Vorlesung

Kryptologie

Marian Margraf

Version: SS 2014

Inhaltsverzeichnis

1 Einführung

Das Wort Kryptographie setzt sich zusammen aus den altgriechischen Wörtern kryptos (verborgen, geheim) und graphein (schreiben), ist also die Wissenschaft der Verschlüsselung von Informationen. Heute wird dieser Begriff aber allgemein auch für weitere Verfahren zum Thema Informationssicherheit, wie z.B. Datenauthentisierung, Instanzauthentisierung, Schlüsseleinigung usw. verwendet. Zusammen mit der Kryptoanalyse, d.h. dem Feld der Analyse kryptographischer Verfahren mit dem Ziel diese zu brechen oder aber die Sicherheit nachzuweisen, bildet sie das Gebiet der Kryptologie.

Kryptographische Verfahren werden eingesetzt, um unterschiedliche Sicherheitsziele zu erreichen. Dabei sind häufig mehrere Mechanismen zum Erreichen eines Ziels nötig. Um z. B. die Vertraulichkeit von über das Internet zu versendenden Daten zu gewährleisten, ist es nicht nur wichtig, die Daten zu verschlüsseln, auch der Schlüssel muss geheim gehalten werden, auf sicherem Wege ausgetauscht und zufällig gewählt worden sein, so dass ein potentieller Angreifer diesen praktisch nicht erraten kann.

Darüber hinaus ist es auch wichtig, dafür Sorge zu tragen, dass die eingesetzten Geräte und Programme keine kryptographischen Schwächen aufweisen, z. B. durch schlechte Implementierungen der kryptographischen Algorithmen.

Definition 1.1. *(Vertraulichkeit) Vertraulichkeit soll sicherstellen, dass Informationen unautorisierten Personen nicht zugänglich sind.*

Erreicht werden kann dies durch Verschlüsselung, wenn z. B. sensible Daten über das Internet versendet werden müssen. Eine weitere Methode ist das Versenden sensibler Daten durch vertrauenswürdige Kuriere, persönliche übergabe an berechtigte Personen und andere organisatorische Maßnahmen, wie z. B. geeignete Zugriffsschutzregeln.

Definition 1.2. *(Integrität) Unter Intergrität versteht man die Vollständigkeit und Unverfälschtheit der Daten für den Zeitraum, in dem sie von einer autorisierten Person erstellt, übertragen oder gespeichert wurden. Darin sind sowohl absichtliche als auch unabsichtliche, z. B. durch technische Fehler verursachte, Veränderungen enthalten.*

Kryptographische Mittel Integritätsverletzungen zu erkennen, sind z. B. Message-Authentication-Codes (MAC-Verfahren) oder elektronische Signaturen.

Definition 1.3. *(Authentizität)* (*i*) *Die Authentiziät von Daten ist gewährleistet, wenn der Urheber der Daten vom Empfänger eindeutig identifizierbar und seine Urheberschaft nachprüfbar ist. Dies beinhaltet auch die Integrität der Daten.*
(*ii*) *Ein Objekt oder Subjekt wird als authentisch bezeichnet, wenn dessen Echtheit und Glaubwürdigkeit anhand einer eindeutigen Identität und charakteristischer Eigenschaften überprüfbar ist.*

Erreicht werden kann dies durch den Einsatz von MAC-Verfahren, bei denen nur zwei Parteien den symmetrischen Schlüssel kennen oder durch geeignete Anwendung von Signaturverfahren. Die Prüfung der Authentizität wird auch als *Authentifikation* bezeichnet.

Zusätzlich zur Authentizität von Daten wird häufig auch gefordert, dass der Urheber der Daten im Nachhinein nicht abstreiten kann, diese erzeugt zu haben, es also gegenüber einer dritten Partei möglich ist, die Urheberschaft nachzuweisen.

Definition 1.4. *(Nichtabstreitbarkeit) Die Nichtabstreitbarkeit von Daten ist gewährleistet, wenn der Ersteller der Daten die Erzeugung im Nachhinein nicht abstreiten kann.*

Typische Verfahren, mit denen dieses Ziel erreicht werden kann, sind Signaturalgorithmen.

