Bachelorarbeit

Digitale Signaturen und Sicherheit

Samed Saroglu (10004887)

15. August 2019

Institut für Theoretische Informatik
Fakultät für Elektrotechnik und Informatik
Leibniz Universität Hannover

Erstprüfer:	Prof. Dr. Heribert Vollmer
Zweitprüfer:	Dr. Arne Meier
Betreuer:	Dr. Arne Meier, M. Sc. Fabian Müller

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die wörtlich oder inhaltlich aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Datum, Ort	Unterschrift

Inhaltsverzeichnis

1. Einleitung

1.1. Einführung

In der heutigen Welt ist die Gewährleistung von Sicherheit unter der stetig fortschreitenden Digitalisierung zunehmend schwieriger. Immer mehr Menschen verwenden aus Bequemlichkeit, aber auch aufgrund der großen Verfügbarkeit und leichten Handhabe, *Electronic Banking* (auch genannt: *Online-Banking*) als Zahlungsmethode für ihren Einkauf im Internet. Des Weiteren finden auch Überweisungen immer häufiger über das Internet statt.

(1) Woher aber wissen wir, dass das Geld auch wirklich vom Absender stammt und der Betrag nicht auf dem Weg modifiziert wurde? (2) Wie können wir gewährleisten, dass das Geld auch tatsächlich beim Empfänger ankommt? (3) Wie lässt sich sicherstellen, dass eine Transaktion weder vom Sender noch vom Empfänger im Nachhinein zurückgezogen werden kann?

Diese (Sicherheits-)Probleme lassen sich in drei Oberbegriffe kategorisieren. Es handelt sich hier um (1) die *Integrität*, (2) die *Authentizität* und (3) die *Nicht-Abstreitbarkeit* einer Nachricht beziehungsweise einer Transaktion wie im angeführten Beispiel [Buc16, S. 245, *Kap. 12.1*].

Unter Nichtbeachtung genannter Probleme wäre es Angreifern möglich, sich als Absender auszugeben. Als Empfänger geht man davon aus, dass die Nachricht vom ursprünglichen Absender stammt, wobei sie tatsächlich während der Übertragung vom Angreifer beliebig modifiziert wurde (auch genannt: *Man-in-the-Middle-Angriff*). Dies ist analog für elektronische Geldtransaktionen anwendbar; beispielsweise wenn sich ein Angreifer bei der Bank als Kontoinhaber ausgibt und daraufhin unter falscher Identität Geld abhebt.

1.2. Funktionsweise von digitalen Signaturen

Zur Vermeidung eines solchen Szenarios sind digitale oder elektronische Signaturen anwendbar, mit denen sich die drei oben genannten Probleme lösen lassen. So können wir eine Nachricht oder die Transaktion aus unserem Beispiel signieren. Dabei erhalten wir eine Signatur, die eine Unterschrift in digitaler Form darstellt und welche wir dann gebündelt mit der Nachricht dem Empfänger zusenden. Die Signatur hat die Eigenschaften, dass die Gültigkeit auf die jeweilige Nachricht beschränkt ist und bei Veränderung der Nachricht die Signatur ungültig wird. Außerdem muss sich aus der Signatur eindeutig erschließen lassen, dass sie vom Absender stammt. Damit ist (1) die *Integrität*, (2) die *Authentizität* und (3) die *Nicht-Abstreitbarkeit* durch die Signatur gewährleistet.

Um die obere Beschreibung der Funktionsweise von digitalen Signaturen zu konkretisieren, benötigen wir also ein Verfahren, welches diese Eigenschaften berücksichtigt. Dafür werden als Grundlage *Public-Key-Kryptosysteme* verwendet, welche einen asymmetrischen Verschlüsselungsalgorithmus nutzen [Buc16]. Vereinfacht dargestellt besitzt man bei solchen Kryptosystemen ein *Schlüsselpaar* mit je einem *öffentlichen* und einem *privaten Schlüssel*, wobei der private Schlüssel einer Geheimhaltung unterliegt, während der öffentliche Schlüssel für andere herausgegeben wird. Mit dem *öffentlichen Schlüssel* lassen sich dann Nachrichten an den Besitzer des verwendeten Schlüssels verschlüsseln. Diese verschlüsselten Nachrichten können dann nur mit dem *privaten Schlüssel*, welcher aus dem selben Schlüsselpaar stammt, entschlüsselt werden.

Bei digitalen Signaturen wird der *private Schlüssel* als *Signierschlüssel* verwendet, mit welchem wir eine Nachricht unter Verwendung eines *Signieralgorithmus* signieren. Der *öffentliche Schlüssel* wird als *Verifikationsschlüssel* genutzt, sodass der Empfänger der Nachricht mit dem *Verifikationsschlüssel* des Absenders, der Signatur und eines *Verifikationsalgorithmus* die Nachricht verifizieren lassen kann. So lässt sich bestätigen, dass die Nachricht tatsächlich vom Absender stammt und seit der Erstellung der dazugehörigen Signatur nicht modifiziert wurde [Buc16, S. 246, *Def. 12.2*].

1.3. Sicherheit

Die Sicherheit einer digitalen Signatur hängt von der Sicherheit des zugrundeliegenden *Public-Key-Kryptosystems* beziehungsweise einer jeden Komponente ab, denn die Sicherheit eines Kryptosystems ist nur so stark, wie die ihrer schwächsten Komponente.

Das Hauptziel eines Angreifers ist hierbei die Überzeugung des Gegenübers, dass er der gewünschte Kommunikationspartner sei, indem er versucht, ohne Kenntnis des *Signierschlüssels* eine gültige Signatur zu erstellen [Buc16, S. 249, *Kap. 12.4.1*].

Es existieren verschiedene Arten von Fälschungen für Signaturen:

Bei einer *existentiellen Fälschung* lässt sich die Signatur einer Nachricht fälschen. Bei einer *selektiven Fälschung* kann man die Signatur von einer Nachricht seiner Wahl fälschen. Von einer *universelle Fälschung* spricht man, wenn der Angreifer Signaturen für beliebe Nachrichten fälschen kann. Bei einem *Total Break* kann der Angreifer den *Signierschlüssel* berechnen und anschließend jede Nachricht damit signieren. [GMR88]

Außerdem gibt es, neben den verschiedenen Fälschungen, unterschiedliche Angriffsarten auf Signaturverfahren:

Zunächst haben wir den sogenannten *No-Message-Angriff*. Hier hat der Angreifer nur den Zugang zum *Verifikationsschlüssel*. Weiterhin gibt es den *Known-Message-Angriff*. Hier besitzt der Angreifer zusätzlich zum *Verifikationsschlüssel* noch eine Menge an Nachrichten und deren Signaturen. Zuletzt haben wir den *Chosen-Message-Angriff*. Der Angreifer besitzt den *Verifikationsschlüssel* und kann sich außerdem Nachrichten signieren lassen, wobei diese nicht vom ursprünglichen *Nachrichten-Schlüsselpaar* stammen darf, welcher konstruiert werden soll. Dies stellt ein Beispiel einer *existentiellen Fälschung* dar [Buc16, S. 250, *Kap. 12.4.2*].

1.4. Thema dieser Bachelorarbeit

Die vorliegende Bachelorarbeit befasst sich mit der grundlegenden Definition von digitalen Signaturen. Dabei gehen wir auf die Konzepte und Voraussetzungen von Signaturverfahren ein und sprechen über die Sicherstellung dieser. Hierzu werden wir uns hauptsächlich mit den digitalen Signaturverfahren, die das *RSA*-Verfahren verwenden. Außerdem besprechen wir unter Anführung von Beispielen die möglichen Angriffsarten bei Signaturverfahren, deren Funktionsweise. Neben den Angriffsarten gehören auch die Sicherheitskonzepte von Signaturverfahren zum Inhalt der Arbeit. Aufgeführt werden soll, wie sicher die jeweiligen Verfahren gegenüber den möglichen Angriffsarten sind.

2. Grundlagen

Dieser Abschnitt behandelt die erforderlichen Definitionen für diese Arbeit. Diese umfassen die Sicherheitsmodelle für digitale Signaturen, die hierfür benötigten Public-Key-Kryptosysteme mit einer Vertiefung auf das RSA-Verfahren, ein Einblick in die Zufallszahlen und Hashfunktionen. Im Laufe dieser Arbeit werden die aufgeführten Themen an diversen Stellen erweitert.

2.1. Zufallszahlen

Zufall spielt eine große Rolle in der Informatik. Sei es im Bereich der Simulation, womit die zufälligen Ereignisse der Natur simuliert werden, oder auch für das Testen der Funktionalität der Algorithmen [Knu97].

Bei dieser Arbeit spielt Zufall beziehungsweise spielen die Zufallszahlen ebenso eine wichtige Rolle. Die Generierung der Primzahlen beim *RSA*-Verfahren und somit auch der Schlüssel sollte zufällig erfolgen, damit man nicht auf die gewählten Primzahlen rückschließen kann.

Wir gehen bei unseren probabilistischen Algorithmen davon aus, dass deren Wahrscheinlichkeitsverteilung gleichverteilt ist, das heißt also, dass alle Ereignisse mit einer gleichen Wahrscheinlichkeit auftreten können. Dies ist in der Realität nicht immer der Fall.

