnen einfach die Signatur als

$$\sigma = s^{\prod_{j=1}^{n} 1/h(m_{|j})}.$$

Die n Werte der Signatur σ' können also in einen Wert σ „komprimiert“ werden.

Das Hohenberger-Waters Verfahren. Wenn wir diese verbesserte Transformation auf GHR-Signaturen anwenden, dann erhalten wir das Signaturverfahren von Hohenberger und Waters [HW09b].

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus erzeugt einen RSA-Modulus $N = PQ$, wobei P und Q zwei zufällig gewählte Primzahlen sind. Außerdem wird $s \xleftarrow{\$} \mathbb{Z}_N$ gewählt, sowie $\kappa \xleftarrow{\$} \{0,1\}^k$ und $\alpha \xleftarrow{\$} \{0,1\}^\ell$, um die Funktion h wie zuvor zu beschreiben.

Die Schlüssel sind $pk := (N, s, (\mathsf{PRF}, \kappa, \alpha))$ und $sk := \phi(N) = (P-1)(Q-1)$.

$\mathsf{Sign}(sk, m)$**.** Um eine Nachricht $m \in \{0,1\}^n$ zu signieren, wird

$$d := \frac{1}{\prod_{i=1}^{n} h(m_{|i})} \bmod \phi(N)$$

berechnet. Die Signatur ist

$$\sigma := s^d \bmod N.$$

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus gibt 1 aus, wenn

$$\sigma^{\prod_{i=1}^{n} h(m_{|i})} \equiv s \bmod N.$$

gilt. Ansonsten wird 0 ausgegeben.

Man kann zeigen, dass dieses Verfahren EUF-naCMA-sicher ist. In Kombination mit einer RSA-basierten Einmalsignatur oder Chamäleon-Hashfunktion ergibt sich so das erste RSA-basierte Signaturverfahren, dessen EUF-CMA-Sicherheit lediglich auf der Schwierigkeit des RSA-Problems beruht.

4.5 Offene Probleme

Das Hohenberger-Waters Verfahren ist das erste RSA-basierte Signaturverfahren mit EUF-naCMA-Sicherheitsbeweis im Standardmodell, und damit ein wichtiger Meilenstein. Die Suche nach RSA-basierten Signaturverfahren ist jedoch noch lange nicht abgeschlossen. Das Hohenberger-Waters Verfahren ist nämlich leider sehr ineffizient, da nämlich nicht nur bei der Schlüsselerzeugung, sondern auch bei der Erzeugung und Verifikation von Signaturen viele Primzahlen gefunden werden müssen.[3] Jeder, der schonmal mit PGP/GnuPG einen RSA-Schlüssel erzeugt hat oder mit einem Computeralgebra-Programm wie MAGMA große Primzahlen gesucht hat weiß, wie zeitaufwändig dies sein kann.

Ein wenig effizientere RSA-basierte Signaturen, welche auf den Techniken von [HW09b] aufbauen, finden sich zum Beispiel in [HJK11]. Jedoch sind auch diese Varianten noch nicht effizient genug, um in der Praxis einsetzbar zu sein. Ein *praktisches* RSA-basiertes Signaturverfahren mit EUF-CMA-Sicherheitsbeweis im Standardmodell ist ein sehr wichtiges offenes Problem.

[3] Zum Vergleich, beim wesentlich effizienteren RSA-FDH Verfahren aus Kapitel 4.2 müssen nur bei der (einmaligen) Schlüsselerzeugung zwei große Primzahlen gefunden werden.

Kapitel 5

Pairing-basierte Signaturverfahren

Signaturen, die auf so genannten *Pairings*[1] (bilinearen Abbildungen) basieren, sind neben RSA-basierten Signaturen eine zweite große Klasse von Signaturverfahren. Während Pairings ursprünglich zunächst in der Kryptoanalyse benutzt wurden, insbesondere zur effizienteren Lösung des diskreten Logarithmusproblems [MVO91], hat sich herausgestellt, dass sie auch für die Konstruktion von Kryptosystemen sehr hilfreich sein können. Eine Vielzahl interessanter kryptographischer Primitive wird durch die Existenz von Pairings erst ermöglicht, wie zum Beispiel identitätsbasierte Verschlüsselung [Sha85, BF01, Wat05] oder die nicht-interaktiven Beweissysteme von Groth und Sahai [GS08].

In diesem Kapitel geben wir zunächst eine kurze Einführung in Pairings und zeigen als erste Anwendung das 3-Parteien Schlüsselaustausch-Protokoll von Joux [Jou04]. Dann beschreiben wir das Pairing-basierte Signaturverfahren von Boneh, Lynn und Shacham [BLS01, BLS04]. Dieses sehr simple Verfahren hat zahlreiche nette Eigenschaften, wie zum Beispiel kurze Signaturen, Aggregierbarkeit, Batch-Verifikation und beweisbare EUF-CMA-Sicherheit. Zum Schluss beschreiben wir das Signaturverfahren von Waters [Wat05]. Wir nutzen diese Gelegenheit, um das Konzept von *programmierbaren Hashfunktionen* (PHFs) [HK08] vorzustellen. PHFs sind eine wichtige Abstraktion, die für die Konstruktion von beweisbar sicheren Signaturen und identitätsbasierten Verschlüsselungsverfahren sehr hilfreich ist. Insbesondere wird der EUF-CMA-Sicherheitsbeweis des Waters-Signaturverfahrens durch diese Abstraktion stark vereinfacht.

5.1 Pairings

Definition 78. Seien $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ zyklische Gruppen mit Ordnung p. Ein *Pairing* (manchmal auch treffenderweise „bilineare Abbildung“ genannt) ist eine Abbildung

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

mit den folgenden drei Eigenschaften.

1. **Bilinearität.** Für alle $g_1, g_1' \in \mathbb{G}_1$ und $g_2, g_2' \in \mathbb{G}_2$ gilt

$$e(g_1 g_1', g_2) = e(g_1, g_2) \cdot e(g_1', g_2) \qquad \text{und} \qquad e(g_1, g_2 g_2') = e(g_1, g_2) \cdot e(g_1, g_2').$$

[1] Im Deutschen „Paarung“, aber wir bleiben der Einfachheit halber bei der auch im Deutschen häufig üblichen englischen Bezeichnung.

2. **Nicht-Degeneriertheit.** Für Generatoren $g_1 \in \mathbb{G}_1$ und $g_2 \in \mathbb{G}_2$ ist $e(g_1, g_2)$ ein Generator von $\mathbb{G}_T$. Falls die Gruppen prime Ordnung haben ist dies äquivalent zu

$$e(g_1, g_2) \neq 1.$$

3. **Effiziente Berechenbarkeit.** Die Abbildung e kann effizient berechnet werden.

Excercise 79. Sei $a \in \mathbb{Z}_p$. Zeigen Sie, dass die folgende Gleichung aus der Bilinearität von e folgt:

$$e(g_1^a, g_2) = e(g_1, g_2)^a = e(g_1, g_2^a)$$

Dabei sind die Gruppen $\mathbb{G}_1$ und $\mathbb{G}_2$ in der Regel Elliptische Kurven Gruppen. Wir nennen diese Gruppen auch die „Ursprungs-Gruppen“. Die Gruppe $\mathbb{G}_T$ ist in der Regel eine Untergruppe eines endlichen Körpers $\mathbb{F}_Q$, im Folgenden auch die „Target-Gruppe“ genannt. Wir gehen hier nicht weiter auf die (hochgradig nichttriviale) konkrete Konstruktion von Pairings ein, sondern wollen sie nur als abstrakten Baustein zur Konstruktion digitaler Signaturverfahren verwenden.

Remark 80. Die ursprüngliche Anwendung von Pairings in der Kryptographie lag in der Kryptoanalyse. Menezes, Okamoto und Vanstone [MVO91] hatten beobachtet, dass eine Abbildung $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ dabei helfen kann, das diskrete Logarithmusproblem in $\mathbb{G}$ zu lösen – nämlich dann, wenn das diskrete Logarithmusproblem in $\mathbb{G}_T$ einfacher ist als in $\mathbb{G}$. Zum Glück gibt es jedoch auch Pairings, für die das diskrete Logarithmusproblem in $\mathbb{G}_T$ mindestens genauso schwer ist wie das in $\mathbb{G}$. Diese Pairings kann man für kryptographische Anwendungen benutzen. Siehe zum Beispiel [CF05] für Details.

Typen von Pairings. Es gibt verschiedene Typen von Pairings, die nach [GPS08] in drei Kategorien eingeteilt werden.

- **Typ-1-Pairings.** Dies sind Pairings, bei denen $\mathbb{G}_1$ und $\mathbb{G}_2$ die gleiche Gruppe sind, also $\mathbb{G}_1 = \mathbb{G}_2$. Diese Klasse von Pairings nennt man auch *symmetrische* Pairings.

  Da wir in diesem Falle nicht zwischen den Gruppen $\mathbb{G}_1$ und $\mathbb{G}_2$ unterscheiden müssen, schreiben wir manchmal auch $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.

- **Typ-2-Pairings.** Dies sind asymmetrische Pairings, es gilt also $\mathbb{G}_1 \neq \mathbb{G}_2$. Jedoch existiert ein effizient berechenbarer, nicht-trivialer Homomorphismus $\psi : \mathbb{G}_2 \to \mathbb{G}_1$.

- **Typ-3-Pairings.** Dies sind asymmetrische Pairings, also $\mathbb{G}_1 \neq \mathbb{G}_2$, und es ist kein effizient berechenbarer, nicht-trivialer Homomorphismus $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ bekannt.

Die Eigenschaften dieser Pairings unterscheiden sich zuweilen stark, zum Beispiel hinsichtlich der Größe der Darstellung von Elementen von $\mathbb{G}_1$ oder ob Hashfunktionen bekannt sind, deren Bildmenge der Gruppe $\mathbb{G}_2$ entspricht. Wir verweisen hier auf [GPS08] für einen Überblick.

Im Folgenden betrachten wir der Einfachheit halber vornehmlich Typ-1-Pairings. Viele der vorgestellten Konstruktionen funktionieren jedoch auch mit Typ-2- und Typ-3-Pairings.

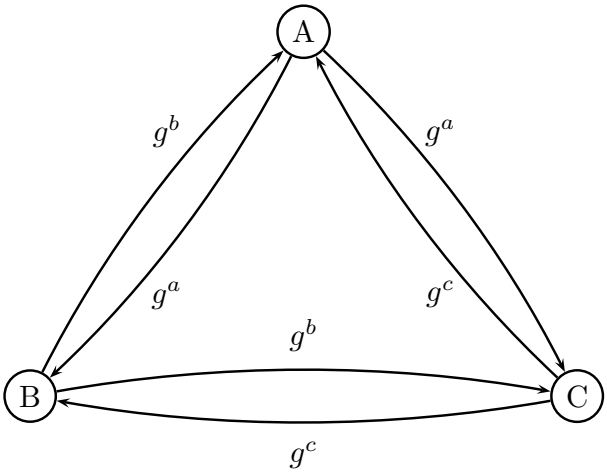

Abbildung 5.1: Nachrichtenaustausch in Joux's 3-Parteien Diffie-Hellman Protokoll. Der gemeinsame Schlüssel k der Parteien A, B und C wird berechnet als $k = e(g, g,)^{abc}$.

Joux's 3-Parteien Diffie-Hellman Protokoll. Als erstes einfaches Beispiel für die Nützlichkeit von Pairings eignet sich das 3-Parteien Diffie-Hellman Protokoll von Joux [Jou04] gut. Durch dieses Protokoll können drei Parteien A, B und C *in nur einer Runde* einen gemeinsamen Schlüssel berechnen.

Sei $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ ein symmetrisches Pairing und g ein Generator der Gruppe $\mathbb{G}$ mit Ordnung p.

- Partei A wählt einen geheimen, zufälligen Wert $a \xleftarrow{\$} \mathbb{Z}_p$. B wählt $b \xleftarrow{\$} \mathbb{Z}_p$, und C wählt $c \xleftarrow{\$} \mathbb{Z}_p$.
- Partei A sendet den Wert $g^a \in \mathbb{G}$ an B und C. Parallel dazu sendet B den Wert g^b an A und C, und C den Wert g^c an A und B (siehe Abbildung 5.1).
- Partei A berechnet den gemeinsamen Schlüssel als $k = e(g^b, g^c)^a$. B berechnet $k = e(g^a, g^c)^b$, und C berechnet $k = e(g^a, g^b)^c$. Die *Correctness* dieses Protokolls folgt aus der Bilinearität von e, denn es gilt

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

  somit berechnen also alle Parteien den gleichen Schlüssel k.

5.2 Boneh-Lynn-Shacham Signaturen

Das Signaturverfahren von Boneh, Lynn und Shacham [BLS01, BLS04] („BLS-Signaturen") ist das einfachste Pairing-basierte Signaturverfahren. Trotzdem bietet es sehr kurze Signaturen, ist sehr effizient, und beweisbar EUF-CMA-sicher (im Random Oracle Modell, vgl. Kapitel 4.2.1). Zusätzlich hat es durch die *Aggregierbarkeit* und *Batch-Verifikation* Eigenschaften, die es interessant machen für Anwendungsszenarien, bei denen die Bandbreite des Übertragungskanal oder die Ressourcen des Verifizierers beschränkt sind.

Seien $\mathbb{G}$ und $\mathbb{G}_T$ zwei Gruppen primer Ordnung p und g ein Generator von $\mathbb{G}$. Sei $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ eine bilineare Abbildung vom Typ 1. Sei $H : \{0,1\}^* \to \mathbb{G}$ eine Hashfunktion, die Bit-Strings auf Gruppenelemente abbildet.