Definition 1.5. *(Verfügbarkeit) Ein IT-System gewährt* Verfügbarkeit*, wenn authentifizierte und autorisierte Subjekte in der Wahrnehmung ihrer Berechtigungen nicht unautorisiert beeinträchtigt werden können.*

Die Messung der Verfügbarkeit erfolgt üblicherweise nach folgender Formel:

$$\text{Verfügbarkeit} = \frac{\text{Gesamtlaufzeit} - \text{Ausfallzeit}}{\text{Gesamtlaufzeit}}$$

Bemerkung 1.6. Die Begriffe Integrität, Authentizität und Nichtabstreitbarkeit werden häufig nicht klar voneinander getrennt. Offensichtlich folgt aus der Nichtabstreitbarkeit sofort auch die Authentizität der Daten und aus der Authentizität auch die Integrität. Umgekehrt kann aber durch einen Message Authentication Code (siehe Unterabschnitt 5.1) zwar die Authentizität garantiert werden, nicht jedoch die Nichtabstreitbarkeit. Die mit dem Message Authentication Code unter Zuhilfenahme eines zwischen den Parteien vorab ausgetauschten symmetrischen Schlüssels berechnete Prüfsumme kann von jeder der Parteiin berechnet werden. Deswegen kann gegenüber Dritten kein unmittelbarer Nachweis erfolgen, wer die Prüfsumme (und somit auch die Daten) erzeugt hat.

Weitere Schutzziele, die insbesondere offene Kommunikationsnetze betreffen, sind

Definition 1.7. *(Anonymität, Pseudonymität)*

Anonymität: Personenbezogene Daten werden so verändert, dass diese nicht oder nur mit unverhältnismäßigem Aufwand einer Person zugeordnet werden können.

Pseudonymität: Personenbezogene Daten werden so verändert, dass diese nur unter Kenntnis der Zuordnungsvorschrift einer Person zugeordnet werden können.

Ziel der Kryptoanalyse ist es, aus einem bestehenden Geheimtext und eventuellen Zusatzinformationen den Klartext zu ermitteln. Einen Versuch bezeichnet man als *Angriff*. Wir unterscheiden folgende Angriffstypen:

Ciphertext-only: einer oder mehrere Chiffretexte sind bekannt.

Known-plaintext: Zu einigen Nachrichten ist sowohl Chiffrat als auch Klartext bekannt.

Chosen-plaintext: Der zu verschlüsselnde Text kann vom Angreifer gewählt werden.

Adaptive-chosen-plaintext: Spezialfall von oben, basierend auf dem Ergebnis der Verschlüsselung wird Klartext modifiziert.

Bei der Konstruktion kryptographischer Verfahren sollten die von Auguste Kerckoffs (ein niederländischer Linguist und Kryptograph) 1883 in seiner Schrift "La Cryptographie militaire" formulierten Prinzipien beachtet werden.

1. Wenn ein System nicht theoretisch beweisbar sicher ist, so sollte es praktisch sicher sein.
2. Das Design eines System sollte keine Geheimhaltung erfordern und sollte sich ohne Gefahr in den Händen des Feindes befinden können.
3. Der Schlüssel soll einfach zu merken und zu ändern sein.
4. Kryptogramme sollen per Telegraph übertragbar sein.
5. Ein Kryptosystem soll portabel sein und dessen Bedienung nicht mehrere Menschen erfordern.

6. Ein Kryptosystem muss einfach bedienbar sein.

Wir beginnen mit Verfahren zur Umsetzung des Schutzzieles Vertraulichkeit.

Sei Σ ein endliches Alphabet, Σ^* die Menge der endlichen Folgen in Σ und $\mathcal{X} \subseteq \Sigma^*$ eine Teilmenge von Σ^*. Die Elemente von $\mathcal{X}$ heißen *Klartexte*. Sei weiter $\mathcal{Y} \subseteq \Sigma^*$ ebenfalls eine Teilmenge von Σ^*, die sogenannten *Geheimtexte, Chiffretexte,* oder auch *Chiffrate*, und $\mathcal{Z}$ die Menge der *Schlüssel*.

Eine *Verschlüsselungsfunktion, Chiffrierer* ist eine injektive[1] Abbildung $f : \mathcal{X} \longrightarrow \mathcal{Y}$, f^{-1} heißt dann auch *Entschlüsselung, Dechiffrierer*.