Das Thema über Zufall, insbesondere Zufallszahlen wird im Buch von Knuth [Knu97] intensiver behandelt. Wir werden aber nicht weiter auf dieses Thema eingehen, da ein grober Überblick ausreicht.

2.2. Laufzeit von Algorithmen

Ein wichtiger Aspekt bei der Betrachtung von digitalen Signaturen und dessen Sicherheit ist die Laufzeit von den Algorithmen, die in diesen Verfahren genutzt werden. Diese sollte so klein wie möglich gehalten werden, aber dennoch eine gewisse

Sicherheit gewährleisten (In Abschnitt 3 gehen wir genauer auf den Sicherheitsbegriff ein).

2.2.1. Polynomielle Laufzeit

Wir sagen, dass ein Algorithmus ein Polynomialzeit-Algorithmus ist, wenn seine „worst-case“ Laufzeit durch ein Polynom, mit Abhängigkeit seiner Eingabelänge x, beschränkt ist $O(x^n)$ für ein $n \in \mathbb{N}$ [Buc16].

Die hier dargelegten Algorithmen werden hauptsächlich eine polynomielle Laufzeit besitzen, da diese in einer absehbaren Zeit lösbar beziehungsweise berechenbar sind.

2.3. Einwegfunktionen

Einwegfunktionen haben die Eigenschaft, dass sie einfach zu berechnen, aber schwer umzukehren beziehungsweise zu invertieren sind. Ein alphabetisch sortiertes Telefonbuch ist ein praktisches Beispiel dafür. Mit Hilfe eines Namens kann die dazugehörige Telefonnummer sehr leicht herausgefunden werden. Wenn man das Beispiel umgekehrt versucht, also mit Hilfe einer Telefonnummer den dazugehörigen Namen herauszufinden, wird das ganze zu einem Problem.

Hinweis: Wir beziehen uns bei dem Begriff „leicht“ darauf, dass die Funktion, durch einen Polynomialzeit-Algorithmus berechnet werden kann. Dementsprechend weist der Begriff „schwer“ daraufhin, dass kein Polynomialzeit-Algorithmus, das Inverse dieser Funktionen berechnen kann [BG08, Jag18].

2.3.1. Hashfunktionen

Wir schauen uns die Definition von Hashfunktionen an, da sie benötigt werden, um *Kryptographische Hashfunktionen* zu definieren und ein wichtiger Bestandteil für die Authentizität und Sicherheit von digitalen Signaturen ist.

Grundsätzlich komprimieren Hashfunktionen Wörter zu kleineren Wörtern. Daraus kann dann meistens nicht mehr das ursprüngliche Wort erkannt werden. Dem entsprechend können Hashfunktionen in ähnlicher Weise wie Prüfsummen betrachtet werden. Wir geben mit Hilfe von Buchmann [Buc16] eine formale Definition für Hashfunktionen:

Definition 1 (Hashfunktionen)**.** Sei H eine Hashfunktion:

$$H(s) = h, \text{ mit:}$$

$$H : \Sigma^* \to \Sigma^n, \quad n \in \mathbb{N}$$

H bekommt als Eingabe ein Wort s und gibt eine komprimierte Version h (auch *Hash* genannt) aus. Hierbei ist zu beachten, dass die Ausgabe immer eine feste Größe n besitzt und somit unabhängig von der Länge der Eingabe s ist.

Bemerkung: Σ ist eine endliche Menge, welches unser Alphabet definiert. Σ^n ist ein n-Zeichen langes Wort aus unserem Alphabet. Σ^* ist wiederum ein beliebig langes Wort.

Kryptographische Hashfunktionen gehören zu den Einwegfunktionen und sind spezifische Hashfunktionen, welche in der Kryptographie Anwendung finden. Diese haben besondere Eigenschaften. Unter anderem sind sie *kollisionsresistent*, das heißt die Wahrscheinlichkeit ist vernachlässigbar klein für zwei verschiedene Eingaben, den gleichen Hash zu erhalten. Außerdem gelten die Eigenschaften der Einwegfunktionen. Es sollte demnach nahezu unmöglich sein, dass man aus einem Hash h, die Eingabe s *nicht* herausfinden kann. Ebenso sollte beachtet werden, dass die Hashes effizient berechnet werden können [Buc16, Jag18].

Die Anwendung dieser Funktionen kann man zum Beispiel bei der Authentifizierung von Benutzern, mit Hilfe von Passwörtern, wiedererkennen. Wenn ein Benutzer sich bei einer Internetseite anmelden möchte, muss er dazu seinen Benutzernamen und das dazugehörige Passwort eingeben. Die Internetseite speichert diese Informationen in einer Datenbank ein, wobei das Passwort nicht ohne Weiteres als Klartext gespeichert wird, da dies einen Risikofaktor darstellt. Falls die Datenbank kompromittiert wird, können die Passwörter durch Dritte gelesen werden. Deshalb verwendet man Hashfunktionen und speichert den Hashwert des Passwortes. Falls der Benutzer dann ein Passwort eingibt, wird der Hashwert aus der Datenbank mit dem Hashwert des eingegebenen Passwortes überprüft. Bei Übereinstimmung wird der Zugang für den Benutzer gewährt, andernfalls nicht [BG08].

2.3.2. Einwegfunktionen mit Hintertür

Einwegfunktionen mit Hintertür besitzen, zusätzlich zu den normalen Einwegfunktionen, die Eigenschaft, dass man sie mit Hilfe einer geheimen Zusatzinformation effizient

umkehren kann. Diese Eigenschaft ist ein wichtiger Bestandteil der *RSA*-Funktion, welcher im weiteren Verlauf dieser Arbeit definiert wird.

Vorstellen kann man sich das an einem realen Beispiel anhand einer Spardose: Jedem ist es möglich Münzen in eine Spardose hineinzuwerfen, aber nur dem Besitzer des Schlüssels ist es auch möglich das Geld wieder herauszuholen [BG08, Jag18].

2.4. Random Oracle Modell

Bei dem *Random Oracle Modell* handelt es sich um eine ideale Hashfunktion. Das heißt, sie gibt für ein Wort s einen *gleichverteilt zufälligen Wert* h zurück: $ROM(s) = h$. Dabei speichert das *Random Oracle* das Wort s und den dazugehörigen Hash h ab. Somit kriegt man bei mehrfacher Anfrage eines gleichen Wortes w, immer den gleichen Hash h zurück. Bei einem neuen Wort s wird wiederum dann wieder *gleichverteilt zufälligen Wert* und dieser abgespeichert. Außerdem kann man mit Hilfe des Hashes h keine Rückschlüsse zu dem dazugehörigen Wort w ziehen. [Jag18].

2.5. Public-Key-Kryptosysteme

Bei einem *Public-Key-Kryptosystem*, auch *asymmetrisches Kryptosystem* genannt, handelt es sich um ein (Public-Key-)Verschlüsselungsverfahren. Man besitzt, anders als beim *symmetrischen Kryptosystem*, nicht einen gemeinsamen Schlüssel für die Ver- und Entschlüsselung, sondern ein Schlüsselpaar (e, d). Mit Hilfe von e kann verschlüsselt werden und wird daher öffentlich zugänglich gemacht. Aus diesem Grund nennt man e den *öffentlichen Schlüssel*. Analog dazu werden mit Hilfe von d Chiffretexte entschlüsselt und muss daher geheim gehalten werden. Man nennt d deshalb auch den *privaten Schlüssel* [Buc16].

Eine formale Definition von einem *Public-Key-Kryptosystem* folgt aus den Arbeiten von Bellare, Rogaway [BR05, Def 11.1] und Buchmann [Buc16]:

Definition 2 (Public-Key-Kryptosystem)**.** Sei *PKS* ein Tripel von Algorithmen, welches ein *Public-Key-Kryptosystem* definiert:

$$PKS = (K, E, D), \text{ mit:}$$

$$K(1^k) = (e, d), \quad k \in \mathbb{N},$$

$$E(e, m) = c,$$

$$D(d, c) = m.$$

K ist ein probabilistischer *Schlüsselerzeugungsalgorithmus*, welcher als Eingabe einen Sicherheitsparameter 1^k bekommt und als Ausgabe ein zufällig generiertes Schlüsselpaar (e, d) ausgibt. Der Sicherheitsparameter 1^k beschränkt die Erzeugung der möglichen Schlüsselpaare auf einen bestimmten Bereich im *Schlüsselraum*, welcher die Menge aller möglichen Schlüsselpaare beinhaltet.

E ist hingegen ein *Verschlüsselungsalgorithmus*, welcher als Eingabe den *öffentlichen Schlüssel* e sowie die zu verschlüsselnde Nachricht m bekommt und einen Chiffretext c zu m ausgibt, wenn die Nachricht m im *Nachrichtenraum* liegt. Der *Nachrichtenraum* ist gebunden an einen *öffentlichen Schlüssel* e und definiert die Menge an Nachrichten, die mit e verschlüsselt werden können.