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus wählt $x \xleftarrow{\$} \mathbb{Z}_p$ und berechnet $g^x \in \mathbb{G}$. Der öffentliche Schlüssel ist $pk := (g, g^x)$, der geheime Schlüssel ist $sk := x$.

$\mathsf{Sign}(sk, m)$**.** Um eine Nachricht $m \in \{0,1\}^*$ zu signieren wird $\sigma \in \mathbb{G}$ berechnet als

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

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus gibt 1 aus, wenn

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

gilt, und ansonsten 0.

Correctness. Die *Correctness* folgt wieder aus der Bilinearität des Pairings, denn es gilt

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

Remark 81. Eine bemerkenswerte Eigenschaft des BLS-Verfahrens ist, dass eine Signatur nur aus einem einzigen Gruppenelement besteht. Bei geeigneter Wahl der Gruppe sind so sehr kurze Signaturen möglich.

Remark 82. Das BLS-Signaturverfahren ist sehr eng mit dem identitätsbasierten Verschlüsselungsverfahren (*identity-based encryption, (IBE)* von Boneh und Franklin [BF01] verwandt. Die Erzeugung eines Schlüssels für eine Identität ID in [BF01] entspricht genau der Berechnung einer BLS-Signatur über die „Nachricht“ ID.

Diese Gemeinsamkeit ist kein Zufall. Ein IBE-Verfahren impliziert stets ein Signaturverfahren. Die generische Konstruktion eines Signaturverfahrens aus einem IBE-Verfahren ist auch als *Naor-Transformation* bekannt, siehe [BF01].

Excercise 83. Das BLS-Verfahren kann auch mit asymmetrischen Pairings instantiiert werden. Geben Sie eine Beschreibung basierend auf Typ-2- oder Typ-3-Pairings an.

5.2.1 Das Computational Diffie-Hellman Problem

Die Sicherheit des BLS-Verfahrens basiert auf dem so genannten *Computational Diffie-Hellman* Problem in der Gruppe $\mathbb{G}$.

Definition 84. Sei $\mathbb{G}$ eine Gruppe der Ordnung p, und sei g ein zufälliger Generator von $\mathbb{G}$. Seien $x, y \xleftarrow{\$} \mathbb{Z}_p$. Das *Computational Diffie-Hellman* (CDH) Problem in $\mathbb{G}$ ist:

- Gegeben (g, g^x, g^y),
- berechne g^{xy}.

5.2.2 Sicherheit von BLS-Signaturen

Theorem 85. *Wenn die Hashfunktion H als Random Oracle modelliert wird, dann existiert für jeden PPT-Angreifer $\mathcal{A}$, der die EUF-CMA-Sicherheit des BLS-Signaturverfahrens in Zeit $t_{\mathcal{A}}$ mit Erfolgswahrscheinlichkeit $\epsilon_{\mathcal{A}}$ bricht, ein PPT-Angreifer $\mathcal{B}$, der das CDH-Problem in $\mathbb{G}$ löst in Zeit $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ mit einer Erfolgswahrscheinlichkeit von mindestens*

$$\epsilon_{\mathcal{B}} \geq \frac{\epsilon_{\mathcal{A}}}{q_H},$$

wobei q_H die Anzahl der Anfragen an das Random Oracle sind, die $\mathcal{A}$ im EUF-CMA-Experiment stellt.

Beweisidee. Die Idee des Beweises ist sehr ähnlich zum Sicherheitsbeweis des RSA-FDH Signaturverfahrens aus Kapitel 4.2. Wieder wird die „Programmierbarkeit" des Random Oracles benutzt, um gültige Signaturen zu simulieren, und eine gegebene CDH-Challenge geschickt einzubetten.

Etwas genauer, Algorithmus $\mathcal{B}$ erhält als Eingabe (g, g^x, g^y). $\mathcal{B}$ rät einen Index $\nu \xleftarrow{\$} \{1, \ldots, q_H\}$ und definiert den pk als $pk = (g, g^x)$.

Beantwortung der Random-Oracle-Anfragen. Die i-te Anfrage m_i an das Random Oracle H beantwortet $\mathcal{B}$ wie folgt:

Falls $i \neq \nu$, so wählt $\mathcal{B}$ einen gleichverteilt zufälligen Wert $z_i \xleftarrow{\$} \mathbb{Z}_p$, definiert $H(m_i) := g^{z_i}$, und antwortet mit g^{z_i}. Da $z_i \in \mathbb{Z}_p$ gleichverteilt ist, ist auch g^{z_i} gleichverteilt über $\mathbb{G}$. Somit ist die Antwort des Random Oracles korrekt verteilt.

Falls $i = \nu$, so definiert $\mathcal{B}$ $H(m_\nu) := g^y$, wobei g^y aus der gegebenen CDH-Challenge stammt, und antwortet mit g^y. Auch g^y ist gleichverteilt über $\mathbb{G}$, und somit ist die Antwort des Random Oracles korrekt verteilt. ($\mathcal{B}$„hofft" darauf, dass $\mathcal{A}$ eine Signatur für Nachricht $m^* = m_\nu$ fälscht.)

Beantwortung der Signaturanfragen. Wir nehmen wieder an (wie in Kapitel 4.2), dass $\mathcal{A}$ stets den Hashwert $H(m_i)$ anfragt *bevor* $\mathcal{A}$ eine Signatur für m_i anfragt (ansonsten kann einfach $\mathcal{B}$ die entsprechende Random-Oracle-Anfrage im Namen von $\mathcal{A}$ stellen).

Dann kann $\mathcal{B}$ alle Nachrichten m_i signieren, falls $i \neq \nu$ indem er

$$\sigma_i := (g^x)^{z_i}$$

berechnet. Dies ist eine gültige Signatur, denn es gilt

$$e(g, \sigma_i) = e(g, g^{xz_i}) = e(g, g)^{xz_i} = e(g^x, g^{z_i}) = e(g^x, H(m_i)),$$

die Verifikationsgleichung ist also erfüllt.

Extraktion der CDH-Lösung. Wir nehmen im Folgenden an, dass der Angreifer stets eine Signatur für Nachricht m^* ausgibt, für die er zuvor $H(m^*)$ beim Random Oracle erfragt hat (falls er dies nicht tut, kann der Angreifer nur mit vernachlässigbar kleiner Wahrscheinlichkeit eine Signatur ausgeben).

Falls der Angreifer eine gültige Signatur σ^* für Nachricht $m^* = m_\nu$ ausgibt, so gilt wegen $H(m^*) = H(m_\nu) = g^y$ die Verifikationsgleichung

$$e(g, \sigma^*) = e(g^x, g^y),$$

welche genau dann erfüllt ist, wenn $\sigma^* = g^{xy}$ ist. $\mathcal{B}$ kann also die Signatur σ^* als Lösung der CDH-Challenge ausgeben.

Excercise 86. Geben Sie einen vollständigen, formalen Beweis für Theorem 85 an, basierend auf der oben skizzierten Beweistechnik. (*Hinweis:* Sie können sich am Beweis von Theorem 64 orientieren, der Beweis ist sehr ähnlich.)

5.2.3 Aggregierbarkeit von BLS-Signaturen

Eine interessante Eigenschaft von BLS-Signaturen ist, dass sie *aggregierbar* sind [BGLS03].

Das Problem. Sei $U_1, \ldots, U_n$ eine Liste von Sendern. Dies können zum Beispiel Sensoren in einem Sensor-Netzwerk sein, welches Messdaten an eine Empfangsstation V überträgt. Sei $m_1, \ldots, m_n$ die Liste von Nachrichten, die an V gesendet wird, wobei die Nachricht m_i von Sender U_i stammt. Die Daten werden dann über einen Kanal (mit möglicherweise niedriger Bandbreite) an den Empfänger V übertragen.

Die Nachrichten werden digital signiert. Jeder Sender U_i verfügt dazu über einen öffentlichen Signaturschlüssel pk_i mit dazugehörigem geheimen Schlüssel sk_i. Bei Verwendung eines klassischen Signaturverfahrens müssen n Nachrichten-Signatur-Paare an V gesandt werden,

$$(m_1, \sigma_1), \ldots, (m_n, \sigma_n).$$

Insgesamt müssen also zusätzlich zu den Daten noch n Signaturen übertragen werden.

Die Lösung: Aggregierbarkeit. Falls die Sender $U_1, \ldots, U_n$ das BLS-Signaturverfahren verwenden, ist es jedoch möglich diesen Aufwand stark zu reduzieren. In diesem Falle verfügt jeder Sender über einen öffentlichen BLS-Schlüssel $pk_i = (g, g^{x_i})$ mit geheimem Schlüssel $sk_i = x_i$.[2]

Bevor die signierten Daten an V übertragen werden, werden die Signaturen *aggregiert*. Dazu berechnet jeder Sender U_i zunächst eine Signatur $\sigma_i := H(m_i)^{x_i}$ über Nachricht m_i, und sendet dieses Paar (m_i, σ_i) an einen bestimmten Sender, den *Aggregierer*.

Der Aggregierer berechnet eine *aggregierte Signatur*

$$\sigma := \prod_{i=1}^{n} \sigma_i,$$

und überträgt dann die Liste der Nachrichten $m_1, \ldots, m_n$ zusammen mit der aggregierten Signatur σ an V.

Remark 87. Die aggregierte Signatur kann berechnet werden, ohne dass der Aggregierer dazu ein Geheimnis kennen muss. Die Aggregation ist also eine „öffentliche“ Berechnung.

[2] Alle Sender verwenden den gleichen Generator g. Der Generator kann also auch als *Systemparameter* angesehen werden, und muss dann nicht in jedem öffentlichen Schlüssel enthalten sein.

Der Verifizierer, der die Nachrichten $m_1, \ldots, m_n$ zusammen mit der aggregierten Signatur σ erhält, prüft dann zur Verifikation ob

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

gilt.

Die Sicherheit von aggregierten Signaturen kann wieder unter der CDH-Annahme gezeigt werden (im Random Oracle Modell). Der Beweis ist sehr ähnlich zum Beweis von Theorem 85, siehe auch [BGLS03].

Excercise 88. Zeigen Sie die *Correctness* des aggregierten BLS-Signaturverfahrens.

Remark 89. Es ist bislang kein aggregierbares Signaturverfahren im Standardmodell (also ohne Random Oracles) bekannt. Im Standardmodell kann man bislang nur *sequentiell* aggregierbare Signaturen konstruieren, siehe zum Beispiel [LOS+06].

5.2.4 Batch-Verifikation von BLS-Signaturen

Eine weitere interessante Eigenschaft von BLS-Signaturen ist, dass es möglich ist viele Signaturen „auf einmal“ zu verifizieren („Batch-Verifikation“).

Das Problem. Sei V ein Empfänger, der eine Liste von n Nachrichten-Signatur-Paaren

$$(m_1, \sigma_1), \ldots, (m_n, \sigma_n)$$

von einem Absender U mit öffentlichem Schlüssel pk erhalten hat. Bei Anwendung eines klassischen Signaturverfahrens müsste V jede Signatur einzeln verifizieren, also

$$\mathsf{Vfy}(pk, m_i, \sigma_i) \stackrel{?}{=} 1 \qquad \forall i \in \{1, \ldots, n\}$$

berechnen.

Die Lösung: Batch-Verifikation. Im Falle von BLS-Signaturen kann der Verifizierer jedoch alle Signaturen auf einmal verifizieren. Sei $pk = (g, g^x)$ der öffentliche Schlüssel des Senders U. Der Verifizierer berechnet dazu zunächst die Produkte

$$h := \prod_{i=1}^{n} H(m_i) \qquad \text{und} \qquad \sigma := \prod_{i=1}^{n} \sigma_i$$

und prüft dann, ob

$$\mathsf{Vfy}(pk, h, \sigma) \stackrel{?}{=} 1 \qquad \Longleftrightarrow \qquad e(g, \sigma) \stackrel{?}{=} e(g^x, h)$$

gilt.

Diese Batch-Verifikation ist wesentlich effizienter, da die Auswertung eines Pairings (nach aktuellem Stand) wesentlich teurer ist als eine Multiplikation in $\mathbb{G}$. Batch-Verifikation ist besonders in Szenarien interessant, in denen sehr viele Signaturen eines Absenders in sehr kurzer Zeit authentifiziert werden müssen.

Excercise 90. Zeigen Sie die *Correctness* des Batch-Verifikationsverfahrens für BLS-Signaturen.

Excercise 91. Funktioniert die Batch-Verifikation auch bei aggregierten Signaturen?

5.3 Boneh-Boyen Signatures

Given that BLS signatures could only be proven secure in the Random Oracle Model, it is natural to ask whether it is also possible to construct digital signatures in the pairing-based setting without random oracles. In order to explain a classical technique for constructing such signatures, we now describe a slightly simplified version of the scheme of Boneh and Boyen [BB04b, BB08].

Remark 92. We will describe a variant[3] of the simplified "weakly-secure" scheme from [BB04b], and prove it EUF-naCMA-secure. This weakly-secure scheme already contains the central trick of Boneh and Boyen that we want to consider here. The fully EUF-CMA-secure scheme of [BB04b] is essentially an optimized combination of this weakly-secure scheme with the discrete log-based chameleon hash function from Section 3.3.1, see the paper of Boneh and Boyen [BB04b, BB08] for details.

5.3.1 The Signature Scheme

Let $\mathbb{G}$ and $\mathbb{G}_T$ be prime-order groups, let h be a random generator of $\mathbb{G}$, and let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a Type-1 bilinear map. The scheme has message space $\mathbb{Z}_p$.