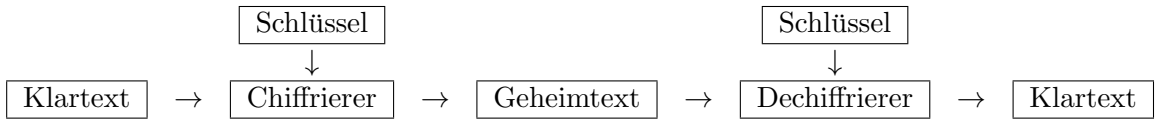
Definition 1.8. (Chiffreverfahren) *Eine Abbildung*

$$F : \mathcal{X} \times \mathcal{Z} \longrightarrow \mathcal{Y}$$

heißt Chiffrierverfahren, wenn $F(\cdot, k)$ für alle $k \in \mathcal{Z}$ eine Verschlüsselungsfunktion ist.

In der Literatur wird anstelle der Funktion F häufig eine Familie von Funktionen $(E_k : \mathcal{X} \longrightarrow \mathcal{Y})_{k\in\mathcal{Z}}$ (die sogenannten Verschlüsselungen) und eine Familie von Funktionen $(D_k : \mathcal{Y} \longrightarrow \mathcal{X})_{k\in\mathcal{Z}}$ (die Entschlüsselungen) mit $D_k \circ E_k = \mathrm{id}_{\mathcal{X}}$ betrachtet.

Das folgende Diagramm beschreibt das Zusammenwirken der Komponenten eines Chiffreverfahrens.



Wir behandeln in diesem Abschnitt nur *symmetrische Verfahren* auch *private-key Verfahren*, d.h. es ist sehr leicht, aus der Verschlüsselung E_k die Entschlüsselung D_k zu bestimmen. Im Gegensatz dazu gibt es auch nichtsymetrische, oder auch public key Verfahren, bei denen es selbst bei Kenntnis des Schlüssels k sehr schwierig ist, die Umkehrfunktion zu bestimmen.

Wir beginnen mit einem sehr einfachen Chiffrierverfahren, dem sogenannten *Caesarcipher*.

Beispiel 1.9. Sei $\Sigma = \{A, B, C, \cdots, Z\} = \{0, 1, 2, \ldots, 25\} = \mathbb{Z}_{26}$ ein Alphabet.

Für einen Schlüssel $k \in \mathbb{Z}_{26}$ ist

$$f_k : \mathbb{Z}_{26} \longrightarrow \mathbb{Z}_{26}; x \mapsto x + k \mod 26$$

eine Bijektion. Also ist die Abbildung

$$F : \Sigma^* \times \mathbb{Z}_{26} \longrightarrow \Sigma^*; ((x_1, \ldots, x_n), k) \mapsto f_k(x_1) \cdots f_k(x_n)$$

ein Chiffrierverfahren, das sogenannte Caesar-Verfahren. Beispielsweise gilt

$$F(\mathrm{CAESAR}, 3) = \mathrm{FDHVDU}.$$

Man sollte immer damit rechnen, dass ein möglicher Gegner das Chiffrierverfahren vollständig kennt (Kerckoff-Prinzip). Im obigen Beispiel bedeutet dies, dass die Zuordnung (oder auch Kodierung) $A \leftrightarrow 0, B \leftrightarrow 1, \ldots$ und die Funktion F bekannt ist. Die Sicherheit der Verfahren hängt also allein von der Geheimhaltung des Schlüssels k ab, der vorab ausgetauscht wurde.

Es ist offensichtlich, dass das obige Verfahren nicht besonders sicher ist. Liegt nämlich ein Ciphertext vor und man kennt den Schlüssel k nicht, so muss man lediglich 26 Schlüssel durchprobieren, um den Klartext zu rekonstruieren. Das durchprobieren aller Schlüssel nennt man auch *Exhaustion des Schlüsselraums*.

Als nächstes betrachten wir eine Verallgemeinerung des obigen Beispiels:

[1] Im allgemeinen müssen Verschlüsselungsfunktionen nicht injektiv sein, häufig verlangt man nur, dass die Wahrscheinlichkeit, dass zwei verschieden Elemente den gleichen Wert unter f haben sehr klein ist.

Beispiel 1.10. Wir betrachten wieder das Alphabet $\Sigma = \{A, B, C, \cdots, Z\} = \{0, 1, 2, \ldots, 25\} = \mathbb{Z}_{26}$ für die Menge der Klar- und Chiffretexte. Der Schlüsselraum sei gegeben durch

$$\mathcal{K} := \{(k_1, \ldots, k_{26}) \in \mathbb{Z}_{26}^{26}; k_1, \ldots, k_{26} \text{ sind paarweise verschieden }\}.$$

Das Chiffrierverfahren wird über die folgende Abbildung definiert:

$$f : \Sigma^* \times \mathcal{K} \longrightarrow \Sigma^*; (x, k) \mapsto (k_{x_1}, k_{x_2}, \ldots, k_{x_n}).$$

Ein Schlüssel k gibt also an, welche Buchstaben aus dem Klartext wie ersetzt werden, um den Chiffretext zu erhalten.