D ist ein deterministischer *Entschlüsselungsalgorithmus* und bekommt als Eingabe einen *privaten Schlüssel* d und einen Chiffretext c. D entschlüsselt korrekt und gibt eine Nachricht m aus, wenn c von E ausgegeben wurde, mit dem korrespondierenden *öffentlichen Schlüssel* e zu d. Es muss also $D(d, E(e, m))$ gelten.

2.5.1. RSA

Das *RSA*-Verfahren, benannt nach seinen Erfindern *Ronald L. Rivest*, *Adi Shamir* und *Leonard Adleman* [Buc16, S. 168], ist ein sogenanntes *Public-Key-Kryptosystem* und wird zum einen für eine verschlüsselte Kommunikation und zum anderen als Grundlage für das digitale Signieren verwendet. Für die folgende Definition von RSA folgen wir Buchmann, Bellare und Rogaway [Buc16, BR96]:

Definition 3 (RSA)**.** *RSA* ist ein *PKS* mit:

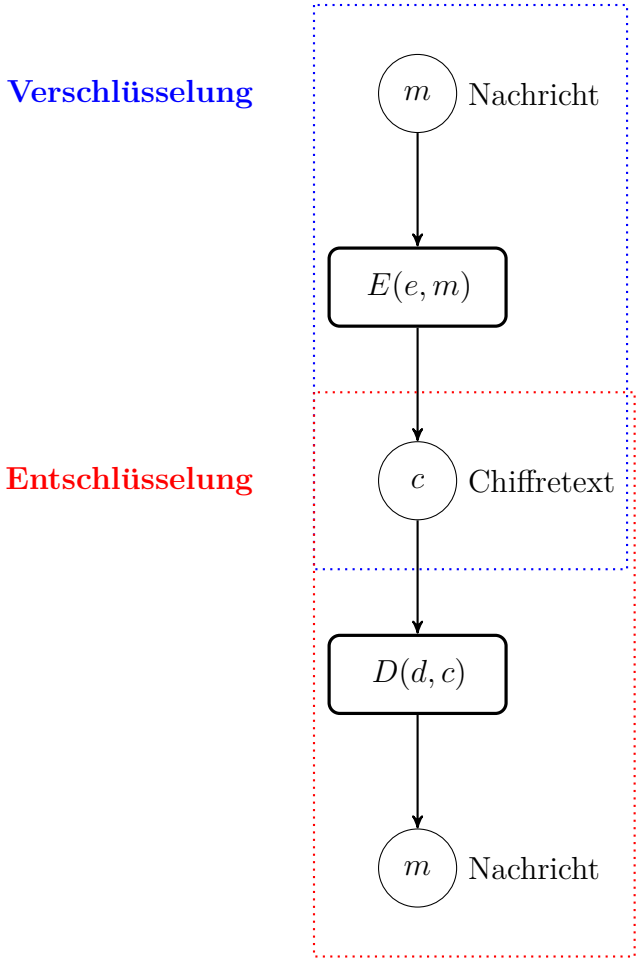
$$K(1^k) = ((e, N), (d, N)), \quad k \in \mathbb{N},$$

$$E(e, m) = m^e \bmod N,$$

$$D(d, c) = c^d \bmod N.$$

Schlüsselerzeugungsalgorithmus K**:** (N, e) sei der *öffentliche Schlüssel* und (N, d) der *private Schlüssel*. N ist ein k-bit-Modulo, welches das Produkt von zwei zufällig gewählten $\frac{k}{2}$-bit Primzahlen p, q ist. $e \in \mathbb{N}$ wird ebenfalls zufällig

Abbildung 2.5.1.: Ver- und Entschlüsselung einer Nachricht



ausgewählt, mit folgenden Eigenschaften:

$$1 < e < \varphi(N) \text{ und } ggT(e, \varphi(N)) = 1.$$

$d \in \mathbb{N}$ wird wiederum berechnet mit:

$$1 < d < \varphi(N) \text{ und } d \cdot e \equiv 1 \bmod \varphi(N)$$

Die Ausgabe ist dann ein Tupel von dem erzeugten Schlüsselpaar, mit dem benötigten Modulo N.

Bemerkung: Bei $\varphi(N)$ handelt es sich hierbei um die Eulersche Phi-Funktion (siehe Anhang A.1).

Verschlüsselungsalgorithmus E: E ist hier die *RSA*-Funktion $f\colon Z_N^* \to Z_N^*$ (siehe

A.2), welche wie folgt definiert ist:

$$c = m^e \bmod N$$

Entschlüsselungsalgorithmus D**:** D ist das Inverse der RSA-Funktion f^{-1}: $Z_N^* \rightarrow Z_N^*$:

$$m = c^d \bmod N$$

Beispiel 3.1. Sei $k = 8$. Das heißt, N ist 8-bit lang. K wählt „zufällig“ die Primzahlen $p = 11$ und $q = 13$, mit jeweils der Länge von 4-bits, aus. N ist somit $11 \cdot 13 = 143$. $\varphi(143) = 120$, da $\varphi(N) = \varphi(p \cdot q) = \varphi(p) \cdot \varphi(q) = (p-1) \cdot (q-1)$ (Siehe auch 7.3). Außerdem wählt K „zufällig“, unter Berücksichtigung der Bedingungen, $e = 7$ aus. Die Voraussetzung $ggT(7, \varphi(143)) = 1$ wird ebenfalls erfüllt. Jetzt müssen wir noch d bestimmen:

Dazu muss gelten: $d \cdot e \equiv 1 \bmod \varphi(N) \Rightarrow d \cdot 7 \equiv 1 \bmod 120$. Der *private Schlüssel* ist also $d = 103$. Die Berechnung erfolgt mit dem *erweiterten euklidischen Algorithmus*.

Unser *öffentlicher Schlüssel* ist somit $(7, 143)$ und unser *privater Schlüssel* ist $(103, 143)$.

Wenn wir jetzt die Nachricht $m = 25$ *verschlüsseln* wollen, berechnen wir dafür $c = m^e \bmod n$, also: $25^7 \bmod 143 = 64$. Die Zahl $c = 64$ ist also unser Chiffretext (beziehungsweise unsere Chiffrezahl) zu unserer Nachricht $m = 25$.

Analog dazu können wir diesen Chiffretext *entschlüsseln*: $m = c^d \bmod n \Rightarrow$ $64^{103} \bmod 143 = 25$. Das Ergebnis entspricht unserer ursprünglichen Nachricht m.

2.6. Digitale Signaturen

In der Einleitung wurden bereits die Anwendungsgebiete und die grobe Funktionsweise von digitalen Signaturen aufgeführt. Nun soll eine formale Definition nach Buchmann, Bellare und Rogaway [Buc16, BR96] sowie ein anschließendes Beispiel folgen.

Definition 4 (Digitale Signaturverfahren)**.** Ein digitales Signaturverfahren wird beschrieben durch ein Tripel:

$$DSS = (K, S, V),$$

wobei K ein probabilistischer *Schlüsselerzeugungsalgorithmus*, S ein probabilistischer

Signieralgorithmus und V ein *Verifikationsalgorithmus* ist. Alle drei Algorithmen haben eine polynomielle Laufzeit.

Schlüsselerzeugung: K gibt bei Eingabe 1^k ein Schlüsselpaar (e, d) zurück. e ist, wie schon zuvor bekannt, der *öffentliche Schlüssel* und d der *private Schlüssel*.

Signatur: S bekommt als Eingabeparameter die zu signierende Nachricht m und d und gibt für m eine gültige *Signatur* x aus:

$$x = S(m, d).$$

Verifikation: V bekommt als Eingabeparameter eine Nachricht m, eine potenzielle *Signatur* x und den *öffentlichen Schlüssel* e und gibt ein Bit b aus:

$$V(m, x, e) = \begin{cases} 1, & \text{falls } x \text{ gültige Signatur für } m \\ 0, & \text{falls } x \text{ ungültige Signatur für } m. \end{cases}$$

Wir sagen, dass unser Signaturverfahren korrekt verifiziert, wenn:

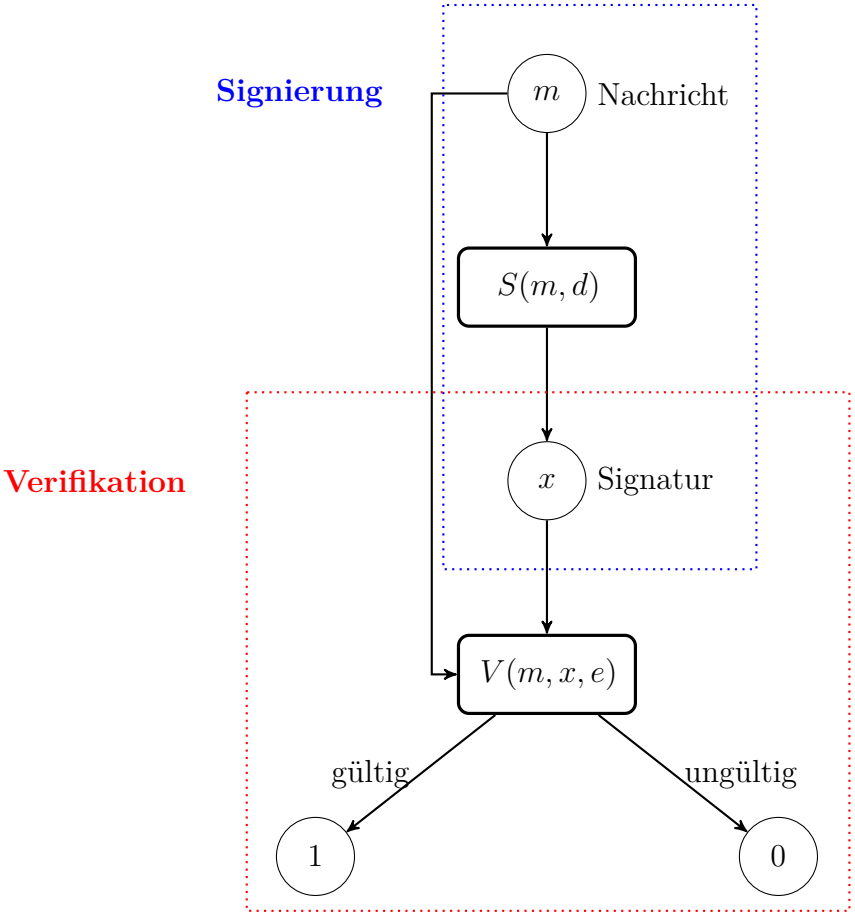
$$V(m, x, e) = 1 \;\Leftrightarrow\; x \in S(m, d).$$

Beispiel 4.1. Wir verwenden für dieses Beispiel das vorhin definierte *RSA*-Verfahren, da es sich, wegen ihrem ähnlichen Aufbau, auch als digitales Signaturverfahren verwenden lässt. Zum Signieren einer Nachricht verwendet man lediglich den Entschlüsselungsalgorithmus D, welcher dann die zu signierende Nachricht m als Eingabe erhält und die dazugehörige Signatur x ausgibt. Zum Verifizieren nutzt man den Verschlüsselungsalgorithmus E, der dann die Signatur x als Eingabe bekommt und als Ausgabe eine Nachricht m' ausgibt. Falls m' identisch zu der Nachricht m ist, gilt die Signatur x als gültig für m, sonst als ungültig. Der Schlüsselerzeugungsalgorithmus K bleibt hierbei identisch.

Für dieses Beispiel werden wir das im vorherigen Abschnitt erstellte Schlüsselpaar verwenden (also $e = 7, d = 103, N = 143$).

Wenn wir also eine Nachricht $m = 19$ *signieren* wollen, müssen wir unsere Nachricht erstmal mit unserem *privaten Schlüssel* signieren. Das heißt konkret, wir „verschlüsseln“ unsere Nachricht mit unserem *privaten Schlüssel* nach dem Prinzip des *RSA-Verfahrens*. Das Ergebnis, und somit auch die Ausgabe von S, wäre dann 10. Anschließend schicken wir die Nachricht m und die Signatur $x = 10$ zum Empfänger.

Abbildung 2.6.1.: Signieren und Verifizieren einer Nachricht



Der Empfänger überprüft die Signatur, indem er den *Verifikationsalgorithmus* V verwendet. Dieser „entschlüsselt“ mit unserem *öffentlichen Schlüssel* die Signatur x und überprüft, ob dies mit dem Wert der Nachricht m übereinstimmt. Bei unserer Überprüfung von x lautet das Ergebnis 19, was mit dem Wert unserer gesendeten Nachricht m übereinstimmt. Somit gibt V eine 1 (also eine gültige Signatur) zurück.

3. Sicherheitsmodelle

In diesem Abschnitt schauen wir uns die Angriffstypen von digitalen Signaturen, aus der wissenschaftlichen Publikation von Goldwasser, Micali und Rivest [GMR88], an und gehen darauf ein, wie solche digitalen Signaturverfahren „gebrochen“ werden können.

Bei den gesamten Angriffstypen haben wir einen Herausforderer H, und einen Angreifer A. Der Angreifer versucht hierbei das vom Herausforderer verwendete digitale Signaturverfahren zu „brechen“.

3.1. Angriffstypen

Es gibt grundsätzlich zwei verschiedene Arten von Angriffen auf Signaturschemata: die sogenannten **Key-Only Angriffe**, bei denen der Angreifer lediglich den *öffentlichen Schlüssel* vom Signierenden kennt sowie die **Message Angriffe**, bei denen der Angreifer Signaturen zu bekannten oder selbst ausgesuchten Nachrichten kennt und mit Hilfe von diesen Informationen das digitale Signaturverfahren angreift [GMR88]. In Tabelle 3.1 unterteilen wir die **Message Angriffe** in weitere Unterkategorien. Die Angriffsarten sind dabei so sortiert, dass die Möglichkeiten des Angreifers vom *Known-Message Angriff* bis zum *Adaptive-Chosen-Message Angriff* sich immer weiter erhöhen und somit auch der Angriff schwieriger wird, da mehr Voraussetzungen vorherrschen.

3.2. Fälschungsarten

Der Erfolg eines Angriffes muss nicht zwangsweise die Ermittlung des *privaten Schlüssels* von H sein. In Tabelle 3.2 zeigen wir, aus der wissenschaftlichen Publikation von Goldwasser, Micali und Rivest [GMR88], die möglichen Fälschungsarten für Signaturverfahren. Dabei sind die Fälschungsarten im absteigenden Ausmaß sortiert. Während ein *Total Break* bei einem digitalen Signaturschema fatal sein würde,

Tabelle 3.1.: Angriffsarten

Angriffsart	**Zugang zu:**	**Bemerkung**
Known-Message	eine Menge an Signaturen von zufälligen Nachrichten $M = m_1, ..., m_n$, die von H kommen	keine Kontrolle über die erhaltene Menge
Generic-Chosen-Message	eine Menge an Signaturen von selbst ausgesuchten Nachrichten $M = m_1, ..., m_n$	M ist unabhängig vom *öffentlichen Schlüssel* von H, M muss also vor Kenntnis des *öffentlichen Schlüssels* erstellt worden sein
Directed-Chosen-Message	eine Menge an Signaturen von selbst ausgesuchten Nachrichten $M = m_1, ..., m_n$	die Menge M kann nach Kenntnis des *öffentlichen Schlüssels*, und somit auch abhängig davon, erstellt werden
Adaptive-Chosen-Message	eine Menge an Signaturen von selbst ausgesuchten Nachrichten $M = m_1, ..., m_n$, Signaturen können ebenso einzeln angefordert werden	angeforderte Signaturen können zusätzlich abhängig von den vorherigen Signaturen sein

ist eine *existentielle Fälschung* nicht immer eine schwerwiegende Sicherheitslücke, da der Angreifer keine Kontrolle darüber hat, welche Signatur gefälscht wird.

3.3. Sicherheits-Spiel

Damit wir die Sicherheit von digitalen Signaturschemata danach bewerten können, wie sicher ein digitales Signaturverfahren ist, definieren wir „Sicherheits-Spiele“. In so einem Spiel spielt ein Angreifer A gegen einen Herausforderer H. Der Angreifer gewinnt das Spiel, falls er das Verfahren, mit dem vorgegebenen Angriffstyp, „bricht“. In unserem Fall sprechen wir von einem „Gewinnen“, wenn A die erwartete Fälschungsart erstellen kann.

Tabelle 3.2.: Fälschungsarten

Fälschungsart	**Bedingungen**	**Möglichkeiten für A**
Total Break	*Privater Schlüssel* kann berechnet werden	Signaturverfahren wurde gebrochen, A kann beliebig viele Nachrichten fälschen
Universelle Fälschung	Besitzt einen *Signieralgorithmus*, der äquivalent zu dem von H ist	Beliebig viele Nachrichten können gefälscht werden
Selektive Fälschung	Besitzt die Signatur einer selbst ausgewählten Nachricht	Eine Signatur, über die H keine Kontrolle hat, kann gefälscht werden
Existentielle Fälschung	Besitzt Signaturen von mindestens einer Nachricht, hat dabei aber *keine* Kontrolle darüber, welche Signatur hier gefälscht wird	Nachricht kann sinnfrei sein und somit keinen wahren Nutzen für A, schwächste Art der Fälschung und, wegen der Unkontrollierbarkeit der Signatur, nicht schwerwiegend

In Abbildung 3.3.1 und 3.3.2 haben wir „Sicherheits-Spiele“ für die in Tabelle 3.1 definierten Angriffstypen: *Directed-Chosen-Message*, *Adaptive-Chosen-Message* erstellt.