$\mathsf{Gen}(1^k)$**.** The key generation algorithm chooses a random exponent $x \xleftarrow{\$} \mathbb{Z}_p$. The public key is $pk := (h, h^x) \in \mathbb{G}^2$, the corresponding secret key is $sk := x \in \mathbb{Z}_p$.

$\mathsf{Sign}(sk, m)$**.** A signature for message $m \in \mathbb{Z}_p$ is computed as follows. If $m = -x$, then set $\sigma := x$. Otherwise, compute

$$\sigma := h^{\frac{1}{x+m}}$$

Return σ.

$\mathsf{Vfy}(pk, m, \sigma)$**.** The verification algorithm outputs 1 if and only if

$$e(h^x \cdot h^m, \sigma) = e(h, h)$$

Remark 93. Note that the above signing algorithm essentially outputs the secret key when given as input a message m such that $m = -x$. We do this to obtain a perfectly-correct signature scheme, however, we will have to deal with this in the security proof.

Excercise 94. Prove that the above signature scheme is *correct*.

5.3.2 Security Analysis

This signature scheme can be proven secure under the following complexity assumption, which is called the *Q-strong Diffie-Hellman assumption*.

Definition 95. Let $\mathbb{G}$ be a group of order p and let $g \in \mathbb{G}$ be a random generator. Let $x \xleftarrow{\$} \mathbb{Z}_p$. The *$Q$-strong Diffie-Hellman problem* in $\mathbb{G}$ is:

- Given $(g, g^x, g^{x^2}, \ldots, g^{x^Q})$

[3] Essentially, the difference is only that we consider the symmetric type-1 bilinear group setting, while Boneh and Boyen considered asymmetric type-2 bilinear groups in their paper.

- Compute $(y, s) \in \mathbb{G} \times \mathbb{Z}_p$ such that $y = g^{\frac{1}{x+s}}$

Theorem 96. *From each adversary $\mathcal{A}$ that breaks the EUF-naCMA-security of the Boneh-Boyen signature scheme by querying for q signatures in the experiment and running in time $t_{\mathcal{A}}$ and with success probability $\epsilon_{\mathcal{A}}$, we can construct an algorithm $\mathcal{B}$ that solves the Q-strong Diffie-Hellman problem with $Q = q + 1$ in time $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ with success probability*

$$\epsilon_{\mathcal{B}} \geq \epsilon_{\mathcal{A}}$$

Beweis. Algorithm $\mathcal{B}$ receives as input a challenge $(g, g^x, g^{x^2}, \ldots, g^{x^{q+1}})$ of the Q-strong Diffie-Hellman problem. It starts algorithm $\mathcal{A}$, which outputs a list of q messages $m_1, \ldots, m_q \in \mathbb{Z}_p$. First, $\mathcal{B}$ checks whether there exists an $i \in \{1, \ldots, q\}$ such that $g^x = g^{-m_i}$. If this holds for some i, then $\mathcal{B}$ is trivially able to solve the Q-strong Diffie-Hellman challenge. Thus, we may henceforth assume that $x \neq -m_i$ for all i.

Simulation, part one: set-up of the public key. Let P be the polynomial $P(X) := \prod_{i=1}^{q}(X + m_i)$. Algorithm $\mathcal{B}$ defines the public key as $pk := (h, u)$, where

$$h := g^{P(x)} \qquad \text{and} \qquad u := h^x = g^{xP(x)}$$

where x is the random value from the Q-strong Diffie-Hellman challenge. The difficulty here is that we have to describe a way how $\mathcal{B}$ is able to compute $h = g^{P(x)}$ and $h^x = g^{xP(x)}$ *without knowing* x. Here we will use the additional terms $g^x, g^{x^2}, \ldots, g^{x^{q+1}}$ from the Q-strong Diffie-Hellman challenge.

To this end, consider the polynomial $P(X)$, which has degree q. We can write this polynomial as

$$P(X) = \prod_{i=1}^{q}(X + m_i) = \sum_{i=0}^{q} a_i X^i \tag{5.1}$$

for values $a_1, \ldots, a_q \in \mathbb{Z}_p$ that are *efficiently computable*, given $m_1, \ldots, m_q$. Thus, algorithm $\mathcal{B}$ is able to compute h and h^x efficiently as follows:

1. First, it computes the values $a_1, \ldots, a_q$ as in Equation (5.1) from $m_1, \ldots, m_q$
2. Then it defines h and u as

$$h := \prod_{i=0}^{q} \left(g^{x^i}\right)^{a_i} \qquad \text{and} \qquad u := \prod_{i=0}^{q} \left(g^{x^{i+1}}\right)^{a_i}$$

   Note that $u = h^x$. Moreover, note that $\mathcal{B}$ can efficiently compute h and u, because the Q-strong Diffie-Hellman challenge contained exactly the required terms g^{x^i} for all $i \in \{0, \ldots, q + 1\}$.

Note that

$$h = \prod_{i=0}^{q} \left(g^{x^i}\right)^{a_i} = \prod_{i=0}^{q} g^{a_i x^i} = g^{\sum_{i=0}^{q} a_i x^i} = g^{P(x)}$$

and similarly $h^x = g^{xP(x)}$, as desired.

To see that this public key is correctly distributed, recall that we have $x \neq -m_i$ for all i, thus x is not a root of P modulo p. Therefore h is a random generator of $\mathbb{G}$, and it holds that $u = h^x$ for uniformly random $x \in \mathbb{Z}_p$, as required.

Simulation, part 2: computing valid signatures. Next, $\mathcal{B}$ has to compute valid signatures for the messages $m_1, \ldots, m_q$ chosen by $\mathcal{A}$. For each $j \in \{1, \ldots, q\}$, the signature σ_j for message m_j is computed as follows.

1. First, $\mathcal{B}$ uses $m_1, \ldots, m_q$ to compute values $b_1, \ldots, b_q \in \mathbb{Z}_p$ that satisfy the equation

$$\prod_{i=1, i \neq j}^{q} (x + m_i) = \sum_{i=0}^{q-1} b_i x^i$$

   (Note that the index i with $i = j$ is removed from the product.)

2. Then it defines σ_j as

$$\sigma_j := \prod_{i=1, i \neq j}^{q} \left(g^{x^i}\right)^{b_i}$$

To verify that indeed σ_j is a valid signature for m_j, observe that it holds that

$$\sigma_j = \prod_{i=0, i \neq j}^{q} \left(g^{x^i}\right)^{b_i} = g^{\sum_{i=0}^{q-1} b_i x^i} = g^{\prod_{i=1, i \neq j}^{q} (x+m_i)} = g^{P(x)/(x+m_j)}$$

which implies

$$e(h^x \cdot h^{m_j}, \sigma_j) = e(h^{x+m_j}, g^{P(x)/(x+m_j)}) = e(h, g^{P(x)}) = e(h, h)$$

Therefore σ_j is a valid signature for m_j with respect to $pk = (h, h^x)$.

Finally, algorithm $\mathcal{B}$ outputs $(pk, \sigma_1, \ldots, \sigma_q)$ to $\mathcal{A}$.

Extraction. Assume that $\mathcal{A}$ outputs a pair (m^*, σ^*) that satisfies $m^* \notin \{m_1, \ldots, m_q\}$ and

$$e(h^x \cdot h^{m^*}, \sigma^*) = e(h, h)$$

By our assumption on $\mathcal{A}$, this happens with probability $\epsilon_{\mathcal{A}}$. Algorithm $\mathcal{B}$ now proceeds as follows to extract a solution to the Q-strong Diffie-Hellman problem.

Note first that $e(h^x \cdot h^{m^*}, \sigma^*) = e(h^x \cdot h^s, \sigma^*) = e(h, h)$ implies that

$$\sigma^* = h^{1/(x+s)} = g^{P(x)/(x+s)}$$

Using long division, we can write $P(x)$ as

$$P(x) = C(x) \cdot (x + s) + c'$$

for some polynomial $C(x) = \sum_{i=0}^{q-1} c_i x^i$ and some constant $c' \in \mathbb{Z}_p$, and the coefficients $c_0, \ldots, c_{q-2}, c' \in \mathbb{Z}_p$ are efficiently computable, given $m_1, \ldots, m_q, m^*$.

Excercise 97. Show that indeed the values $c_0, \ldots, c_{q-2}, c' \in \mathbb{Z}_p$ can efficiently be computed from $m_1, \ldots, m_q, m^*$.

Therefore we have

$$\sigma^* = g^{P(x)/(x+s)} = g^{(C(x)\cdot(x+s)+c')/(x+s)} = g^{C(x)+c'/(x+s)} = g^{\left(\sum_{i=0}^{q-1} c_i x^i\right)+c'/(x+s)}$$
$$= \prod_{i=0}^{q-1} g^{c_i x^i} \cdot g^{c'/(x+s)}$$

Now, using $c_0, \ldots, c_{q-2}, c'$ and again the g^{x^i}-terms from the Q-strong Diffie-Hellman challenge, adversary $\mathcal{B}$ computes

$$y := \left(\sigma^* / \prod_{i=0}^{q-1} g^{c_i x^i}\right)^{1/c'}$$

ans sets $s := m^*$.

Finally, to see that indeed (y, s) is a valid solution to the Q-strong Diffie-Hellman challenge, note that

$$y = \left(\sigma^* / \prod_{i=0}^{q-1} g^{c_i x^i}\right)^{1/c'}$$
$$= \left(\left(\prod_{i=0}^{q-1} g^{c_i x^i} \cdot g^{c'/(x+s)}\right) / \prod_{i=0}^{q-1} g^{c_i x^i}\right)^{1/c'}$$
$$= \left(g^{c'/(x+s)}\right)^{1/c'} = g^{1/(x+s)}$$

□

5.3.3 The fully-secure scheme of Boneh and Boyen

The fully EUF-CMA-secure scheme of Boneh and Boyen works as follows:

Gen(1^k). The key generation algorithm chooses random exponents $x, u \xleftarrow{\$} \mathbb{Z}_p$. The public key is $pk := (h, h^x, h^u) \in \mathbb{G}^2$, the corresponding secret key is $sk := (x, u) \in \mathbb{Z}_p$.

Sign(sk, m). A signature for message $m \in \mathbb{Z}_p$ is computed as follows. If $m = -x$, then set $\sigma := x$. Otherwise, choose $r \xleftarrow{\$} \mathbb{Z}_p$ and compute

$$\sigma' := h^{\frac{1}{x+ur+m}}$$

Return $\sigma = (\sigma', r)$.

Vfy(pk, m, σ). The verification algorithm parses $(\sigma', r) := \sigma$ and outputs 1 if and only if

$$e(h^x \cdot h^m \cdot (h^u)^r, \sigma') = e(h, h)$$

Excercise 98. Prove that for each algorithm $\mathcal{A}$ that breaks the EUF-CMA-security of this scheme by querying the signing-oracle at most q times in time $t_{\mathcal{A}}$ with success probability $\epsilon_{\mathcal{A}}$, there exists an algorithm $\mathcal{B}$ that solves the Q-strong Diffie-Hellman problem with $Q = q + 1$ in time $t_{\mathcal{B}} \approx t_{\mathcal{A}}$ with success probability at least $\epsilon_{\mathcal{A}}$. (You may want to have a look at the papers of Boneh and Boyen [BB04b, BB08] to receive some hints.)

5.3.4 Similarity to GHR Signatures

In a sense, the Boneh and Boyen [BB04b, BB08] can be seen as a pairing-based variant of the strong-RSA-based signature scheme of Gennaro, Halevi and Rabin that were introduced and analyzed in Section 4.3. Note that both the construction and the security analysis share many similarities:

Signatures. Recall that a GHR-signature is defined as

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

where s is some value contained in the public key, and h is a hash function that maps messages to primes.

Similarly, a Boneh-Boyen signature is defined as

$$\sigma = h^{1/f(m)}$$

where h is some value from the public key, and function f is defined by the value h^x contained in the public key as $f(m) = x + m$.

Simulation in the security proof. Recall that in the security proof of GHR-signatures, we have defined the value s contained in the public key as

$$s := y^{\prod_i h(m_i)}$$

Here we used the fact that in the EUF-naCMA security experiment we receive all messages that the attacker wishes to have signed at the very beginning of the experiment, before we have to provide the attacker with the public key. We have used this setup in the proof to simulate signatures, by computing a signature for message m_j as

$$\sigma_i := y^{\prod_{i \neq j} h(m_i)} = s^{1/h(m_j)}$$

Similarly, for Boneh-Boyen signatures we have defined the value h in the public key as

$$h := g^{\prod_i (x+m_i)}$$

and simulated signatures by computing

$$\sigma_i := g^{\prod_{i \neq j} (x+m_i)} = h^{1/h(m_j)}$$

Extraction in the security proof. In the case of GHR-signatures, a successful attacker provided us with a signature

$$\sigma^* = s^{1/h(m^*)} = y^{\left(\prod_i h(m_i)\right)/h(m^*)}$$

and we used Shamir's trick (Lemma 31) to extract $y^{1/h(m^*)}$ from σ^*.

In comparison, in the case of Boneh-Boyen signatures the attacker gives us

$$\sigma^* = h^{1/(x+m^*)} = g^{\left(\prod_i (x+m_i)\right)/(x+m^*)}$$

and we used a different trick to extract $g^{1/(x+m^*)}$ from σ^*.

"Flexible" complexity assumptions. Finally, note that we were able to prove the EUF-naCMA-security of GHR-signatures under the strong-RSA assumption, which states that given (N, y) it is hard to compute a pair (x, e) with $e > 1$ such that $x = y^{1/e} \bmod N$. Note that this is a "flexible" assumption, which asserts that it is hard to compute e-th roots for *any* e, which may be chosen by the adversary (of course e must be non-trivial, hence $e > 1$).