Für den Schlüssel $k = (3, 0, 16, 18, \ldots, 25, \ldots)$ (siehe auch Abbildung 1.1) lautet die Verschlüsselung von CAECAR $= (2, 0, 4, 2, 0, 17)$:

$$F((2, 0, 4, 2, 0, 17), k) = (k_2, k_0, k_4, k_2, k_0, k_{17}) = 16, 3, 18, 16, 3, 25.$$

Dies entspricht dem Wort Q,D,S,Q,D,Z.

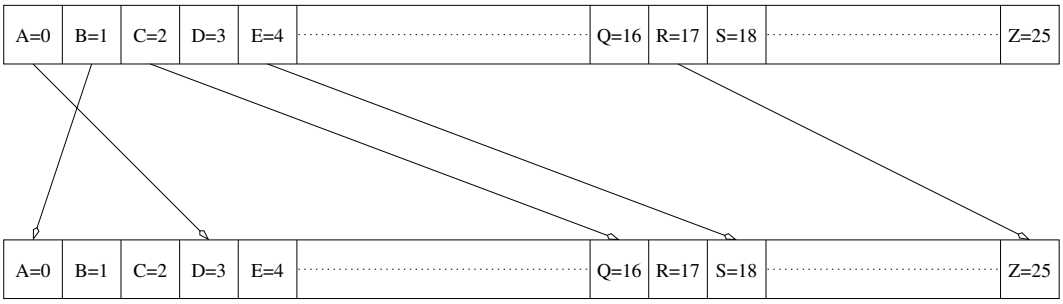


Abbildung 1.1: Darstellung eines Schlüssels für das modifizierte CAECAR-Verfahren.

Ist im obigen Beispiel der Schlüssel von der Form $k = (3, 4, 5, \ldots, 25, 0, 1, 2)$, dann ist die Verschlüsselung die gleiche wie in Beispiel 1.9 mit Schlüssel $k = 3$.

Bei diesem Chiffrierverfahren gibt es $26! = 2^{88}$ verschiedene Schlüssel, ein Durchsuchen des Schlüsselraums, um aus einem Chiffretext den Klartext zu ermitteln, ist mit heutigen Rechnern daher kaum möglich. Allerdings gibt es andere Angriffe auf dieses Verfahren, die tatsächlich zum Ziel führen.

Dazu betrachten wir zunächst die relativen Häufigkeiten von Buchstaben (Monogrammhäufigkeiten) in sinnvollen deutschen Texten, siehe die folgende Tabelle:

Da die Häufigkeiten der Anzahl der Buchstaben im Chiffretext übernommen werden, kann mit diesem Wissen der Klartext relativ einfach aus dem Chiffretext rekonstruiert werden. Dazu ermittelt man zunächst die Anzahl der Vorkommen der Buchstaben im Chiffretext. Der Buchstabe, der dort am häufigsten vorkommt, ist mit hoher Wahrscheinlichkeit ein E, der zweite ein N usw.

Die Idee, anstelle eines Buchstaben Paare von Buchstaben durch andere Paare zu ersetzen, ist ebenfalls nicht sicher genug. Auch hier lässt sich auf gleiche Weise mit Hilfe der bekannten Bigrammhäufigkeiten das Verfahren brechen.

Diese und ähnliche Methoden der Kryptoanalyse heißen auch statistische Angriffe.

Ein weiteres sehr einfaches, aber diesmal, wie wir gleich sehen werden, absolut sicheres Verschlüsselungsverfahren, das sogenannte *One-time-pad Private Key Cryptosystem*, werden wir nun kennenlernen.