Die genauere Auswahl der Angriffstypen hat mehrere Hintergründe:

- Wir müssen in der Praxis davon ausgehen, dass ein Angreifer gewisse Möglichkeiten besitzen kann, zum Beispiel ausgesuchte Nachrichten von dem Herausforderer signieren lassen. Auch dies muss bei den Sicherheitsbeweisen der jeweiligen Signaturverfahren berücksichtigt werden. Außerdem bietet der *Known-Message* Angriff sehr „schwache“ Möglichkeiten, da der Angreifer hier nur eine Menge an Signaturen von zufällig ausgesuchten Nachrichten erhält. Zwischen dem *Generic-* und dem *Directed-Chosen-Message* Angriff haben wir den Unterschied, dass beim *Generic-Chosen-Message* Angriff der *öffentliche Schlüssel* nach Erstellung der Menge an Signaturen, von selbst ausgesuchten Nachrichten, bekannt ist. Da aber der *öffentliche Schlüssel* meistens schon vorher bekannt ist, wählten wir

den *Directed-Chosen-Message* Angriff.

- Der *Adaptive-Chosen-Message* Angriff weist den gravierenden Unterschied auf, dass der Angreifer während des gesamten Angriffes, Signaturen für seine selbst ausgewählten Nachrichten anfragen kann. Dies bietet ihm mehr Möglichkeiten, gezieltere Anfragen, abhängig von den vorherigen, zu stellen.

- Durch die hierarchische Sortierung der Angriffsarten können wir außerdem sagen, dass wenn ein digitales Signaturverfahren sicher ist gegen ein *Adaptive-Chosen-Message* Angriff, dieser auch sicher gegen die unteren drei Angriffsarten ist. Aus diesem Grund decken wir mit unseren gewählten Angriffstypen ebenso die anderen beiden ab.

Bei den Sicherheits-Spielen haben wir die *universelle Fälschung* als Fälschungsart modelliert, wobei wir auch die *existentielle Fälschung* in Betracht ziehen werden. Dafür können wir bei beiden Sicherheits-Spielen die ersten beiden Schritte wegdenken, da diese uns eine Nachricht vorgibt, die der Angreifer A fälschen muss. Die Signatur x^* ist dann somit eine Signatur, einer beliebigen Nachricht und nicht von H vorgegeben. Es muss aber dennoch gelten, dass x^* keine von A angefragte Signatur war. Die anderen beiden Fälschungsarten werden nicht weiter in Betracht gezogen, da das Auftreten eines *Total Breaks* sehr unwahrscheinlich ist. Bei einem *Total Break* kann man den *privaten Schlüssel* berechnen, was wiederum zu einem größeren Problem führt, da wir damit auch das dahinterstehende Verschlüsselungsverfahren gebrochen haben. Die *selektive Fälschung* wird nicht berücksichtigt, da es sich mit der *universellen Fälschung* ähnelt. Wenn man die Signatur einer selbst ausgewählten Nachricht berechnen kann, hat man damit auch einen Signieralgorithmus gefunden, der diese Signatur berechnet.

Wir gehen bei unseren Sicherheits-Spielen davon aus, dass der Angreifer Zugriff auf ein Orakel $O_{\texttt{SIG}}$ hat, welches für gegebene Nachrichten, die dazugehörigen Signaturen zurückgibt. Obwohl der Angreifer in der Praxis kein Zugriff auf so Orakel besitzt, kann der Angreifer dennoch Signaturen von seinem „Angriffsziel“ bekommen. Bei dem Versuch einer *universellen Fälschung* darf das Orakel selbstverständlich nicht die Signatur für die zu fälschende Nachricht m^* herausgeben.

Bei dem `Adaptive-Chosen-Message Angriff` darf der Angreifer er während des gesamten Angriffes einzelne Signaturanfragen, für Nachrichten tätigen. Somit hat er ebenso die Möglichkeit, dass er diese Anfragen gegebenenfalls auf die vorherigen Anfragen anpassen kann.

Laufzeit: Damit das Sicherheits-Spiel und die dazugehörigen Algorithmen in der Praxis verwendbar sind, sollten sie insgesamt in polynomieller Zeit laufen ($t \in O(x^k)$ mit $k \in \mathbb{N}$).

Erfolgswahrscheinlichkeit: Die Erfolgswahrscheinlichkeit gibt die Wahrscheinlichkeit an, mit der der Angreifer das Spiel gewinnt. Diese sollte außerdem nicht vernachlässigbar sein.

Abbildung 3.3.1.: Sicherheits-Spiel für einen `Directed-Chosen-Message Angriff`

Herausforderer H		**Angreifer** A		**Orakel** $O_{\texttt{SIG}}$
Erstelle Nachricht m^* für A				
	$\xrightarrow{m^*}$			
Erzeuge $((e, N), (d, N))$				
	$\xrightarrow{(N,e)}$			
		Erstelle $M = \{m_1, ..., m_n\}$ mit $m^* \notin M$		
			$\xrightarrow{M}$	
				Signiere jede Nachricht in M und erstelle dafür eine Signaturmenge $X = \{x_1, ..., x_n\}$, wobei (m_i, x_i), mit $i \in 1, ..., n$
			$\xleftarrow{X}$	
		A versucht die Signatur für m^* *zu fälschen*		
	$\xleftarrow{x^*}$			
Falls $V(m^*, x^*, e) == 1$: A hat gewonnen, sonst nicht.				

Abbildung 3.3.2.: Sicherheits-Spiel für einen `Adaptive-Chosen-Message Angriff`

Herausforderer H		**Angreifer** A		**Orakel** $O_{\texttt{SIG}}$
Erstelle Nachricht m^* für A				
	$\xrightarrow{m^*}$			
Erzeuge $((e,N),(d,N))$				
	$\xrightarrow{(N,e)}$			
		Erstelle Nachricht m_0 mit $m^* \neq m_0$		
			$\xrightarrow{m_0}$	
				Signiere Nachricht m_0
			$\xleftarrow{x_0}$	
		A versucht die Signatur für m^* zu fälschen		
			$\xrightarrow{\ldots}$	
			$\xleftarrow{\ldots}$	
		Erstelle Nachricht m_n mit $m^* \neq m_n$		
			$\xrightarrow{m_n}$	
				Signiere Nachricht m_n
			$\xleftarrow{x_n}$	
		A versucht die Signatur für m^* zu fälschen		
	$\xleftarrow{x^*}$			
Falls $V(m^*, x^*, e) == 1$: A hat gewonnen, sonst nicht.				

4. Einfache Signaturverfahren

Im Beispiel 4.1 haben wir grob eine mögliche Implementation von einem digitalen Signaturverfahren, basierend auf dem *RSA*-Verfahren, kennengelernt. Wir werden nun weitere Verfahren darstellen und auf deren Sicherheit eingehen.

4.1. RSA-DSS

RSA-DSS wurde bereits im Beispiel 4.1 beschrieben. Wichtig ist bei den folgenden digitalen Signaturverfahren, dass die Sicherheit dieser Verfahren stark gebunden an die Sicherheit der darunter verwendeten Verfahren liegt. In unserem Fall ist es die Annahme, dass es praktisch „unmöglich“ ist, für einen Wert $c = f(m)$, das Urbild $m = f^{-1}(c)$ zu finden [BR96].

Bei diesem Verfahren werden wir wiederum zeigen, warum dieses Verfahren nicht sicher ist ,um anschließend ein wichtiges Konzept für die Sicherheit der folgenden digitalen Signaturverfahren einzuführen.