Similarly, we proved the EUF-naCMA-security of Boneh-Boyen signatures under the assumption that, given $g, g^x, \ldots, g^{x^{q+1}}$, it is hard to compute a pair $(g^{1/(x+m^*)}, m^*)$ for any m^*. Again, this is a "flexible" assumption, where the attacker is able to compute an $(x + m^*)$-th root of g for *any* m^*, which again may be chosen by the adversary.

Therefore the Boneh-Boyen signature scheme is sometimes considered as the "pairing-based dual" of GHR-signatures, and the q-strong Diffie-Hellman assumption can be seen as the "pairing-based dual" of the strong-RSA assumption.

5.4 Waters-Signaturen

Ein weiteres wichtiges Signaturverfahren, das auf bilinearen Abbildungen basiert, ist das Verfahren von Waters [Wat05]. Dieses Verfahren basiert, wie das BLS-Verfahren, auf dem CDH-Problem in bilinearen Gruppen, kann jedoch im Standardmodell (ohne Random Oracles) EUF-CMA-sicher bewiesen werden.

Das Waters-Signaturverfahren benutzt (implizit) eine *programmierbare Hashfunktion* (PHF). PHFs sind eine Abstraktion einer häufig benutzten Beweistechnik, welche von Hofheinz und Kiltz [HK08] explizit beschrieben wurde. Diese Abstraktion eignet sich gut um den Beweis von Waters-Signaturen zu verstehen. Daher stellen wir im Folgenden zunächst das Konzept von programmierbaren Hashfunktionen vor, und erklären danach das Waters-Signaturverfahren und seinen Sicherheitsbeweis.

5.4.1 Programmierbare Hashfunktionen

In den Sicherheitsbeweisen von RSA-FDH (Kapitel 4.2) und Boneh-Lynn-Shacham-Signaturen (Kapitel 5.2) haben wir ausgenutzt, dass wir das Random Oracle geeignet „programmieren" konnten, und zwar so, dass wir eine gegebene (RSA/CDH)-Challenge an „genau der richtigen Stelle" in die Simulation des Challengers durch den Algorithmus $\mathcal{B}$ einbetten konnten. *Ist es möglich, dass wir eine konkrete Hashfunktion angeben, die auf eine ähnliche Art und Weise „programmiert" werden kann?*

Im Allgemeinen lautet die Antwort leider „nein", denn Random Oracles existieren nicht. Es ist jedoch möglich eine etwas schwächere Form von Programmierbarkeit zu erreichen, welche ausreicht, um beweisbar sichere Kryptosysteme zu konstruieren.

Gruppen-Hashfunktionen. Programmierbare Hashfunktionen sind Hashfunktionen, die eine Menge auf Gruppenelemente abbilden. Wir definieren daher zunächst Gruppen-Hashfunktionen. Wir betrachten im Folgenden Hashfunktionen mit der Urbildmenge $\{0,1\}^\ell$, wobei $\ell = \ell(k)$ ein Polynom im Sicherheitsparameter ist.

Definition 99. Eine *Gruppen-Hashfunktion* für eine Gruppe $\mathbb{G}$ besteht aus zwei Polynomialzeit-Algorithmen $(\mathsf{Gen}, \mathsf{Eval})$.

- Der *Erzeugungsalgorithmus* $\kappa \xleftarrow{\$} \mathsf{Gen}(g)$ erhält als Eingabe einen Generator g der Gruppe $\mathbb{G}$, und gibt eine Funktionsbeschreibung κ aus. Dieser Algorithmus kann probabilistisch sein.

- Der *Auswertungsalgorithmus* $\mathsf{Eval}(\kappa, m)$ ist ein deterministischer Algorithmus, der eine Hashfunktion
$$H_\kappa : \{0,1\}^\ell \to \mathbb{G},$$
implementiert. Diese Hashfunktion bildet Bit-Strings auf Gruppenelemente ab, und ist parametrisiert durch κ. Gegeben einen Wert $m \in \{0,1\}^\ell$ gibt $\mathsf{Eval}(\kappa, m)$ den Hashwert $H_\kappa(m)$ aus.

Programmierbare Hashfunktionen. Basierend auf Definition 99 können wir nun programmierbare Hashfunktionen (PHFs) definieren, denn PHFs sind Gruppen-Hashfunktionen mit einer besonderen Trapdoor-Eigenschaft.

Definition 100. Eine Gruppen-Hashfunktion $(\mathsf{Gen}, \mathsf{Eval})$ ist eine (v, w, γ)-*programmierbare Hashfunktion*, wenn zusätzlich zwei Polynomialzeit-Algorithmen $(\mathsf{TrapGen}, \mathsf{TrapEval})$ existieren mit den folgenden Eigenschaften.

- Der *Trapdoor-Erzeugungsalgorithmus* $\mathsf{TrapGen}(g, h)$ erhält als Eingabe zwei Generatoren $g, h \in \mathbb{G}$ einer Gruppe der Ordnung p, und gibt eine Funktionsbeschreibung κ und eine Trapdoor τ aus. TrapGen kann probabilistisch sein.

  Wir fordern, dass Funktionsbeschreibungen, die von Gen oder TrapGen erzeugt wurden, perfekt[4] ununterscheidbar sind. Genauer, seien $\kappa_0 \xleftarrow{\$} \mathsf{Gen}(g)$ und $(\kappa_1, \tau) \xleftarrow{\$} \mathsf{TrapGen}(g, h)$. Dann gilt für alle Generatoren g, h, dass κ_0 und κ_1 identisch verteilt sind.

- Der *Trapdoor-Auswertungsalgorithmus* $\mathsf{TrapEval}(\tau, m)$ ist ein deterministischer Algorithmus, der als Eingabe die Trapdoor τ sowie einen Wert $m \in \{0,1\}^\ell$ erhält. Dieser Algorithmus gibt $(a, b) \in \mathbb{Z}_p^2$ aus, sodass für durch $(\kappa, \tau) \xleftarrow{\$} \mathsf{TrapGen}(g, h)$ erzeugte (κ, τ) gilt, dass
$$H_\kappa(m) = h^a g^b.$$

- Die „a-Komponente“ der Ausgabe des TrapEval-Algorithmus ist *wohlverteilt*. Genauer, sei π_a die Projektion $\pi_a(a, b) = a$. Dann muss

  - für alle g, h,
  - für alle κ erzeugt durch $(\kappa, \tau) \xleftarrow{\$} \mathsf{TrapGen}(g, h)$,
  - für alle $m_1^*, \ldots, m_v^* \in \{0,1\}^\ell$ und alle $m_1, \ldots, m_w \in \{0,1\}^\ell$,

  gelten, dass
$$\Pr\left[\begin{array}{c} \pi_a(\mathsf{TrapEval}(\tau, m_i^*)) \equiv 0 \bmod p \quad \forall i \in \{1, \ldots, v\} \\ \wedge \\ \pi_a(\mathsf{TrapEval}(\tau, m_i)) \not\equiv 0 \bmod p \quad \forall i \in \{1, \ldots, w\} \end{array}\right] \geq \gamma,$$

[4] Diese Forderung kann man noch abschwächen. Im Allgemeinen genügt eine *statistisch* ununterscheidbare Verteilung. Für viele Anwendungen reicht auch dass Funktionen, die durch Gen oder TrapGen erzeugt wurden, nicht *effizient* unterschieden werden können. Siehe [HK08] für Details.

wobei die Wahrscheinlichkeit über die Wahl der Trapdoor τ (aus allen möglichen Trapdoors für κ) geht.

Remark 101. Diese Definition ist leider *wesentlich* sperriger als ihre Intuition. Die Idee ist jedoch einfach:

- Der $\mathsf{TrapGen}(g, h)$-Algorithmus soll ermöglichen, dass die Gruppen-Hashfunktion gemeinsam mit einer Trapdoor τ erzeugt werden kann, sodass mit Trapdoor erzeugte Hashfunktionen *ununterscheidbar* sind von ehrlich erzeugten Hashfunktionen.
- Die Trapdoor τ soll es ermöglichen, den *diskreten Logarithmus* (a, b) von Hashwerten bezüglich der Generatoren (g, h) zu berechnen.
- Die geforderte *Wohlverteilung* der a-Komponente stellt die eigentliche (im Vergleich zum Random Oracle natürlich stark eingeschränkte) „Programmierbarkeit“ dar. Eine Hashfunktion ist (v, w, γ)-programmierbar, wenn mit Wahrscheinlichkeit mindestens γ eintritt, dass für *beliebige* vom Angreifer gewählte Werte $m_1^*, \ldots, m_v^*$ und $m_1, \ldots, m_w$ gilt, dass
  - die h-Komponente in allen Hashwerten $H(m_i^*) = h^{a_i^*} g^{b_i^*}$ „nicht enthalten ist“, also $a_i^* \equiv 0 \bmod p$ gilt, für alle $m_1^*, \ldots, m_v^*$,
  - Die h-Komponente in allen Hashwerten $H(m_i) = h^{a_i} g^{b_i}$ „enthalten ist“, also $a_i \not\equiv 0 \bmod p$ gilt, für alle $m_1, \ldots, m_w$.

Remark 102. Wir werden uns später für $(1, q, \gamma)$-programmierbare PHFs interessieren, sodass γ nicht-vernachlässigbar ist. Sei $m_1, \ldots, m_q$ die Liste von Signatur-Anfragen, die der Angreifer im $\mathsf{EUF\text{-}CMA}$-Experiment stellt, und sei m^* die Nachricht für die der Angreifer eine Fälschung ausgibt. Wir werden im Beweis dann darauf hoffen, dass

$$\pi_a(\mathsf{TrapEval}(\tau, m^*)) \equiv 0 \bmod p \quad \wedge \pi_a(\mathsf{TrapEval}(\tau, m_i)) \not\equiv 0 \bmod p \quad \forall i \in \{1, \ldots, q\}$$

gilt. Wenn die PHF $(1, q, \gamma)$-programmierbar ist, dann tritt dies mit Wahrscheinlichkeit mindestens γ ein.

Remark 103. In anderen Kontexten können auch $(v, 1, \gamma)$-programmierbare Hashfunktionen sehr hilfreich sein, mit kleinem v wie zum Beispiel $v = 4$ oder $v = 8$. Beispielsweise in [HJK11] wurden solche PHFs mit dem „Präfix-Trick“ von Hohenberger und Waters (Kapitel 4.4) kombiniert, um effizientere und kürzere RSA-basierte Signaturen zu erhalten. Effiziente $(v, 1, \gamma)$-PHFs für große Werte von v existieren leider nicht, denn sonst könnte man einen Standardmodell-Beweis für RSA-FDH (und BLS-Signaturen) angeben, der dem Unmöglichkeitsergebnis in [DHT12] widerspricht.

Excercise 104. Sei H eine $(1, 1, \gamma)$-programmierbare Hashfunktion für $\mathbb{G}$. Zeigen Sie dass H kollisionsresistent ist (Definition 15), wenn γ nicht vernachlässigbar ist und das diskrete Logarithmusproblem in $\mathbb{G}$ schwer ist. *Tipp:* Nehmen Sie an es gäbe einen Algorithmus $\mathcal{A}_H$, der mit nicht-vernachlässigbarer Wahrscheinlichkeit eine Kollision für H berechnet. Zeigen Sie dass Sie dann mit Hilfe von $\mathcal{A}_H$ einen Algorithmus $\mathcal{A}_{\mathbb{G}}$ konstruieren können, der das diskrete Logarithmusproblem in $\mathbb{G}$ löst, sodass $\mathcal{A}_{\mathbb{G}}$ ungefähr die gleiche Laufzeit wie $\mathcal{A}_H$ hat. Schätzen Sie zum Schluss die Erfolgswahrscheinlichkeit von $\mathcal{A}_{\mathbb{G}}$ ab.

5.4.2 Das Signaturverfahren

Seien $\mathbb{G}$ und $\mathbb{G}_T$ zwei Gruppen primer Ordnung p und g ein Generator von $\mathbb{G}$. Sei $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ eine bilineare Abbildung vom Typ 1. Sei $(\mathsf{Gen}, \mathsf{Eval})$ eine Gruppen-Hashfunktion, die auf $\mathbb{G}$ abbildet.

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus wählt ein zufälliges Gruppenelement $g^\alpha \xleftarrow{\$} \mathbb{G}$ und berechnet $e(g,g)^\alpha = e(g, g^\alpha)$. Dann wird mittels $\kappa \xleftarrow{\$} \mathsf{Gen}(g)$ die Beschreibung einer Gruppen-Hashfunktion $H_\kappa : \{0,1\}^\ell \to \mathbb{G}$ erzeugt. Wir schreiben im Folgenden H als Kurzform für H_κ.

Der öffentliche Schlüssel ist $pk := (g, \kappa, e(g,g)^\alpha)$, der geheime Schlüssel ist $sk := g^\alpha$.

$\mathsf{Sign}(sk, m)$**.** Um eine Nachricht $m \in \{0,1\}^\ell$ zu signieren wird ein zufälliger Wert $r \xleftarrow{\$} \mathbb{Z}_p$ gewählt, und die Werte

$$\sigma_1 := g^r \in \mathbb{G} \qquad \text{und} \qquad \sigma_2 := g^\alpha \cdot H(m)^r \in \mathbb{G}$$

berechnet. Eine Signatur für Nachricht m besteht aus $\sigma := (\sigma_1, \sigma_2) \in \mathbb{G}^2$.

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus gibt 1 aus, wenn

$$e(g,g)^\alpha \cdot e(\sigma_1, H(m)) = e(g, \sigma_2)$$

gilt, und ansonsten 0.