Beispiel 1.11. Wir nehmen an, dass Texte, die verschlüsselt werden, durch Strings der Länge n kodiert sind, d.h. $\Sigma = \{(a_1, \ldots, a_n); a_i \in \{0, 1\}\}$. Weiter sei $\mathcal{X} = \Sigma$ (d.h. es wird, wie der Name

Platz	Buchstabe	Relative Häufigkeit
1.	E	17,40 %
2.	N	09,78 %
3.	I	07,55 %
4.	S	07,27 %
5.	R	07,00 %
6.	A	06,51 %
7.	T	06,15 %
8.	D	05,08 %
9.	H	04,76 %
10.	U	04,35 %
11.	L	03,44%
12.	C	03,06 %
13.	G	03,01 %
14.	M	02,53 %
15.	O	02,51 %
16.	B	01,89 %
17.	W	01,89 %
18.	F	01,66 %
19.	K	01,21 %
20.	Z	01,13 %
21.	P	00,79 %
22.	V	00,67 %
23.	ß	00,31 %
24.	J	00,27 %
25.	Y	00,04 %
26.	X	00,03 %
27.	Q	00,02 %

Tabelle 1.1: Relative Buchstabenhäufigkeiten in deutschen Texten

one-time-pad schon sagt, zu einem Schlüssel auch nur eine Nachricht der Länge n versendet) und $\mathcal{Z} = \mathcal{Y} = \mathcal{X}$.

Die Abbildung lautet dann

$$F : \mathcal{X} \times \mathcal{Z} \longrightarrow \mathcal{Y}; (x, k) \mapsto x \oplus k$$

dabei ist $\oplus$ die komponentenweise Addition modulo 2.

Liegt dann eine verschlüsselte Nachricht y vor, so bildet der Empfänger $x = y \oplus k$ und erhält so den Klartext zurück.

Es ist klar, dass ein möglicher Gegner nichts mit dem Chiffretext y anfangen kann, solange er den Schlüssel nicht kennt. Denn für jede Nachricht x und für jede mögliche Verschlüsselung y vom x gilt

$$\Pr(x \oplus k = y) = 1/2^n \tag{1.1}$$

für jeden möglichen Schlüssel k. D.h. vom Blickwinkel des Gegners aus ist die Verteilung der Strings y gleichmäßig und unabhängig von der aktuellen Nachricht x, und enthält somit keinerlei Informationen über x. Im informationstheoretischen Sinn ist solch ein Verschlüsselungssystem absolut sicher.

Dies ändert sich aber, wenn man mehrere Nachrichten mit demselben Schlüssel k verschlüsselt. Für zwei Nachrichten x_1 und x_2 und den zugehörigen Chiffretexten $y_i = x_i \oplus k,\ i = 1, 2$ gilt

$$y_1 \oplus y_2 = x_1 \oplus x_2.$$

Der Gegner kann also lernen. Ist zum Beispiel $x_1 = 0^n$, so ist es möglich, x_2 zu berechnen, ohne den Schlüssel k zu kennen.

Der amerikanische Mathematiker C. E. Shannon hat in den Jahren 1948 und 1949 Arbeiten zur Informationstheorie und deren Anwendung auf Verschlüsselungsverfahren veröffentlicht. Die Frage dabei ist, wie sicher kann ein Chiffriersystem gegenüber einem (idealisierten) Angreifer, der über unbeschränkte Ressourcen, wie zum Beispiel Rechenkapazität, Zeit, usw., verfügt, überhaupt sein?

Eine der wesentlichen Erkenntnisse aus dieser Arbeit ist:

Satz 1.12. *Ein Chiffriersystem kann nur dann absolut sicher sein, wenn die eingesetzte Schlüssellänge genauso groß ist wie der zu verschlüsselnde Klartext.*

Ein solches absolut sicheres Chiffriersystem haben wir schon kennen gelernt, dass one time pad Verfahren. Allerdings können solche Verfahren nur in sehr beschränktem Maße eingesetzt werden,

Die Sicherheit kryptographischer Verfahren hängt also primär von der Stärke der zugrunde liegenden Algorithmen ab. Weitere Faktoren für die Sicherheit sind natürlich die konkreten Implementierungen der Algorithmen und die Zuverlässigkeit eventueller Hintergrundsysteme, wie zum Beispiel benötigte Public Key Infrastrukturen für den sicheren Austausch von Zertifikaten. Wir werden in Anhang 11 kurz auf solche Hintergrundsysteme eingehen.

Insgesamt sollten heute alle genutzten kryptographischen Verfahren ein Sicherheitsniveau von 100 Bit erreichen. Das Sicherheitsniveau hängt dabei sowohl von der Stärke des Algorithmus als auch von der Schlüssellänge ab. Einfach ausgedrückt bedeutet ein Sicherheitsniveau von n Bit, dass ein Angreifer ca. 2^n Versuche benötigt, um den kryptographischen Algorithmus zu brechen (z.B. über ein Durchsuchen des Schlüsselraums oder die Anwendung kryptoanalytischer Verfahren).