Sicherheit von RSA-DSS

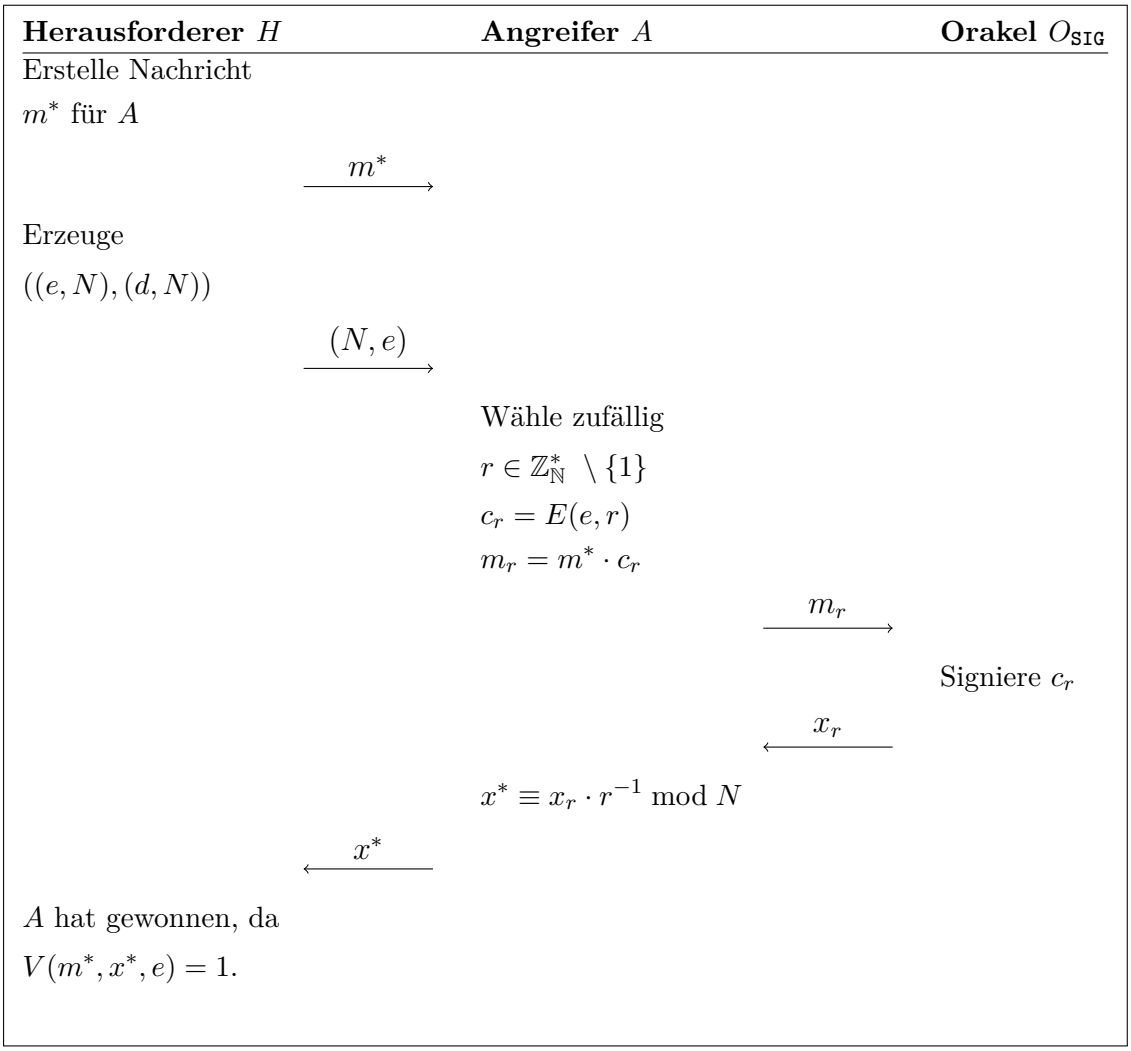
Wie in der Abbildung 4.1 zu sehen ist, bekommt der Angreifer A von dem Herausforderer H eine Nachricht m^*, die er fälschen muss. Dafür wählt der Angreifer einen zufälligen Wert $r \in \mathbb{Z}_{\mathbb{N}}^* \setminus \{1\}$ und berechnet $c_r = E(e, r)$. Danach berechnet er $m_r = m^* \cdot c_r$ und macht eine Orakelanfrage $x_r = O_{\texttt{SIG}}(m_r)$. Außerdem gilt $m^* \neq m_r$, da $r \neq 1 \bmod N$ und somit auch $c_r \neq 1$.

Da $r \in \mathbb{Z}_{\mathbb{N}}^*$ invertierbar ist, können wir die Signatur zu fälschende $x^* \equiv x_r \cdot r^{-1} \bmod N$ berechnen. [Jag18]

$$(x^*)^e \equiv (x_r \cdot r^{-1})^e \equiv x_r^e \cdot (r^e)^{-1} \equiv m_r \cdot c_r^{-1} \equiv m^* \cdot c_r \cdot c_r^{-1} \equiv m^* \bmod N$$

Somit ist es dem Angreifer A gelungen, eine *universelle Fälschung*, mit Hilfe eines *Generic-Chosen-Message Angriffes*, von dieser Nachricht zu erzeugen und somit

Abbildung 4.1.1.: Sicherheits-Spiel für einen `Directed-Chosen-Message Angriff` auf RSA-DSS



gewinnt A das Sicherheits-Spiel. Außerdem laufen diese Algorithmen in polynomieller Laufzeit, da die einzelnen Schritte mathematische Teiloperationen des eigentlichen RSA-Verfahrens sind. Der Angreifer A hat dabei Erfolgswahrscheinlichkeit von 1, da immer die gewünschte Signatur x^* berechnet werden kann.

5. Hashbasierte Signaturverfahren

Anhand des Sicherheitsbeweises vom *RSA-DSS*-Verfahrens haben wir gesehen, dass man durch einen vergleichsweise einfachen Angriff, das Verfahren brechen kann und somit eine universelle Fälschung erstellt.

Die hashbasierten Signaturverfahren behandeln dieses Sicherheitsproblem, indem man Einwegfunktionen, also kryptographische Hashfunktionen verwendet.

5.1. Hash-then-Sign-Verfahren

Wir zeigen hier eine fortgeschrittene Form vom *RSA-DSS*, indem wir vor dem Bilden der Signatur, den Hash h der zu signierenden Nachricht m berechnen und diesen Hash dann signieren. Damit der Empfänger die Signatur x dann verifizieren kann, muss er lediglich den Hash h' der Nachricht m bilden (die Hashfunktion ist ebenso öffentlich) und x mit dem *öffentlichen Schlüssel* des Senders „entschlüsseln“. Anschließend überprüft er, ob diese mit dem Hash h' übereinstimmt [BR96].

Definition 5 (Hash-then-Sign)**.** Sei HTS (*Hash-then-Sign*) ein erweitertes DSS mit folgenden, veränderten Eigenschaften:

Hashfunktion $H(m)$: $H(m)$ ist eine Hashfunktion, die eine Nachricht m als Eingabe bekommt und einen Hash y für diese Nachricht ausgibt.

Signatur: Der Hash h der Nachricht m wird als erstes durch die Hashfunktion $h = H(m)$ gebildet. S bekommt als Eingabeparameter den Hash der zu signierenden Nachricht m und d und gibt für h eine gültige *Signatur* x aus:

$$x = S(h, d).$$

Verifikation: V bekommt als Eingabeparameter eine Nachricht m, eine potenzielle

Signatur x und den *öffentlichen Schlüssel* e und gibt ein Bit b aus:

$$V(m,x,e) = \begin{cases} 1, & \text{falls } x \text{ gültige Signatur für } H(m) \\ 0, & \text{falls } x \text{ ungültige Signatur für } H(m). \end{cases}$$

Wir sagen, dass unser Signaturverfahren korrekt verifiziert, wenn:

$$V(m,x,e) = 1 \;\Leftrightarrow\; x \in S(H(m),d).$$

Sicherheit von hashbasierten Signaturverfahren

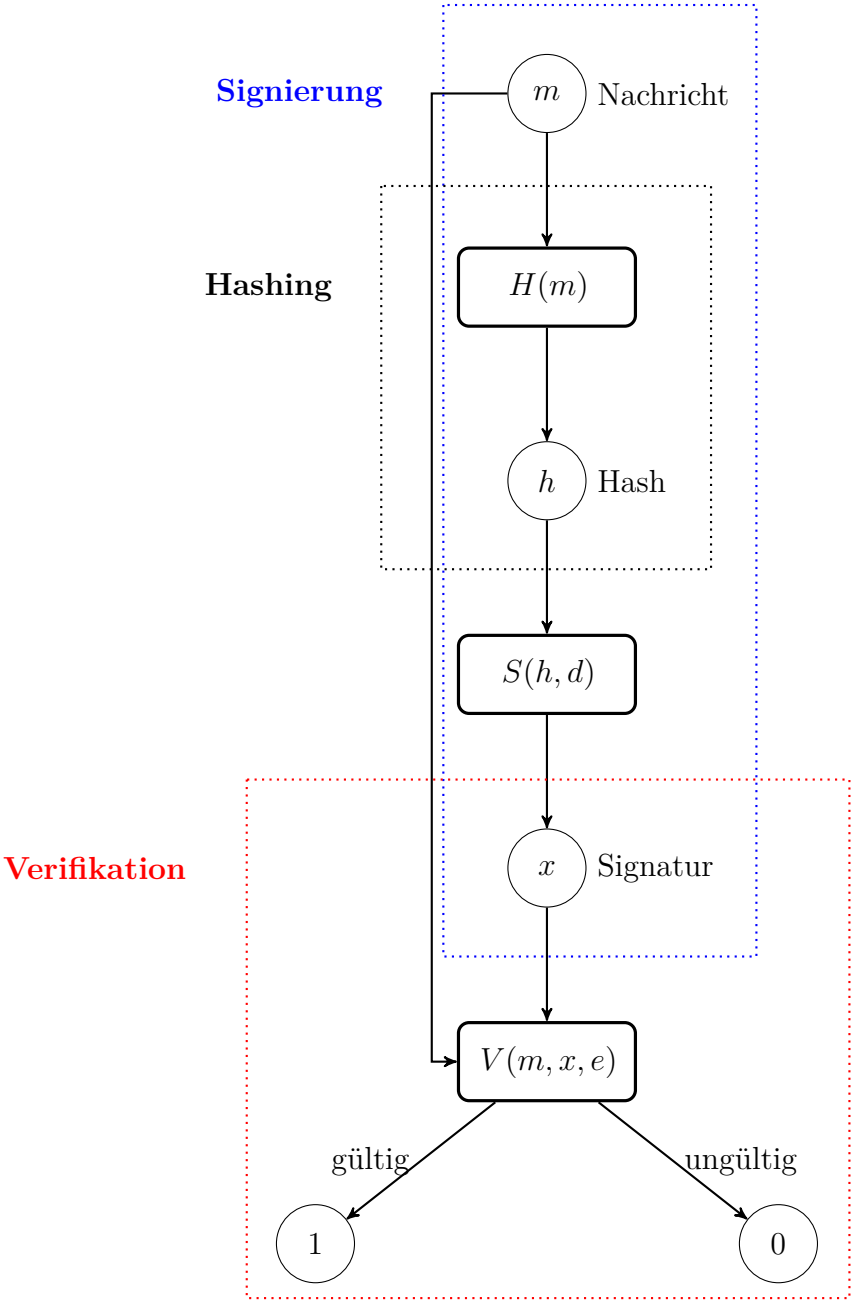
Laut Bellare und Rogaway [BR96] ist die Sicherheit vom *Hash-then-Sign*-Verfahren im Grunde genommen davon abhängig, wie sicher beziehungsweise *kollisionsresistent* das im Hintergrund verwendete Hashverfahren ist. Auch wenn es laut Bellare und Rogaway keinen ihnen bekannten Angriff auf das digitale Signaturverfahren gibt, ist dennoch zu beachten, dass bei Hashfunktionen das Problem beherrscht, dass sie nur eine vergleichsweise kleine, zusammenliegende Teilmenge von $\mathbb{Z}_{\mathbb{N}}^*$ sind.