Excercise 105. Zeigen Sie die *Correctness* des Waters-Signaturverfahrens.

Remark 106. Ebenso wie sich das BLS-Signaturverfahren aus dem Algorithmus für die Berechnung von Identitätsschlüsseln des Boneh-Franklin IBE-Verfahrens ergibt (siehe Bemerkung 82), ergibt sich das Waters-Signaturverfahren aus der Identitätsschlüssel-Erzeugung von Waters' IBE-Verfahren [Wat05].

5.4.3 Sicherheit von Waters-Signaturen

Theorem 107. *Sei die Hashfunktion H, die im Waters-Signaturverfahren verwendet wird, eine $(1, q, \gamma)$-PHF. Sei $\mathcal{A}$ ein PPT-Angreifer, der die EUF-CMA-Sicherheit des Waters-Signaturverfahrens in Zeit $t_\mathcal{A}$ mit Erfolgswahrscheinlichkeit $\epsilon_\mathcal{A}$ bricht, mit höchstens q Signatur-Anfragen. Dann existiert ein PPT-Angreifer $\mathcal{B}$, der das CDH-Problem (Definition 84) in $\mathbb{G}$ in Zeit $t_\mathcal{B} \approx t_\mathcal{A}$ mit Erfolgswahrscheinlichkeit*

$$\epsilon_\mathcal{B} \geq \gamma \cdot \epsilon_\mathcal{A}$$

löst.

Beweis. Algorithmus $\mathcal{B}$ erhält als Eingabe eine CDH-Challenge $(g, g^x, g^y) \in \mathbb{G}^3$, wobei $|\mathbb{G}| = p$. $\mathcal{B}$ generiert die Hashfunktion H durch Berechnung von

$$(\kappa, \tau) \xleftarrow{\$} \mathsf{TrapGen}(g, g^x).$$

Der öffentliche Schlüssel wird definiert als $pk := (g, \kappa, e(g^x, g^y))$. Dies ist ein korrekt verteilter öffentlicher Schlüssel. Man beachte, dass der dritte Wert $e(g^x, g^y) = e(g,g)^{xy}$ im *public key*

den geheimen Schlüssel g^α implizit als $g^\alpha := g^{xy}$ definiert. Der Wert g^{xy} ist genau die von $\mathcal{B}$ gesuchte Lösung des CDH-Problems.

$\mathcal{B}$ gibt den pk an $\mathcal{A}$ aus, der nun Signatur-Anfragen für Nachrichten $m_1, \ldots, m_q$ stellen darf. Dabei „hofft“ $\mathcal{B}$ darauf, dass die folgenden zwei Bedingungen erfüllt sind:

- Für jede Nachricht m_i, für die $\mathcal{A}$ eine Signatur anfragt, gilt $\pi_a(\mathsf{TrapEval}(\tau, m_i)) \not\equiv 0 \bmod p$.

  Genau in diesem Falle kann $\mathcal{B}$ nämlich eine gültige Signatur simulieren ohne den geheimen Schlüssel zu kennen, wie wir in Kürze erklären werden.

- Wenn $\mathcal{A}$ eine Fälschung (m^*, σ^*) ausgibt, dann gilt $\pi_a(\mathsf{TrapEval}(\tau, m^*)) \equiv 0 \bmod p$.

  Genau in diesem Falle kann $\mathcal{B}$ nämlich die Lösung g^{xy} des CDH-Problems aus der Fälschung (m^*, σ^*) extrahieren, wie wir ebenfalls erklären werden.

Falls diese Bedingungen erfüllt sind, sagen wir, dass das Ereignis E eingetreten ist. Da H nach Annahme eine $(1, q, \gamma)$-programmierbare Hashfunktion ist, gilt

$$\Pr[E] = \Pr\left[\begin{array}{c} \pi_a(\mathsf{TrapEval}(\tau, m_i)) \not\equiv 0 \bmod p \quad \forall i \in \{1, \ldots, q\} \\ \wedge \\ \pi_a(\mathsf{TrapEval}(\tau, m^*)) \equiv 0 \bmod p \end{array}\right] \geq \gamma.$$

Wir nehmen im Folgenden an, dass Ereignis E eintritt. In diesem Falle kann $\mathcal{B}$ alle Signaturanfragen von $\mathcal{A}$ beantworten, und gleichzeitig aus der Fälschung eine Lösung des CDH-Problems extrahieren.

Simulation von gültigen Signaturen. Wenn $\mathcal{A}$ die i-te Signatur mit Nachricht m_i anfragt, dann benutzt $\mathcal{B}$ den Trapdoor-Evaluationsalgorithmus $(a_i, b_i) = \mathsf{TrapEval}(\tau, m_i)$ von H zur Berechnung von (a_i, b_i) mit

$$H(m_i) = (g^x)^{a_i} \cdot g^{b_i}.$$

Weil Ereignis E eintritt, und somit $a_i \not\equiv 0 \bmod p$ gilt, kann $\mathcal{B}$ dann die Signatur $\sigma_i = (\sigma_{i,1}, \sigma_{i,2})$ wie folgt berechnen.

1. $\mathcal{B}$ wählt einen zufälligen Wert $s_i \xleftarrow{\$} \mathbb{Z}_p$.

2. Der erste Teil der Signatur wird berechnet als $\sigma_{i,1} := (g^y)^{-1/a_i} \cdot g^{s_i}$. Weil $s_i \xleftarrow{\$} \mathbb{Z}_p$ gleichverteilt über $\mathbb{Z}_p$ ist, ist auch $\sigma_{i,1}$ gleichverteilt über $\mathbb{G}$.

   Wenn man $\sigma_{i,1}$ in der Form $\sigma_{i,1} = g^{r_i}$ schreiben will, dann definiert dies den Wert r_i als $r_i = -y/a_i + s_i$.

3. Der zweite Teil der Signatur wird berechnet als $\sigma_{i,2} := (g^x)^{a_i s_i} \cdot (g^y)^{-b_i/a_i} \cdot g^{b_i s_i}$.

   Um zu zeigen, dass dies eine gültige Signatur ist, müssen wir zeigen dass $\sigma_{i,2} = g^{xy} H(m_i)^{r_i}$ gilt, wobei $r_i = -y/a_i + s_i$ ist. Tatsächlich gilt dies:

$$\begin{aligned} g^{xy} H(m_i)^{r_i} &= g^{xy} \left((g^x)^{a_i} g^{b_i}\right)^{r_i} = g^{xy} \left((g^x)^{a_i} g^{b_i}\right)^{-y/a_i + s_i} \\ &= g^{xy} \left(g^{x a_i (-y/a_i + s_i)} g^{b_i (-y/a_i + s_i)}\right) \\ &= g^{xy} \left(g^{-xy} g^{x a_i s_i} g^{b_i (-y/a_i + s_i)}\right) \\ &= g^{x a_i s_i} g^{b_i (-y/a_i + s_i)} = (g^x)^{a_i s_i} (g^y)^{-b_i/a_i} g^{b_i s_i}. \end{aligned}$$

Die so simulierten Signaturen sind also gültig und korrekt verteilt.

Der „Trick“ bei der Simulation ist also, dass alle Werte geschickt so aufgesetzt werden, dass sich der g^{xy}-Term herauskürzt. Dieser Trick geht zurück auf Boneh und Boyen [BB04a].

Extraktion der Lösung des CDH-Problems. Wenn $\mathcal{A}$ eine gültige Fälschung (m^*, σ^*) ausgibt, $\sigma^* = (\sigma_1^*, \sigma_2^*)$, dann benutzt $\mathcal{B}$ den Trapdoor-Evaluationsalgorithmus $(a^*, b^*) = \mathsf{TrapEval}(\tau, m^*)$ von H zur Berechnung von (a^*, b^*) mit

$$H(m^*) = (g^x)^{a^*} g^{b^*} = (g^x)^0 g^{b^*} = g^{b^*},$$

wobei $a^* \equiv 0 \bmod p$ ist, weil nach Annahme Ereignis E eintritt.

Damit kann $\mathcal{B}$ den gesuchten Wert g^{xy} berechnen als $g^{xy} = \sigma_2^* \cdot (\sigma_1^*)^{-b^*}$. Um zu sehen, dass dies tatsächlich der gesuchte Wert g^{xy} ist, schreiben wir

$$\sigma_1^* = g^{r^*} \qquad \text{und} \qquad \sigma_2^* = g^{xy} \cdot H(m^*)^{r^*}.$$

Dies ist möglich, da σ^* eine gültige Signatur ist. Dann gilt

$$\sigma_2^* \cdot (\sigma_1^*)^{-b^*} = g^{xy} \cdot H(m^*)^{r^*} \cdot g^{-r^* b^*} = g^{xy} \cdot g^{b^* r^*} \cdot g^{-r^* b^*} = g^{xy}.$$

Analyse. Falls also Ereignis E eintritt (was aufgrund der Eigenschaften der PHF H für *beliebige* vom Angreifer $\mathcal{A}$ gewählte Nachrichten $m^*, m_1, \ldots, m_q$ mit Wahrscheinlichkeit mindestens γ passiert), so simuliert $\mathcal{B}$ das echte EUF-CMA-Experiment perfekt. In diesem Falle kann $\mathcal{B}$ das CDH-Problem lösen, wenn $\mathcal{A}$ eine gültige Fälschung ausgibt, und es gilt

$$\epsilon_{\mathcal{B}} \geq \Pr[E] \cdot \epsilon_{\mathcal{A}} \geq \gamma \cdot \epsilon_{\mathcal{A}}.$$

□

5.4.4 Die programmierbare Hashfunktion von Waters

Waters [Wat05] hat die folgende konkrete Konstruktion einer Hashfunktion (die „Waters-Hashfunktion“) angegeben.

- Der Erzeugungsalgorithmus $\mathsf{Gen}(g)$ erhält als Eingabe einen Generator $g \in \mathbb{G}$. Er wählt $\ell + 1$ zufällige Gruppenelemente $u_0, \ldots, u_\ell \xleftarrow{\$} \mathbb{G}$ und gibt $\kappa = (u_0, \ldots, u_\ell)$ aus.
- Der Auswertungsalgorithmus Eval erhält als Eingabe κ und $m = (m_1, \ldots, m_\ell) \in \{0,1\}^\ell$ und gibt

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

  aus.

Theorem 108 ([HK08, Theorem 4]). *Sei $q = q(k)$ ein Polynom im Sicherheitsparameter. Die Waters-Hashfunktion ist eine $(1, q, \gamma)$-programmierbare Hashfunktion mit*

$$\gamma \geq \frac{1}{8(\ell+1)q}.$$

Beweisidee. Es müssen die Algorithmen TrapGen und TrapEval beschrieben und analysiert werden.

- Der TrapGen Algorithmus erhält als Eingabe zwei Generatoren (g, h) und wählt

$$\hat{a}_0, \ldots, \hat{a}_\ell \xleftarrow{\$} \{-1, 0, 1\} \qquad \text{und} \qquad \hat{b}_0, \ldots, \hat{b}_\ell \xleftarrow{\$} \mathbb{Z}_p$$

  gleichverteilt zufällig.

  Der Vektor $\kappa = (u_0, \ldots, u_\ell)$ wird definiert als $u_i := g^{\hat{b}_i} h^{\hat{a}_i}$ für alle $i \in \{0, \ldots, \ell\}$. Da alle $\hat{b}_i$-Werte gleichverteilt zufällig aus $\mathbb{Z}_p$ gewählt wurden, sind alle u_i-Werte gleichverteilt über $\mathbb{G}$. Somit ist κ ununterscheidbar von einer mit Gen erzeugten Beschreibung der Hashfunktion.

  Die Trapdoor $\tau = (\hat{a}_0, \ldots, \hat{a}_\ell, \hat{b}_0, \ldots, \hat{b}_\ell)$ besteht aus den diskreten Logarithmen der u_i-Elemente zur Basis (g, h).

- Der TrapEval-Algorithmus erhält als Eingabe τ und $m = (m_1, \ldots, m_\ell) \in \{0,1\}^\ell$. Er berechnet $(a, b) \in \mathbb{Z}_p$ als

$$a := \hat{a}_0 + \sum_{i=1}^{\ell} \hat{a}_i \cdot m_i \qquad \text{und} \qquad b := \hat{b}_0 + \sum_{i=1}^{\ell} \hat{b}_i \cdot m_i$$

  und gibt (a, b) aus. Es ist leicht zu verifizieren, dass

$$H_\kappa(m) = u_0 \prod_{i=1}^{\ell} u_i^{m_i} = g^{\hat{b}_0 + \sum_{i=1}^{\ell} \hat{b}_i \cdot m_i} h^{\hat{a}_0 + \sum_{i=1}^{\ell} \hat{a}_i \cdot m_i} = g^b h^a$$

  gilt.

- Die Analyse der Wohlverteilung der „a-Komponente“ ist sehr technisch. Daher geben wir hier nur die Intuition an, und verweisen auf [Wat05, HK08, HJK12] für Details.