Tabelle 1.2 zeigt die Schlüssellängen ausgewählter Algorithmen für das geforderte Sicherheitsniveau von 100 Bit.

Symmetrische Verfahren		**asymmetrische Verfahren**		
Verschlüsselung	MAC	RSA	DSA	ECDSA
100	100	2048	224/2048	200

Tabelle 1.2: Beispiele für Schlüssellängen für ein Sicherheitsniveau von mindestens 100 Bit

Die folgenden Beispiele sollen einen Eindruck von großen Zahlen vermitteln.

- Anzahl der Atome der Erde 2^{170}
- Anzahl der Atome der Sonne 2^{190}
- Anzahl der Atome in unserer Galaxis 2^{223}
- Anzahl der Atome im Weltall (ohne dunkle Materie) 2^{265}
- Zeit bis zur nächsten Eiszeit 2^{14} Jahre
- Zeit bis die Sonne zur Nova wird 2^{30} Jahre
- Alter der Erde 2^{30} Jahre
- Alter des Universums 2^{34} Jahre

- Wenn das Universum geschlossen ist (d.h. die Expansion des Universums kommt irgendwann zum Stillstand und geht wieder in eine Kontraktion über):
  - Lebensdauer ca. 2^{37} Jahre
- Wenn das Universum offen ist (die Expansion des Universums dauert unendlich fort):
  - Zeit bis sich die Planeten aus dem Sonnensystem lösen 2^{50} Jahre
  - Zeit bis sich die Sterne aus den Galaxienverbänden lösen 2^{64} Jahre

Würde ein Computer, der in jeder Sekunde $2 * 10^9$ das heißt 2000000000 Verschlüsselungen berechnen kann, alle 2^{128} möglichen Schlüssel durchprobieren, so bräuchte er:

$$\frac{\text{Anzahl der möglichen Schlüssel}}{\text{Verschlüsselungen je Sekunde * Sekunden pro Jahr}} = \frac{2^{128}}{(2 * 10^9 s^{-1}) * (3.15 * 10^7 s/\text{Jahr})}.$$

Das sind circa $5.3 * 10^{21}$ Jahre.

Wenn das Weltall geschlossen ist, lässt sich diese Rechnung nicht mehr bis zum Ende durchführen.

Häufig kann ein kryptographisches Verfahren für verschiedene Anwendungen eingesetzt werden, so zum Beispiel Signaturverfahren zur Datenauthentisierung und zur Instanzauthentisierung. In diesem Fall muss darauf geachtet werden, dass für die unterschiedlichen Anwendungen jeweils verschiedene Schlüssel eingesetzt werden.

Ein weiteres Beispiel sind symmetrische Schlüssel zur Verschlüsselung und symmetrischen Datenauthentisierung. Hier ist es technisch möglich, einen Schlüssel für beide Verfahren zu benutzen. Allerdings muss bei konkreten Implementierungen dafür gesorgt werden, dass für beide Verfahren jeweils verschiedene Schlüssel eingesetzt werden, die insbesondere nicht voneinander ableitbar sind, siehe auch Abschnitt 10.1.

Leider ist die Kryptographie keine exakte Wissenschaft wie die Mathematik. Die Sicherheit vieler asymmetrischer Verfahren beruhen auf der **vermuteten** Schwierigkeit bestimmter Probleme (Faktorisierung ganzer Zahlen, Diskretes Logarithmusproblem in endlichen Körpern). Bei der Konstruktion symmetrischer Verfahren werden zwar heute bestimmte Designkriterien zugrunde gelegt (Substitution, Permutation), von einer beweisbaren Sicherheit, die ohne Grundvoraussetzungen auskommt, ist die Kryptographie, in Bezug auf effiziente Algorithmen, aber weit entfernt. Zudem können auf den ersten Blick kleine Unachtsamkeiten große negative Folgen haben. Wir werden im Laufe der Vorlesung an einigen Stellen darauf eingehen. Aus diesem Grund sollten insbesondere die folgenden Regeln, zusätzlich zum bereits kennen gelernten Kerckoffschen Prinzip, beachtet werden:

1. Trenne wo du trennen kannst: Kryptographische Schlüssel sollten jeweils nur für ein Verfahren eingesetzt werden.
2. Kryptographische Algorithmen sollten probabilistisch sein: Der Chiffretext einer Nachricht sollte nie allein von der Nachricht und dem Schlüssel, sondern auch von anderen Parametern abhängen. Gleiches gilt für Signaturen, Authentisierungsverfahren, Schlüsselaustauschverfahren usw.
3. Nutze allgemein anerkannte Algorithmen: Versuche nicht, selbst entwickelte Algorithmen einzusetzen.
4. Sei achtsam bei Zufallswerten: Bei der Generierung kryptographischer Schlüssel und anderer notwendiger Systemparameter werden **echte** Zufallszahlen benötigt. Kleinste Abweichungen können verheerende Folgen haben.