Damit wir dennoch die Sicherheit beweisen, verwenden wir ein Verfahren, welches das *Random-Oracle-Modell* zu nutzen zieht.

5.2. RSA-FDH

Das *RSA-FDH*-Verfahren, auch genannt *RSA-Full-Domain-Hash*-Verfahren, ist ein digitales Signaturverfahren, welches das im vorherigen Abschnitt gezeigte *Hashing* verwendet und durch die Eigenschaft ihrer „idealen“ Hashfunktion eine genauere Sicherheitsdefinition ermöglicht. Die Hashfunktion bildet hierbei auf $\mathbb{Z}_N^*$ ab. Wir verwenden hier das in den Abschnitt 2.4 beschriebene *Random-Oracle-Model*, welches als Eingabe eine Nachricht m bekommt und uns für diese Nachricht einen Hash h ausgibt. Das *Random-Oracle* gibt dabei einen gleichverteilt zufälligen Wert zurück. Außerdem merkt er sich für diese Nachricht m, den zurückgegeben Hash h, sodass man bei erneuter Anfrage, von der gleichen Nachricht m, den gleichen Hashwert h zurückbekommt. In der Praxis besitzen wir so eine ideale Hashfunktion nicht. Wir verwenden aber Hashfunktionen, die heutzutage mit einer vernachlässigbaren Wahrscheinlichkeit als *kollisionsresistent* gelten. Ein solches Verfahren ist zum Beispiel `SHA3-512` [BR96].

Abbildung 5.1.1.: Signieren und Verifizieren einer Nachricht mit `Hash-then-Sign`



Für folgenden Sicherheitsbeweis führen wir Definitionen aus Bellare und Rogaway [BR96] ein:

I ist ein Invertierer für das *RSA*-Verfahren, falls er als Eingabe einen *öffentlichen Schlüssel* und eine Nachricht m (oder auch $H(m)$) bekommt und versucht das Inverse dieser Nachricht $f^{-1}(m)$ zu bestimmen. Er versucht also die *RSA*-Funktion zu invertieren.

Definition 6 (t-Invertierer)**.** Sei $t\colon \mathbb{N} \to \mathbb{N}$. Ein Invertieralgorithmus ist ein t-Invertierer, falls seine Laufzeit und seine Größe beschränkt ist durch ein $t(k)$.

Definition 7 ((t, ε)-sicher)**.** I bricht das *RSA*-Verfahren, wenn $\varepsilon\colon \mathbb{N} \to [0, 1]$ und I ein t-Invertierer ist. Für alle k gilt, dass die Erfolgswahrscheinlichkeit von I mindestens $\varepsilon(k)$ ist. *RSA* ist (t, ε)-sicher, falls kein Invertierer I existiert, welcher das *RSA*-Verfahren (t, ε)-bricht.

Sicherheit vom RSA-Full-Domain-Hash-Verfahren

Bellare und Rogaway haben in ihrem wissenschaftlichen Paper [BR96] gezeigt, dass man beim *RSA-FDH*-Verfahren keine *existentielle Fälschung*, mit Hilfe eines *Adaptive-Chosen-Message Angriff*, erstellen kann, unter der Annahme, dass das *Random-Oracle-Model* eingesetzt wird. Nach ihrem Satz 3.1 gilt:

Satz 7.1. Wenn das *RSA*-Verfahren (t', ε')-sicher ist, dann gilt für jedes q_{sig}, q_{hash}, dass das *RSA-FDH*-Verfahren $(t, q_{sig}, q_{hash}, \varepsilon)$-sicher ist, wobei:

$$t(k) = t' - [q_{hash}(k) + q_{sig}(k) + 1] \cdot O(k^3) \quad \text{und}$$

$$\varepsilon(k) = [q_{sig}(k) + q_{hash}(k) + 1] \cdot \varepsilon'(k).$$

Bewiesen wird dieser Satz durch eine Kontraposition. Aus:

$$\textit{RSA}\text{-Verfahren ist sicher} \Rightarrow \textit{RSA-FDH}\text{-Verfahren ist sicher}$$

folgt:

$$\textit{RSA-FDH}\text{-Verfahren ist unsicher} \Rightarrow \textit{RSA}\text{-Verfahren ist unsicher.}$$

Wir besitzen einen Invertierer I, welcher vom Herausforderer einen *öffentlichen Schlüssel* und einen Hashwert h^* bekommt und dafür die Signatur x^* herausfinden muss. Dafür modellieren wir einen Angreifer A, welcher für I das *RSA-FDH*-Verfahren brechen soll, damit I das *RSA*-Verfahren brechen kann. Dafür benötigt der Angreifer A aber Hashwerte. Diese *Random-Oracle*-Anfragen muss der Invertierer beantworten. I wird dabei, an einer Stelle, bei den Anfragen von A die Signatur h^* einbauen, damit A eine Fälschung für diese Signatur finden kann.

Beweis. (Nach Beweis von Satz 3.1 in [BR96]). Sei I ein Invertierer, welcher das

RSA-Verfahren bricht und A ein Angreifer, welcher das RSA-FDH-Verfahren bricht (siehe Abbildung 5.2).

Der Herausforderer H generiert ein *Schlüsselpaar* und wählt einen gleichverteilt zufälligen Wert m^* als Nachricht aus. Der Invertierer I bekommt den *öffentlichen Schlüssel* (N, e) und den Hash von h^* und versucht das Inverse von h^*, also $x^* = f^{-1}(h^*)$, zu bestimmen. Die Funktion f ist hierbei die *RSA*-Funktion.

Um dies durchzuführen, simuliert er einen Angreifer A, welcher als Eingabe (N, e) bekommt. A wird bei dem Versuch, eine *existentielle Fälschung* zu erzeugen, Hash- und Signaturanfragen an das *ROM* stellen, welche I beantworten wird. Dabei darf A maximal $q = q_{sig} + q_{hash}$ Anfragen stellen. I wählt einen gleichverteilt zufälligen Wert j aus $\{1, ..., q\}$. Es wird ein Inkrementierer i benötigt. Dieser wird mit 0 initialisiert. Außerdem besitzt I einen Speicher für die Simulierung vom *ROM* und der Speicherung von den Signaturen. $M = [m_0, ..., m_q]$ ist der Nachrichten-, $H = [h_0, ..., h_q]$ der Hash- und $R = [r_0, ..., r_q]$ der Signaturspeicher. Dabei gehören die Nachrichten, die Hashwerte und Signaturen, mit dem gleichen Index, zusammen.

I beantwortet die Anfragen von A wie folgt:

Bei einer Hashanfrage $H(m)$**:** i wird um 1 inkrementiert und $M[i] = m$ gesetzt.

> **Falls** $i \neq j$**:** I wählt einen gleichverteilt zufälligen Wert r_i, setzt $R[i] = r_i$, $H[i] = f(r_i)$ und gibt $H[i]$ zurück.
>
> **Falls** $i = j$**:** I setzt $H[i] = h^*$ und gibt $H[i]$ zurück an.

Bei einer Signaturanfrage $S(m, d)$**:** Es wird davon ausgegangen, dass I bei eine Signaturanfrage, auch eine Hashanfrage durchführt, falls diese nicht vorher getätigt wurde. Somit steht m schon in dem Speicher. I sucht den Index i, an dem m gespeichert ist und gibt den korrespondierenden Wert r_i zurück.

Schließlich wird A irgendwann halten und seine gefälschte Signatur (m_i, x^*) ausgeben. F gibt somit x^* aus. Falls für ein i gilt, dass (m_i, x^*) eine gültige Fälschung ist, ist mit einer Wahrscheinlichkeit von $\frac{1}{q}$, dass $i = j$ und somit auch $x = f^{-1}(y_i)$ korrekt.

Die Laufzeit von I ist Laufzeit von A und zusätzlich die Zeit, die für die Auswahl der h_i benötigt wird. Für jeden $h_i = f(r_i)$ Wert führt man eine RSA-Berechnung durch, die kubische Zeit benötigt.