  Bei der Analyse wird ausgenutzt, dass die $\hat{a}_i$-Werte zufällig aus $\{-1, 0, 1\}$ gewählt wurden. Die Summe

$$a := \hat{a}_0 + \sum_{i=1}^{\ell} \hat{a}_i \cdot m_i$$

  entspricht damit einem *Random Walk* der Länge ℓ über $\{-1, 0, 1\}$. Die Tatsache, dass der Wert a (diskret) Gauß-verteilt in der Nähe der Null ist, wird dann ausgenutzt um die Wohlverteilung der „a-Komponente“ zu beweisen.[5]

5.5 Offene Probleme

Die von Waters angegebene programmierbare Hashfunktion ist das einzige bislang bekannte Beispiel für eine $(1, q, \gamma)$-PHF, wobei $q = q(k)$ ein Polynom im Sicherheitsparameter sein

[5] Tatsächlich werden die $\hat{a}_i$ Werte im Beweis aus [HK08] nicht aus $\{-1, 0, 1\}$ gewählt, sondern diskret normalverteilt (mit Erwartungswert 0 und einer von q abhängenden Varianz). Um die Intuition des Beweises zu verstehen genügt es jedoch sich hier $\{-1, 0, 1\}$ vorzustellen, siehe [HK08] für Details.

kann und trotzdem γ nicht vernachlässigbar ist. Die Waters-Hashfunktion wird durch $\ell + 1$ Gruppenelemente beschrieben, wobei ℓ die Länge der zu signierenden Nachrichten ist. Selbst wenn man eine kollisionsresistente Hashfunktion anwendet, um große Nachrichten zu signieren, wird sich ℓ immernoch in der Größenordnung von mindestens $\ell \approx 160$ bewegen. Das ist sehr groß. Eine effizientere Konstruktion (oder ein Unmöglichkeitsbeweis, dass es keine wesentlich effizientere Konstruktion gibt) einer $(1, q, \gamma)$-PHF ist ein wichtiges offenes Problem.

Die Konstruktion eines effizienten Signaturverfahrens (mit konstanter Größe des öffentlichen Schlüssels *und* der Signaturen), das auf der Schwierigkeit des CDH-Problems basiert, ist ebenfalls ein wichtiges offenes Problem. Bislang ist ein *zustandsbehaftetes* CDH-basiertes Signaturverfahren mit diesen Eigenschaften bekannt [HW09a]. Außerdem wurden asymptotisch effizientere Verfahren in [BHJ+13] beschrieben.

Kapitel 6

Ausgewählte praktische Signaturverfahren und Angriffe

Das Ziel dieses Kapitels ist, einige praxisrelevante Signaturverfahren vorzustellen, und gleichzeitig einen Überblick über verschiedene Angriffe zu geben. Diese Angriffe basieren auf (typischen) Implementierungsfehlern, sie brechen die Sicherheit der Signaturverfahren also nicht „kryptographisch“ (durch Kryptoanalyse). Insbesondere können sie in der Praxis durch korrekte Implementierung verhindert werden. Es sind jedoch gute Beispiele dafür, wie wichtig scheinbare Details in der Praxis sein können.

Zunächst beschreiben wir das Signatur-Verfahren von Schnorr [Sch90, Sch91]. Dieses Verfahren ist sehr wichtig, da es die Grundlage von wichtigen standardisierten Signaturverfahren wie DSA und ECDSA [DSS09] bildet. Schnorr-Signaturen sind probabilistisch, bei der Signaturerstellung wird ein Zufallswert gewählt. Wir werden einen Angriff beschreiben, der es erlaubt den geheimen Schlüssel zu rekonstruieren, falls hierfür ein schwacher Zufallsgenerator benutzt wird.

Danach beschreiben wir das DSA-Signaturverfahren. Dieses Verfahren ist Teil des NIST-Standards DSS [DSS09] und wird in der Praxis sehr häufig verwendet, zum Beispiel in OpenPGP. Wir beschreiben den Angriff von Klíma und Rosa [KR02], bei dem ein Angreifer zunächst den *öffentlichen* Schlüssel geschickt modifiziert, und so nach Erhalt einer einzigen gültigen Signatur den geheimen Schlüssel des Opfers lernen kann.

Zum Schluss beschreiben wir das in der Praxis ebenfalls häufig eingesetzte Signaturverfahren RSA-PKCS#1 v1.5. Ein Angriff von Bleichenbacher, den wir ebenfalls beschreiben werden, erlaubt es Signaturen zu fälschen wenn ein kleiner öffentlicher RSA-Exponent (z.B. $e = 3$) eingesetzt wird, und die Implementierung des Verifikationsalgorithmus die Länge der gegebenen Signatur nicht genau prüft.[1]

6.1 Schnorr-Signaturen

In diesem Kapitel stellen wir das Schnorr-Signaturverfahren [Sch90, Sch91] vor. Dieses Verfahren stellt (in gewisser Hinsicht) die Grundlage wichtiger praktischer Signaturverfahren, wie DSA und ECDSA [DSS09] dar.

[1] Dieser Angriff ist nicht zu verwechseln mit dem Bleichenbacher-Angriff auf RSA-PKCS#1 v1.5 *Verschlüsselung*.

Remark 109. Schnorr-Signaturen sind darüber hinaus auch von einem theoretischen Standpunkt aus sehr interessant. Es handelt sich dabei nämlich um ein Beispiel für ein Signaturverfahren, das mittels der Fiat-Shamir Heuristik [FS87] aus einem Identifikationsverfahren (einem Sigma-Protokoll [Cra96]) erzeugt wurde. Die generische Konstruktion von Signaturverfahren aus Sigma-Protokollen (im Random Oracle Modell) ist ein spannendes Thema, auf das wir an dieser Stelle jedoch nicht näher eingehen.[2]

Statt dessen wollen wir in diesem Kapitel Schnorr's Signaturverfahren als ein Beispiel nehmen, das zeigt, dass „gute" Zufallszahlen nicht nur bei der Erzeugung kryptographischer Schlüssel wichtig sind, sondern manchmal auch bei der Berechnung von digitalen Signaturen.

6.1.1 Das Signaturverfahren

Sei im Folgenden $\mathbb{G}$ eine Gruppe primer Ordnung p mit Generator g und $H : \{0,1\}^* \to \mathbb{Z}_p$ eine kryptographische Hashfunktion.

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus wählt einen Zufallswert $x \xleftarrow{\$} \mathbb{Z}_p$ und berechnet $y := g^x$. Das erzeugte Schlüsselpaar ist (pk, sk) mit

$$pk = (g, y) \qquad \text{und} \qquad sk = x.$$

$\mathsf{Sign}(sk, m)$**.** Um eine Nachricht $m \in \{0,1\}^*$ zu signieren, wird ein Zufallswert $r \xleftarrow{\$} \mathbb{Z}_p$ gewählt und die Werte

$$t := g^r \in \mathbb{G}, \qquad c := H(t||m) \in \mathbb{Z}_p, \qquad s := cx + r \in \mathbb{Z}_p$$

berechnet. Die Signatur ist $\sigma := (t, s) \in \mathbb{G} \times \mathbb{Z}_p$.

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus gibt 1 aus, wenn die Gleichung

$$g^s \stackrel{?}{=} y^{H(t||m)} \cdot t$$

gilt, und ansonsten 0.

Excercise 110. Zeigen Sie die *Correctness* dieses Verfahrens.

Man kann zeigen, dass dieses Signaturverfahren sicher ist, wenn man annimmt, dass das diskrete Logarithmusproblem (Definition 27) in $\mathbb{G}$ schwer ist, und die Hashfunktion H als Random Oracle modelliert wird [PS96].

6.1.2 Warum gute Zufallszahlen wichtig sind

In diesem Kapitel wollen wir ein Beispiel zeigen, warum gute Zufallszahlen nicht nur bei der Erzeugung von Schlüsseln wichtig sind, sondern manchmal auch bei der Erstellung von Signaturen.

Bei der Erstellung von Schnorr-Signaturen wird ein Zufallswert $r \xleftarrow{\$} \mathbb{Z}_p$ gewählt. Man könnte auf die Idee kommen, dass es nicht besonders schlimm ist, wenn sich dieser Wert r für zwei Signaturen wiederholt. Dies könnte zum Beispiel aufgrund eines schlechten Zufallszahlengenerators passieren. Eine andere Möglichkeit ist auch, dass die Person, die das Verfahren implementiert hat, aus Effizienzgründen die wiederholte Berechnung von $t = g^r$ ersparen wollte, und deshalb stets das gleiche Paar $(r, t = g^r)$ verwendet.

[2] Bei Interesse kann man jedoch eine hervorragende Einführung in das Thema im Vorlesungsskript *Cryptologic Protocol Theory* [DN10] von Ivan Damgård und Jesper Buus Nielsen finden.

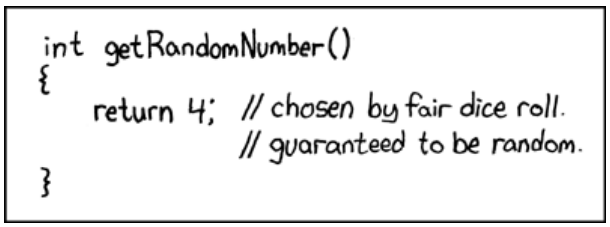

Abbildung 6.1: XKCD Comic Nr. 221, http://xkcd.com/221/.

Der Angriff. Sei $(g, y) = (g, g^x)$ ein öffentlicher Schlüssel des Schnorr-Verfahrens. Angenommen wir erhalten zwei gültige Signaturen $\sigma_1 = (t_1, s_1)$ und $\sigma_2 = (t_2, s_2)$ für zwei Nachrichten $m_1 \neq m_2$, sodass $t_1 = t_2$ gilt. Sei $r = \log_g t_1 = \log_g t_2$ der diskrete Logarithmus von $t_1 = t_2 =: t$ zur Basis g. Dann erhalten wir durch s_1 und s_2 zwei Gleichungen der Form

$$s_1 \equiv x \cdot H(t||m_1) + r \bmod p \qquad \text{und} \qquad s_2 \equiv x \cdot H(t||m_2) + r \bmod p.$$

Wenn wir diese voneinander abziehen, erhalten wir

$$s_1 - s_2 \equiv x \cdot H(t||m_1) - x \cdot H(t||m_2) \bmod p.$$

Falls $H(t||m_1) \neq H(t||m_2)$ gilt (weil $m_1 \neq m_2$ ist, gilt dies für eine kollisionsresistente Hashfunktion H mit einer Wahrscheinlichkeit von nahezu 1), so gilt $H(t||m_1) - H(t||m_2) \not\equiv 0 \bmod p$. Wir können also aus (s_1, s_2) den geheimen Schlüssel x berechnen durch

$$x \equiv \frac{s_1 - s_2}{H(t||m_1) - H(t||m_2)} \bmod p.$$

Remark 111. Der hier beschriebene Angriff widerspricht natürlich nicht dem Sicherheitsbeweis des Schnorr-Signaturverfahrens [PS96], da im Beweis angenommen wird, dass gute Zufallszahlen verwendet werden. Er stellt daher auch keine Schwäche des Schnorr-Verfahrens dar, sondern soll lediglich illustrieren, dass bei der Implementierung auch darauf geachtet werden muss, dass tatsächlich gute Zufallszahlen gewählt werden (vgl. Abbildung 6.1).

Remark 112. Die Wichtigkeit guter Zufallszahlen wird manchmal unterschätzt. Dies zeigt auch der „Debian PRNG-Bug“ [Deb08]. Hier hatte ein Programmierer den Quellcode des Zufallszahlengenerators *(pseudorandom number generator, PRNG)* von OpenSSL in Debian abgeändert, um eine Warnmeldung beim Übersetzen zu „reparieren“. Die Folge dieser Änderung war, dass der vom Zufallszahlengenerator ausgegebene Wert nicht mehr von einem zufälligen *Seed*, sondern nur noch von der Prozess-ID des Betriebssystem-Prozesses abhing. Bei Linux gibt es maximal 32.768 davon.

Dieser Fehler führte dazu, dass zwischen September 2006 und Mai 2008 alle aktuellen Debian-Systeme nur 32.768 verschiedene Schlüssel generieren konnten. Dies betraf Anwendungen wie SSH, OpenVPN, DNSSEC und X.509-Zertifikate, sowie Sitzungsschlüssel für TLS-Verbindungen.

6.2 Der *Digital Signature Algorithm* (DSA)

Der *Digital Signature Algorithm* (DSA) ist ein Signaturverfahren, dessen Sicherheit auf der Schwierigkeit des diskreten Logarithmusproblems in Z_P^* beruht. Er ist Teil des NIST Digital

Signature Standard (DSS) [DSS09], welcher zusätzlich noch die Elliptische-Kurven-Variante ECDSA sowie ein RSA-basiertes Signaturverfahren spezifiziert. Das DSA-Verfahren wird in der Praxis sehr häufig eingesetzt, zum Beispiel in OpenPGP.

6.2.1 Das Signaturverfahren

Sei im Folgenden $H : \bigcup_{i=1}^{64} \{0,1\}^{2^i-1} \to \{0,1\}^{160}$ die SHA-1 Hashfunktion. Das DSA-Verfahren funktioniert wie folgt.

$\mathsf{Gen}()$**.** Der Schlüsselerzeugungsalgorithmus bestimmt eine 1024-Bit Primzahl P, sodass es eine 160-Bit Primzahl p gibt die $P-1$ teilt. Dann wird ein Generator $h \in \mathbb{Z}_P^*$ gewählt, sodass

$$g := h^{\frac{P-1}{p}} \not\equiv 1 \bmod P$$

gilt.[3] Dann wird $x \xleftarrow{\$} \mathbb{Z}_p$ gewählt und $y := g^x \bmod P$ berechnet. Das erzeugte Schlüsselpaar ist (pk, sk) mit

$$pk = (P, p, g, y) \qquad \text{und} \qquad sk = x.$$

$\mathsf{Sign}(sk, m)$**.** Um eine Nachricht $m \in \{0,1\}^{2^{64}-1}$ zu signieren, wird ein Zufallswert $r \xleftarrow{\$} \mathbb{Z}_p$ gewählt und

$$t := (g^r \bmod P) \bmod p \qquad \text{und} \qquad s := (H(m) + xt) \cdot r^{-1} \bmod p$$

berechnet. Die Signatur ist $\sigma := (t, s) \in \mathbb{Z}_p$.