Weitere konkrete Hinweise werden, so nötig, in den entsprechenden Abschnitten angegeben. Bei der Auswahl geeigneter kryptographischer Verfahren folgen wir den Empfehlungen des BSI, siehe auch [47].

2 Symmetrische Verschlüsselungsverfahren

Symmetrische Verschlüsselungsverfahren dienen der Gewährleistung der Vertraulichkeit von Daten, die zum Beispiel über einen öffentlichen Kanal, wie Telefon oder Internet, ausgetauscht werden. Die Authentizität bzw. Integrität der Daten wird dadurch nicht gewährleistet, für einen Integritätsschutz siehe Kapitel 5 und Abschnitt 10.1.

Wir behandeln zunächst symmetrische Verfahren, d.h. Verfahren, in denen der Verschlüsselungs- und der Entschlüsselungsschlüssel gleich sind (im Gegensatz zu asymmetrischen Verfahren, bei denen aus dem öffentlichen Schlüssel der geheime Schlüssel ohne zusätzliche Informationen praktisch nicht berechnet werden kann). Für asymmetrische Verschlüsselungsverfahren, die in der Praxis lediglich als Schlüsselaustauschverfahren eingesetzt werden, siehe Kapitel 3.

2.1 Blockchiffren

Eine Blockchiffre ist ein Algorithmus, der einen Klartext fester Bitlänge (z. B. 128 Bit) mittels eines Schlüssels zu einem Chiffretext gleicher Bitlänge verschlüsselt. Diese Bitlänge heißt auch *Blockgröße* der Chiffre. Für die Verschlüsselung längerer Klartexte werden sogenannte Betriebsarten eingesetzt, siehe 2.2.

Chiffrierverfahren müssen zum Teil große Datenmengen (z.B. Satellitenbilder) in kurzer Zeit verschlüsseln. Aus diesem Grund müssen die Verfahren so konstruiert sein, dass sie möglichst schnell in Soft- und Hardware (z.B. Chipkarten mit beschränkten Ressourcen) arbeiten. Eine Chiffre sollte daher aus relativ einfachen Funktionen, die effizient implementiert werden können, zusammengesetzt sein. Auf der anderen Seite muss das eingesetzte Verfahren natürlich sicher sein, d.h. ohne Kenntnis des verwendeten Schlüssel darf der Klartext nicht aus dem Chiffretext rekonstruiert werden können.

Shannon hat die folgenden Konstruktionsprinzipien für Chiffrierverfahren angegeben:

1. Diffusion (Durchmischung): Die Bits des Klartextblocks werden über den gesamten Block verschmiert. Quantitativ ausdrücken kann man das durch den Lawinen-Effekt (englisch: avalanche effect):
   a) Jedes Bit des Geheimtextblocks hängt von jedem Bit des Klartextblocks ab.
   b) Bei Änderung eines Klartextbits ändern sich ca. 50 Prozent der Geheimtextbits.

   Grundbausteine zur Erreichung von Diffusion sind Permutationen (siehe unten).
2. Konfusion (Komplexität des Zusammenhangs):
   a) Die Beziehung zwischen Klartextblock und Geheimtextblock soll möglichst kompliziert sein (insbesondere hochgradig nichtlinear).
   b) Ähnlich komplex soll auch die Abhängigkeit des Geheimtextblocks vom Schlüssel sein, insbesondere sollen sich bei Änderung eines Schlüsselbits möglichst viele Geheimtextbits ändern, und zwar möglichst unvorhersagbar. Es soll für den Angreifer unmöglich sein zu erkennen, dass er einen Schlüssel fast richtig geraten hat.

   Grundbausteine hierfür sind vor allem Substitutionen (nichtlineare Abbildungen).