Falls A eine Signaturanfrage tätigt, ohne vorher eine Signaturanfrage gemacht zu haben, tätigt I dies intern und führt danach die gewünschte Signaturanfrage durch.

Dass heißt, dass die effektive Anzahl der Hashanfragen höchstens $q_{hash} + q_{sig} + 1$ sein darf. □

Bei dem Beweis von Satz 7.1 wurde das *Random-Oracle-Model* verwendet. So eine Funktion ist in der Praxis schwer realisierbar, da wir keinen perfekten Zufall implementieren können. Es bleibt also die Frage, ob die Aussage von diesem Satz ausreichend ist, wenn man keinen *Random-Oracle-Model* verwendet.

Abbildung 5.2.1.: Sicherheits-Spiel für einen `Adaptive-Chosen-Message Angriff` auf RSA-FDH

Herausforderer H	**Inverter** I	**Angreifer** A
Erzeuge $((e,N),(d,N)$ und $H(m^*)=h^*$		
$\xrightarrow{((N,e),h^*)}$		
	$\xrightarrow{(N,e)}$	
		A versucht im weiterem Verlauf, eine Signatur für ein beliebiges m zu fälschen
	$\xleftarrow{H(m_0)}$	
	$ROM(m_0)=h_0.$	
	$\xrightarrow{h_0}$	
	$\xleftarrow{...}$	
	$\xrightarrow{...}$	
	$\xleftarrow{H(m_j)}$	
	Setze $ROM(m_j)=h^*.$	
	$\xrightarrow{h^*}$	
	$\xleftarrow{...}$	
	$\xrightarrow{...}$	
	$\xleftarrow{S(m_n,d)}$	
	$S(m_n,d)=x_n.$	
	$\xrightarrow{x_n}$	
	$\xleftarrow{(m_i,x^*)}$	
$\xleftarrow{x^*}$		
Falls $V(m^*,x^*,e)==1:$ A und F haben gewonnen, sonst nicht.		

6. Zusammenfassung und Ausblick

Digitale Signaturen sind, wie schon in der Einleitung erwähnt, ein wichtiger Bestandteil für die heutige, stark digitalisierte Welt. Sie sorgen für (1) die *Integrität*, (2) die *Authentizität* und (3) die *Nicht-Abstreitbarkeit* von Informationen beziehungsweise Nachrichten im Internet. Dafür ist es umso mehr wichtig, sicherzustellen, dass die im Hintergrund implementierten Verfahren auch sicher sind, also nicht durch Angreifer manipuliert werden können.

Mit diesem Thema haben wir uns in dieser Bachelorarbeit beschäftigt. Dazu haben wir uns Grundlagen angeeignet, welche notwendig waren, um schlussendlich die Sicherheit der digitalen Signaturverfahren zu beweisen oder auch zu widerlegen. Es wurden die Gebiete der Komplexitätstheorie, der Zufallszahlen und insbesondere die der Einwegfunktionen eingeführt.

Anschließend haben wir uns die Sicherheitsmodelle für digitale Signaturverfahren angeschaut. Dabei gab es verschiedenste Angriffsmöglichkeiten und für diese dementsprechend auch Fälschungsarten. Wir haben uns im Genauen jeweils zwei von den Angriffsmöglichkeiten und den Fälschungsarten betrachtet und dafür Sicherheits-Spiele erstellt. Anhand von diesen Sicherheits-Spielen, konnten wir die Sicherheit dieser digitalen Signaturverfahren beweisen.

Außerdem haben wir gesehen, dass wir die Sicherheitsbeweise nicht immer ohne Weiteres durchführen können. Bei dem *RSA-FDH*-Verfahren mussten wir die Annahme treffen, dass das *Random-Oracle-Modell* verwendet wurde. Dieses Modell ist in der Praxis schwer umzusetzen, da wir zum einen perfekte Zufallsfunktion benötigen, die uns bei Anfrage einen gleichverteilt zufälligen Wert zurückgibt. Zum anderen wäre das in der Theorie ein sehr großer Speicheraufwand, da wir die ganzen Abfragen speichern müssten.

Speziell sind wir in dieser Arbeit auf die Sicherheit der digitalen Signaturverfahren eingegangen, die das *RSA*-Verfahren implementiert haben. Es gibt außerhalb davon noch diverse andere digitale Signaturverfahren, die hier nicht behandelt wurden. In der Literatur von Bellare und Rogaway [BR96], die einen Sicherheitsbeweis für

das *RSA-FDH*-Verfahren gezeigt haben, gibt es ein weiteres Verfahren, welches eine bessere Sicherheit aufweist - das sogenannte *RSA-Probabilistic-Signature-Scheme*.

Ausblick

In der vergangen Zeit hat man gesehen, dass die Schlüsselgrößen der verwendeten *Public-Key-Kryptosystemen* immer größer wurden. Aber auch die Größe der erzeugten Hashwerte von Hashfunktion wurden stetig erhöht. Dies hat hauptsächlich den Grund, dass die Computer leistungsstärker wurden, aber auch erfolgreiche Angriffsmethoden gegen einzelne Verfahren entwickelt wurden.

Es gibt von Behörden empfohlene Schlüsselgrößen und auch Verfahren, welche derzeit als „sicher“ gelten. Dabei wird auch angegeben, in welchem Zeitraum diese verwendet werden sollten. Laut dem Bundesamt für Sicherheit in der Informationstechnik (BSI) liegt die empfohlene Schlüsselgröße, von 2018 - 2022, für das *RSA*-Verfahren bei 2000bit. Als Hashfunktionen werden unter anderem `SHA-256`, `SHA-512`, `SHA3-256`, `SHA-512` empfohlen. [Gir18]

A. Mathematische Grundlagen

Die folgenden Definitionen wurden aus dem Buch von Boehm [Boe16] entnommen und tragen zum mathematischen Grundverständnis für diese Arbeit bei.

A.1. Eulersche Phi-Funktion

Die *Eulersche-Phi-Funktion* ist wie folgt definiert:

$$\varphi(n) := |\{r \in \mathbb{Z} \mid 1 \geq r \geq n, ggT(r, n) = 1\}|.$$

Außerdem gilt nach Boehm [Boe16, Satz 6.2.1]:

Satz 7.2. Wenn n eine Primzahl ist:

$$\varphi(n) = (n - 1).$$

und nach Buchmann [Buc16, Satz 2.21]:

Satz 7.3 (Multiplikativität)**.** Wenn p, q teilerfremde Zahlen sind:

$$\varphi(p \cdot q) = \varphi(p) \cdot \varphi(q).$$

A.2. Prime Restklassengruppe

$\mathbb{Z}_N^*$ (oder auch notiert als $(\mathbb{Z}/n\mathbb{Z})^\times$) ist die prime Restklassengruppe. Für $n \in \mathbb{N}$ ist:

$$(\mathbb{Z}/n\mathbb{Z})^\times = \{\bar{a} \in \mathbb{Z}/n\mathbb{Z} \mid ggT(a, n) = 1\}.$$

Das heißt, dass es zu a ein multiplikatives Inverses existiert, wenn a teilerfremd zu n ist [Boe16, Satz 6.2.1].

A.3. Vernachlässigbar

Definition 8 (Vernachlässigbar nach [Jag18])**.** Eine Funktion $negl(k) : \mathbb{N} \to [0, 1]$ gilt als *vernachlässigbar*, falls es für jede Konstante $c \geq 0$ eine Zahl k_c existiert, sodass für alle $k > k_c$ gilt:

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

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[BG08] Bellare, Mihir ; Goldwasser, Shafi: *Lecture Notes on Cryptography*. Juli 2008

[Boe16] Boehm, J.: *Grundlagen der Algebra und Zahlentheorie*. Springer Berlin Heidelberg, 2016 (Springer-Lehrbuch). `https://books.google.de/books?id=LYvRrQEACAAJ`. – ISBN 9783662452288

[BR96] Bellare, Mihir ; Rogaway, Phillip: The Exact Security of Digital Signatures - HOw to Sign with RSA and Rabin. In: Maurer, Ueli M. (Hrsg.): *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding* Bd. 1070, Springer, 1996 (Lecture Notes in Computer Science). – ISBN 3–540–61186–X, 399–416

[BR05] Bellare, Mihir ; Rogaway, Phillip: Introduction to modern cryptography. In: *Ucsd Cse* (2005)

[Buc16] Buchmann, Johannes: *Einführung in die Kryptographie*. 6. Aufl. Berlin Heidelberg New York : Springer-Verlag, 2016. – ISBN 978–3–642–39775–2

[Gir18] Giry, Damien: *Keylength - Cryptographic Key Length Recommendation*. `https://www.keylength.com/`. Version: 2018

[GMR88] Goldwasser, Shafi ; Micali, Silvio ; Rivest, Ronald L.: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. In: *SIAM J. Comput.* 17 (1988), Nr. 2, 281–308. `http://dx.doi.org/10.1137/0217017`. – DOI 10.1137/0217017

[Jag18] Jager, Tibor: *Digitale Signaturen*. September 2018

[Knu97] Knuth, Donald E.: *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1997. – ISBN 0–201–89684–2