$\mathsf{Vfy}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus berechnet $w := s^{-1} \bmod p$ und gibt 1 aus, wenn die Gleichung

$$t \stackrel{?}{=} \left(\left(g^{H(m)\cdot w} \cdot y^{t\cdot w}\right) \bmod P\right) \bmod p$$

gilt, und ansonsten 0.

Excercise 113. Zeigen Sie die *Correctness* des DSA-Verfahrens.

Es gibt einige Arbeiten, welche die beweisbare Sicherheit des DSA-Verfahrens analysieren [BPVY00, Bro02], jedoch betrachten diese entweder sehr eingeschränkte Angreifermodelle wie das *generische Gruppen Modell* [Sho97] oder modifizierte Varianten des DSA-Verfahrens. Ein Sicherheitsbeweis im Standardmodell oder dem Random Oracle Modell ist nicht bekannt – jedoch auch keine Angriffe.

6.2.2 Der Angriff von Klíma und Rosa

In diesem Kapitel beschreiben wir den Angriff von Klíma und Rosa [KR02] auf die DSA-Implementierung in OpenPGP. Es handelt sich dabei um einen Angriff auf die *Implementierung* des DSA-Verfahrens, nicht um einen kryptographischen Angriff auf das Signaturverfahren.

Remark 114. Der in diesem Kapitel beschriebene Angriff ist nicht zu verwechseln mit dem Angriff von Klíma, Pokorný und Rosa auf OpenSSL [KPR03]. In [KPR03] zeigen die Autoren eine Verfeinerung des Bleichenbacher-Angriffs [Ble98] auf PKCS#1 v1.5 Verschlüsselung.

[3] Anders ausgedrückt: g ist ein Generator der Untergruppe von $\mathbb{Z}_P^*$ mit Ordnung p.

Speicherung von DSA-Schlüsseln auf der Festplatte bei OpenPGP. Ein Schlüsselpaar wird in OpenPGP in einer Datei `sekring.skr` gespeichert. Diese Datei enthält den öffentlichen Schlüssel pk sowie den geheimen Schlüssel sk. Der geheime Schlüssel wird verschlüsselt gespeichert und ist nur über die Eingabe einer Passphrase zugänglich. Der öffentliche Schlüssel wird jedoch im Klartext gespeichert. Darüber hinaus enthält die Datei `sekring.skr` einige Metadaten, wie zum Beispiel Versionsnummern und (nicht-kryptographische) Prüfsummen.

Die Tatsache, dass der öffentliche Schlüssel pk nicht verschlüsselt gespeichert ist sollte kein Problem darstellen – denn der Schlüssel ist ja öffentlich! Klíma und Rosa beobachteten jedoch, dass es in bestimmten Fällen ein Problem sein kann, dass der pk nicht *authentifiziert* gespeichert wird.

Das Angreifermodell von Klíma und Rosa. Klíma und Rosa nehmen an, dass der Angreifer gelegentlich Schreibzugriff auf die Datei `sekring.skr` hat. Dies kann zum Beispiel der Fall sein, wenn der Angreifer gelegentlich Zugang zum Computer des Opfers hat, oder die Schlüssel auf einem Netzlaufwerk oder in einer Cloud gespeichert sind, sodass der Angreifer Zugriff auf die Datei hat.

Vorgehen des Angreifers. Ein Angreifer $\mathcal{A}$, der Schreibzugriff auf die Datei `sekring.skr` hat, kann daher sein Opfer dazu bringen seinen geheimen Schlüssel sk ungewollt zu verraten. Dazu geht $\mathcal{A}$ so vor:

1. $\mathcal{A}$ modifiziert den pk in der Datei `sekring.skr` des Opfers auf geschickte Art und Weise. Er ersetzt den pk also durch einen modifizierten pk'.
2. $\mathcal{A}$ provoziert das Opfer dazu eine Signatur für eine beliebige Nachricht auszustellen, oder wartet einfach darauf bis das Opfer irgendeine Nachricht signiert. Diese Signatur fängt der Angreifer ab.
3. Aus der abgefangenen Signatur berechnet $\mathcal{A}$ den geheimen Schlüssel des Opfers.
4. Zum Schluss macht $\mathcal{A}$ die Änderung des öffentlichen Schlüssels wieder rückgängig. Er ersetzt also pk' wieder durch pk.

Schritt 1: Ersetzen des pk durch pk' . Sei $pk = (P, p, g, y)$ der öffentliche Schlüssel des Opfers. $\mathcal{A}$ wählt eine 159-Bit Primzahl P' sowie einen Generator $g' \in \mathbb{Z}^*_{P'}$. Der modifizierte öffentliche Schlüssel pk' ist

$$pk' := (P', p, g', y).$$

Schritt 2: Das Opfer erstellt eine Signatur unter Benutzung der Parameter aus pk'. Das Opfer wählt einen Zufallswert $r \xleftarrow{\$} \mathbb{Z}_p$ und berechnet zwei Werte (s, t) sodass

$$t := (g'^r \bmod P') \bmod p \qquad \text{und} \qquad s := (H(m) + xt) \cdot r^{-1} \bmod p.$$

Hier liegt die Idee von Klíma und Rosa: Der Angreifer hat P' als eine 159-Bit Primzahl gewählt. Die Primzahl p ist 160-Bit groß, also gilt $P' < p$. Daher wird die „äußere“ Modulo-Reduktion bei der Berechnung von t nicht mehr ausgeführt. Wir können t also auch schreiben als

$$t := g'^r \bmod P'.$$

Schritt 3: Der Angreifer berechnet den geheimen Schlüssel. Der Angreifer erhält vom Opfer eine gültige Signatur (s, t), die mit den Parametern aus dem modifizierten Schlüssel pk' berechnet wurde. Es gilt also

$$t := g'^{r} \bmod P' \qquad \text{und} \qquad s := (H(m) + xt) \cdot r^{-1} \bmod p.$$

Nun berechnet der Angreifer zuerst r, indem er den diskreten Logarithmus von t zur Basis g' modulo P' berechnet. Dies geht relativ schnell, denn P' ist nur 159-Bit groß. Mit Hilfe von r kann er nun (mit hoher Wahrscheinlichkeit) den geheimen Schlüssel x aus s berechnen. Falls $t \not\equiv 0 \bmod p$, was nur mit sehr hoher Wahrscheinlichkeit passiert, so gilt

$$s \equiv (H(m) + xt) \cdot r^{-1} \bmod p \iff x \equiv \frac{s \cdot r - H(m)}{t} \bmod p.$$

Schritt 4: Ersetzen von pk' durch pk . Zum Schluss ersetzt der Angreifer wieder pk' durch den ursprünglichen öffentlichen Schlüssel pk.

Remark 115. Es ist nicht bekannt, ob dieser Angriff jemals in der Praxis eingesetzt wurde. Er zeigt jedoch, dass bei der Implementierung kryptographischer Verfahren auch darauf geachtet werden muss, dass die Integrität von Parametern stets sichergestellt ist.

Excercise 116. Überlegen Sie sich einige Gegenmaßnahmen, die den Angriff von Klíma und Rosa verhindern können.

6.3 RSA-PKCS#1 v1.5 Signaturen

Der RSA-PKCS#1 Standard beschreibt RSA-basierte Verschlüsselungs- und Signaturverfahren. Die verschiedenen Versionen von RSA-PKCS#1 sind auch in Form von RFCs von der IETF standardisiert, und werden in der Praxis häufig verwendet. Die am weitesten verbreitete Version ist die mittlerweile eigentlich obsolete Version 1.5 [Kal98], welche inzwischen durch Versionen 2.0 [KS98] und 2.1 [JK03] aktualisiert wurde.

Das Signaturverfahren, das wir im Folgenden betrachten, stammt aus Version 1.5. Es ist jedoch auch in den aktualisierten Versionen 2.0 und 2.1 noch enthalten. Die aktuellste Version 2.1 beschreibt zusätzlich ein weiteres RSA-basiertes Signaturverfahren, RSA-PSS.

6.3.1 Das Signaturverfahren

Das Signaturschema ist eigentlich lediglich ein Padding-Verfahren, welches auf die Nachricht angewendet wird. Die gepaddete Nachricht wird dann mit dem „Lehrbuch“-RSA Verfahren signiert.

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus ist identisch zur Schlüsselerzeugung beim „Lehrbuch“-RSA Verfahren. Es wird ein ℓ-Bit[4] RSA-Modulus $N = PQ$ erzeugt, wobei P und Q zwei zufällig gewählte Primzahlen sind. Dann wird eine Zahl $e \in \mathbb{N}$ mit $e \neq 1$ und $\mathrm{ggT}(e, \phi(N)) = 1$ gewählt,[5] und der Wert $d := e^{-1} \bmod \phi(N)$ berechnet.

[4] ℓ ist stets ein Vielfaches von 8, in der Praxis übliche Werte sind $\ell \in \{1024, 2048, 4096\}$. Das BSI empfiehlt $\ell \geq 2048$ [BSI08].

[5] Aus Effizienzgründen wird e in der Praxis häufig klein gewählt, zum Beispiel $e = 65536 = 2^{16} + 1$. Ein zu kleiner RSA-Exponent, wie $e = 3$, sollte jedoch nicht verwendet werden, wie wir auch später noch sehen werden.

Der öffentliche Schlüssel ist $pk := (N, e)$, der geheime Schlüssel ist $sk := d$.

Sign(sk, m). Um eine Nachricht m zu signieren wird zunächst der Hashwert $H(m) \in \{0,1\}^n$ berechnet.[6] Da die Bit-Länge n der Ausgabe der Hashfunktion wesentlich kürzer ist als die Länge ℓ des Modulus, wird der Hashwert $H(m)$ nun auf die Länge ℓ gepaddet:

$$M := \texttt{0x00}||\texttt{0x01}||\texttt{0xFF}||\ldots||\texttt{0xFF}||\texttt{0x00}||\mathsf{ASN.1}||H(m).$$

Dabei ist ASN.1 ein ASN.1-code, der die verwendete Hashfunktion H identifiziert. Die Signatur $\sigma \in \mathbb{Z}_N$ wird dann berechnet als

$$\sigma \equiv M^d \bmod N.$$

Vfy(pk, m, σ). Der Verifikationsalgorithmus berechnet ebenfalls die gepaddete Nachricht

$$M' := \texttt{0x00}||\texttt{0x01}||\texttt{0xFF}||\ldots||\texttt{0xFF}||\texttt{0x00}||\mathsf{ASN.1}||H(m).$$

Er gibt 1 aus, wenn $M' \equiv \sigma^e \bmod N$ gilt, und ansonsten 0.

Remark 117. Falls die Hashfunktion SHA-1 benutzt wird, so ist der Hashwert $H(m)$ genau 20 Bytes lang. Die Länge des ASN.1-Strings beträgt 15 Byte, sodass der String

$$\texttt{0x00}||\mathsf{ASN.1}||H(m)$$

insgesamt 36 Bytes lang ist. Diese Parameter werden wir für die Beschreibung des Angriffs später annehmen. Der Angriff funktioniert jedoch auch für andere Hashfunktionen.

6.3.2 Bleichenbacher's Angriff auf RSA-Signaturen mit kleinem Exponenten

Auf der Rump-Session der Crypto 2006 Konferenz hat Daniel Bleichenbacher einen Angriff auf RSA-PKCS#1 v1.5 Signaturen vorgestellt, der dann anwendbar ist wenn zwei Kriterien erfüllt sind [Fin06].

1. Das Opfer verwendet einen RSA *public key* (N, e) mit $e = 3$.
2. Der Verifikationsalgorithmus hat einen bestimmten Implementierungsfehler. Dieser Fehler ist jedoch recht „natürlich", es ist nicht unwahrscheinlich, dass dieser Fehler in der Praxis gemacht wird. Insbesondere hat Bleichenbacher ein Beispiel für eine Anwendung gefunden, die angreifbar war [Fin06].

Remark 118. Der in diesem Kapitel beschriebene Angriff ist nicht zu verwechseln mit dem (wesentlich berühmteren) Angriff von Bleichenbacher auf PKCS#1 v1.5 *Verschlüsselung* [Ble98].

[6] Die in RFC 3447 beschriebene Version 2.1 erlaubt die Hashfunktionen MD2, MD5, SHA-1, SHA-256 und SHA-384. MD2 und MD5 werden jedoch nicht mehr empfohlen, und sind nur noch aus Gründen der Rückwärtskompatibilität enthalten.

Der Implementierungsfehler. Vfy erhält als Eingabe $pk = (N, e)$ mit $e = 3$, eine Nachricht m, und die Signatur σ. In Bleichenbacher's Angriffsszenario geht der Verifikationsalgorithmus Vfy wie folgt vor. Zunächst berechnet er

$$M' := \sigma^3 \bmod N.$$

Dann wird geprüft, ob M' die Form

$$M' := \texttt{0x00}||\texttt{0x01}||\texttt{0xFF}||\ldots||\texttt{0xFF}||\texttt{0x00}||\mathsf{ASN.1}||H(m)$$

hat. Dazu geht der Algorithmus so vor:

1. Vfy prüft, ob die ersten zwei Bytes von M' die Form $\texttt{0x00}||\texttt{0x01}$ haben.
2. Vfy prüft, ob alle folgenden Bytes die Form $\texttt{0xFF}$ haben, so lange bis ein $\texttt{0x00}$-Byte kommt.
3. Vfy ermittelt, welche Hashfunktion benutzt wurde, indem es den nun auf das $\texttt{0x00}$-Byte folgenden String ASN.1 ausliest. Nehmen wir an, die im ASN.1-String spezifizierte Hashfunktion hat eine Ausgabelänge von ℓ_H Bytes.
4. Zum Schluss prüft Vfy, ob die auf den ASN.1-String folgenden ℓ_H Bytes identisch zum Hashwert der Nachricht ist. Falls ja, so wird die Signatur akzeptiert.

Dieser Algorithmus, der recht natürlich ist, hat den folgenden Fehler. Es wird nicht geprüft, ob nach dem Hashwert noch weitere Daten stehen. Bei einer Signatur die über einen korrekt gepaddeten Hashwert gebildet wurde dürften dort keine weiteren Daten stehen. Der Algorithmus überprüft dies jedoch nicht. Eine Signatur wird also auch dann akzeptiert, wenn $\sigma^3 \bmod N$ die Form

$$\sigma^3 \bmod N = \texttt{0x00}||\texttt{0x01}||\texttt{0xFF}||\ldots||\texttt{0xFF}||\texttt{0x00}||\mathsf{ASN.1}||H(m)||\mathsf{Anhang}$$

hat, wobei Anhang ein beliebiger Wert ist.

Der Angriff. Die Tatsache, dass die Bits Anhang am Ende von $\sigma^3 \bmod N$ ignoriert werden, gibt dem Angreifer die Freiheit dort einen beliebigen Wert anzugeben. Die Idee von Bleichenbacher ist, den Wert Anhang so zu wählen, dass

$$\tilde{M} = \texttt{0x00}||\texttt{0x01}||\texttt{0xFF}||\ldots||\texttt{0xFF}||\texttt{0x00}||\mathsf{ASN.1}||H(m)||\mathsf{Anhang}$$

eine *dritte Wurzel in* $\mathbb{Z}$ hat. Falls dies gelingt, so kann man einfach σ als die dritte Wurzel von $\tilde{M}$ in $\mathbb{Z}$ berechnen – ohne „Modulo-Reduktion".

Example 119. Betrachten wir einen RSA-Modulus N der Länge 3072-Bit, und nehmen wir an dass die Hashfunktion SHA-1 benutzt wird.[7] Um eine beliebige Nachricht m zu signieren geht der Angreifer wie folgt vor:

[7] In diesem Falle gibt es ein besonders einfaches Beispiel aus [Fin06].

1. Der Angreifer berechnet

$$D := \texttt{0x00}||\mathsf{ASN.1}||H(m) \qquad \text{und} \qquad \nu := 2^{288} - D.$$

   Falls ν nicht durch 3 teilbar ist, so ändert der Angreifer die Nachricht m ein wenig ab, so lange bis ν ein Vielfaches von 3 ist.

2. Dann berechnet der Angreifer den Wert

$$\sigma := 2^{1019} - \frac{\nu}{3} \cdot 2^{34},$$

   und gibt diesen Wert als Signatur für m aus.

Um zu zeigen, dass diese Signatur vom oben beschriebenen Verifikationsalgorithmus akzeptiert wird, müssen wir zeigen dass $\sigma^3 \equiv \left(2^{1019} - \frac{\nu}{3} \cdot 2^{34}\right)^3 \bmod N$ die Form

$$\left(2^{1019} - \frac{\nu}{3} \cdot 2^{34}\right)^3 = \texttt{0x00}||\texttt{0x01}||\texttt{0xFF}||\ldots||\texttt{0xFF}||\texttt{0x00}||\mathsf{ASN.1}||H(m)||\mathsf{Anhang}$$

hat, für einen beliebigen String Anhang.

Durch Anwendung der Formel $(A-B)^3 = A^3 - 3A^2B + 3AB^2 - B^3$ erhalten wir

$$\begin{aligned}\left(2^{1019} - \frac{\nu}{3} \cdot 2^{34}\right)^3 &= 2^{3057} - \nu \cdot 2^{2072} + (\nu^2/3 \cdot 2^{1087}) - (\nu^3/27 \cdot 2^{102}) \\ &= 2^{3057} + (D - 2^{288}) \cdot 2^{2072} + (\nu^2/3 \cdot 2^{1087}) - (\nu^3/27 \cdot 2^{102}) \\ &= 2^{3057} - 2^{288} \cdot 2^{2072} + D \cdot 2^{2072} + (\nu^2/3 \cdot 2^{1087}) - (\nu^3/27 \cdot 2^{102}) \\ &= 2^{3057} - 2^{2360} + D \cdot 2^{2072} + (\nu^2/3 \cdot 2^{1087}) - (\nu^3/27 \cdot 2^{102})\end{aligned}$$

Wenn wir diese Zahl als Bit-String aufschreiben, dann ergibt sich ein genau 3072-Bit langer String der Form

$$\texttt{0x00}||\texttt{0x01}||\texttt{0xFF}||\ldots||\texttt{0xFF}||\texttt{0x00}||\mathsf{ASN.1}||H(m)||\mathsf{Anhang},$$

wobei $\mathsf{Anhang} = (\nu^2/3 \cdot 2^{1087}) - (\nu^3/27 \cdot 2^{102})$ ist.

Excercise 120. Beschreiben Sie einen korrekten Verifikationsalgorithmus, bei dem Bleichenbacher's Signatur-Angriff nicht anwendbar ist.

Remark 121. Neben diesem Angriff hat RSA mit Exponent $e = 3$ in bestimmten Einsatzszenarien noch weitere Schwächen (zum Beispiel ist Broadcast-Verschlüsselung mit dem „Lehrbuch"-RSA Verfahren unsicher), auf die wir hier jedoch nicht näher eingehen.

Bei vielen Verfahren ist jedoch gegen $e = 3$ nichts einzuwenden. Dazu zählen zum Beispiel RSA-FDH Signaturen (Kapitel 4.2) oder RSA-Verschlüsselung, falls ein gutes Padding-Verfahren verwendet wird (zum Beispiel RSA-OAEP Verschlüsselung [BR94]) und das Verfahren *korrekt implementiert* ist.

Die Praxisrelevanz von Bleichenbacher's Angriff belegt zum Beispiel die Verwundbarkeit des LK-Bootloaders des Android Betriebssystems, welche im Sommer 2014 entdeckt wurde.[8]

[8] Siehe https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0973.

Literaturverzeichnis

[BB04a] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer, May 2004.

[BB04b] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer, May 2004.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, August 2001.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, May 2003.

[BHJ+13] Florian Böhl, Dennis Hofheinz, Tibor Jager, Jessica Koch, Jae Hong Seo, and Christoph Striecks. Practical signatures from standard assumptions. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 461–485. Springer, 2013.

[BK10] Zvika Brakerski and Yael Tauman Kalai. A framework for efficient signatures, ring signatures and identity based encryption in the standard model. Cryptology ePrint Archive, Report 2010/086, 2010. http://eprint.iacr.org/.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer, August 1998.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume

2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, December 2001.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

[BPVY00] Ernest F. Brickell, David Pointcheval, Serge Vaudenay, and Moti Yung. Design validations for discrete logarithm based signature schemes. In Hideki Imai and Yuliang Zheng, editors, *PKC 2000: 3rd International Workshop on Theory and Practice in Public Key Cryptography*, volume 1751 of *Lecture Notes in Computer Science*, pages 276–292. Springer, January 2000.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73. ACM Press, November 1993.

[BR94] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, May 1994.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer, May 1996.

[BR08] Mihir Bellare and Todor Ristov. Hash functions from sigma protocols and improvements to VSH. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 125–142. Springer, December 2008.

[Bro02] Daniel R. L. Brown. Generic groups, collision resistance, and ecdsa. Cryptology ePrint Archive, Report 2002/026, 2002. http://eprint.iacr.org/.

[BSI08] Kryptographische Verfahren: Empfehlungen und Schlüssellängen. Bundesamt für Sicherheit in der Informationstechnik (BSI), BSI TR-02102, 2008. https://www.bsi.bund.de/ContentBSI/Publikationen/TechnischeRichtlinien/tr02102/index_htm.html.

[BSW06] Dan Boneh, Emily Shen, and Brent Waters. Strongly unforgeable signatures based on computational Diffie-Hellman. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 229–240. Springer, April 2006.

[CF05] Henri Cohen and Gerhard Frey, editors. *Handbook of elliptic and hyperelliptic curve cryptography*. CRC Press, 2005.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th Annual ACM Symposium on Theory of Computing*, pages 209–218. ACM Press, May 1998.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 207–222. Springer, May 2004.

[Cra96] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, CWI and Uni.of Amsterdam, November 1996.

[CS99] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. In *ACM CCS 99: 6th Conference on Computer and Communications Security*, pages 46–51. ACM Press, November 1999.

[DA99] Tim Dierks and Christopher Allen. *RFC 2246 - The TLS Protocol Version 1.0*. Internet Activities Board, January 1999.

[Deb08] Debian Sicherheitsankündigung. DSA-1571-1 openssl – Voraussagbarer Zufallszahlengenerator, 2008. http://www.debian.org/security/2008/dsa-1571.

[DHT12] Yevgeniy Dodis, Iftach Haitner, and Aris Tentes. On the instantiability of hash-and-sign RSA signatures. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 112–132. Springer, March 2012.

[DN10] Ivan Damgård and Jesper Buus Nielsen. Cryptologic Protocol Theory. Course Website, 2010. http://www.daimi.au.dk/~ivan/CPT.html.

[DOP05] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 449–466. Springer, August 2005.

[DSS09] Digital signature standard (DSS). National Institute of Standards and Technology (NIST), FIPS PUB 186-3, U.S. Department of Commerce, 2009. http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf.

[EGM96] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, 1996.

[Fin06] Hal Finney. Bleichenbacher's RSA signature forgery based on implementation error. Posting at IETF-OpenPGP Mailing List, 2006. https://www.ietf.org/mail-archive/web/openpgp/current/msg00999.html.

[Fis03] Marc Fischlin. The Cramer-Shoup strong-RSA signature scheme revisited. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 116–129. Springer, January 2003.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, August 1987.

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139. Springer, May 1999.

[GMR85] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A "paradoxical" solution to the signature problem (abstract) (impromptu talk). In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, page 467. Springer, August 1985.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[Gol87] Oded Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 104–110. Springer, August 1987.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 444–459. Springer, December 2006.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, April 2008.

[HJK11] Dennis Hofheinz, Tibor Jager, and Eike Kiltz. Short signatures from weaker assumptions. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 647–666. Springer, December 2011.

[HJK12] Dennis Hofheinz, Tibor Jager, and Edward Knapp. Waters signatures with optimal security reduction. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Workshop on Theory and Practice in Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 66–83. Springer, May 2012.

[HK08] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 21–38. Springer, August 2008.

[HW09a] Susan Hohenberger and Brent Waters. Realizing hash-and-sign signatures under standard assumptions. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 333–350. Springer, April 2009.

[HW09b] Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 654–670. Springer, August 2009.

[JK03] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447 (Informational), February 2003.

[Jou04] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, September 2004.

[Kal98] B. Kaliski. PKCS #1: RSA Encryption Version 1.5. RFC 2313 (Informational), March 1998. Obsoleted by RFC 2437.

[Kat10] Jonathan Katz. *Digital Signatures*. Springer-Verlag, 2010.

[KK12] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 537–553. Springer, April 2012.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[KPR03] Vlastimil Klíma, Ondrej Pokorný, and Tomás Rosa. Attacking RSA-based sessions in SSL/TLS. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 426–440. Springer, September 2003.

[KR00] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *ISOC Network and Distributed System Security Symposium – NDSS 2000*. The Internet Society, February 2000.

[KR02] Vlastimil Klíma and Tomás Rosa. Attack on private signature keys of the OpenPGP format, PGP(TM) programs and other applications compatible with OpenPGP. Cryptology ePrint Archive, Report 2002/076, 2002. http://eprint.iacr.org/.

[KS98] B. Kaliski and J. Staddon. PKCS #1: RSA Cryptography Specifications Version 2.0. RFC 2437 (Informational), October 1998. Obsoleted by RFC 3447.

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, October 1979.

[Lin03] Yehuda Lindell. A simpler construction of cca2-secure public-key encryption under general assumptions. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 241–254. Springer, May 2003.

[LOS+06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 465–485. Springer, May / June 2006.

[May04] Alexander May. Computing the RSA secret key is deterministic polynomial time equivalent to factoring. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 213–219. Springer, August 2004.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, August 1988.

[Moh10] Payman Mohassel. One-time signatures and chameleon hash functions. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *SAC 2010: 17th Annual International Workshop on Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 302–319. Springer, August 2010.

[MVO91] Alfred Menezes, Scott A. Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *STOC*, pages 80–89. ACM, 1991.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer, August 2002.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, August 1992.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer, May 1996.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Symposium on Foundations of Computer Science*, pages 543–553. IEEE Computer Society Press, October 1999.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, August 1990.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[Sch11] Sven Schäge. Tight proofs for signature schemes without random oracles. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 189–206. Springer, May 2011.

[Sha83] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Trans. Comput. Syst.*, 1(1):38–44, 1983.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, August 1985.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, May 1997.

[SPW07] Ron Steinfeld, Josef Pieprzyk, and Huaxiong Wang. How to strengthen any weakly unforgeable signature into a strongly unforgeable signature. In Masayuki Abe, editor, *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *Lecture Notes in Computer Science*, pages 357–371. Springer, February 2007.

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, May 2005.

[Zha07] Rui Zhang. Tweaking TBE/IBE to PKE transforms with chameleon hash functions. In Jonathan Katz and Moti Yung, editors, *ACNS 07: 5th International Conference on Applied Cryptography and Network Security*, volume 4521 of *Lecture Notes in Computer Science*, pages 323–339. Springer, June 2007.