3. Wiederholte Anwendung von Permutation und Konfusion: Ziel dieses Konstruktionsprinzips ist, durch mehrfache abwechselnde Hintereinanderausführung der zwei Funktion die Komplexität der Chiffre zu erhöhen. Zudem sollte hier frühzeitig der Schlüssel eingebracht werden, um die Forderung 2.b) zu erfüllen.

$$\underbrace{((\cdot \oplus k_n) \circ P \circ S)}_{\text{Runde } n} \circ \cdots \circ \underbrace{((\cdot \oplus k_n) \circ P \circ S)}_{\text{Runde 2}} \circ \underbrace{((\cdot \oplus k_n) \circ P \circ S)}_{\text{Runde 1}}$$

Um statistische Angriffe, wie in Beispiel 1.10 dargestellt, zu verhindern, sollten zusätzlich nur noch Blockchiffren eingesetzt werden, deren Blockgröße mindestens 128 Bit beträgt.

Permutationen sind spezielle lineare Abbildungen, die ein gegebenes $n-$Tupel $(x_1, \ldots, x_n)$ mittels einer bijektiven Abbildung $\pi : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\}$ auf das $n-$Tupel $(x_{\pi(1)}, \ldots, x_{\pi(n)})$ abbilden, siehe Abbildung 2.1.

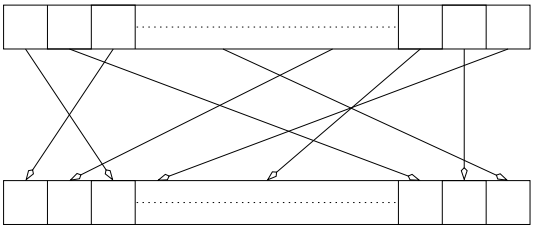


Tabelle 2.1: Schematische Darstellung einer Permutation

Diese Abbildungen lassen sich sehr effizient implementieren, z.B. mit dem folgenden Pseudocode:

1. int array x=$[x_1, \ldots, x_n]$; (der Klartext)
2. int array y=new array[]; (das Ergebnis der Abbildung)
3. int array π =$[\pi_1, \ldots, \pi_{128}]$; (beschreibt die bijektive Abbildung)
4. for $i = 1$ to n do
5. y[i]=x[π_i];
6. end if
7. return y;

Permutationen allein reichen für eine sichere Chiffrierfunktion nicht aus. Da diese Abbildungen linear sind, lässt sich aus der Kenntnis verschiedener Klartexte und Chiffretexte, die mit dem selben Schlüssel verarbeitet wurden, ein lineares Gleichungssystem erstellen, um den eingesetzten Schlüssel zu berechnen. Diese sind aber effizient lösbar. Es ist also erforderlich, auch nichtlineare Abbildungen einzusetzen (die für die Konfusion sorgen). Diese erzeugen dann nichtlineare Gleichungssysteme, von denen wir wissen, dass diese deutlich schwerer zu lösen sind.

Substitutionen Diese nichtlinearen Abbildungen sind aber deutlich schwieriger zu implementieren. Eine nichtlineare Funktion, die auf der gesamten Blockgröße operiert (hier 128 Bit), ist nicht umsetzbar. Aus diesem Grund beschränkt man sich häufig auf Teilblöcke, meist mit einer Länge von 8 oder 16 Bit und führt die nichtlineare Abbildung parallel aus. Um solch eine Funktion effizient zu berechnen, werden die 8 bzw. 16-bit Blöcke als natürliche Zahlen $0, 1, \ldots 2^8 - 1(, \ldots, 2^{16} - 1)$ interpretiert und ein array genutzt, siehe folgenden Pseudocode:

1. int x; (der Klartext)
2. int y; (das Ergebnis der Abbildung)

3. int array S_Box[] =$[s_1, \ldots, s_{256}]$; (beschreibt die bijektive Abbildung)
4. y= S_Box[x];
5. return y;

Schlüsseladdition Nach erfolgter Anwendung einer Substitution und einer Permutation wird auf den berechneten Block $y_1, \ldots, y_{128}$ ein Rundenschlüssel bitweise modulo zwei addiert (modulo zwei heißt $0 + 0 = 0, 1 + 0 = 1$ und $1 + 1 = 0$, der Operator für diese Addition wird auch als XOR oder $\oplus$ bezeichnet). Hierzu wird der eingesetzte Schlüssel (meist 128 Bit) mittels eines geeigneten Verfahrens expandiert und Rundenschlüssel abgeleitet.

Insgesamt erhält man mit dieser Konstruktionsmethode die in Abbildung 2.2 dargestellte Chiffre.

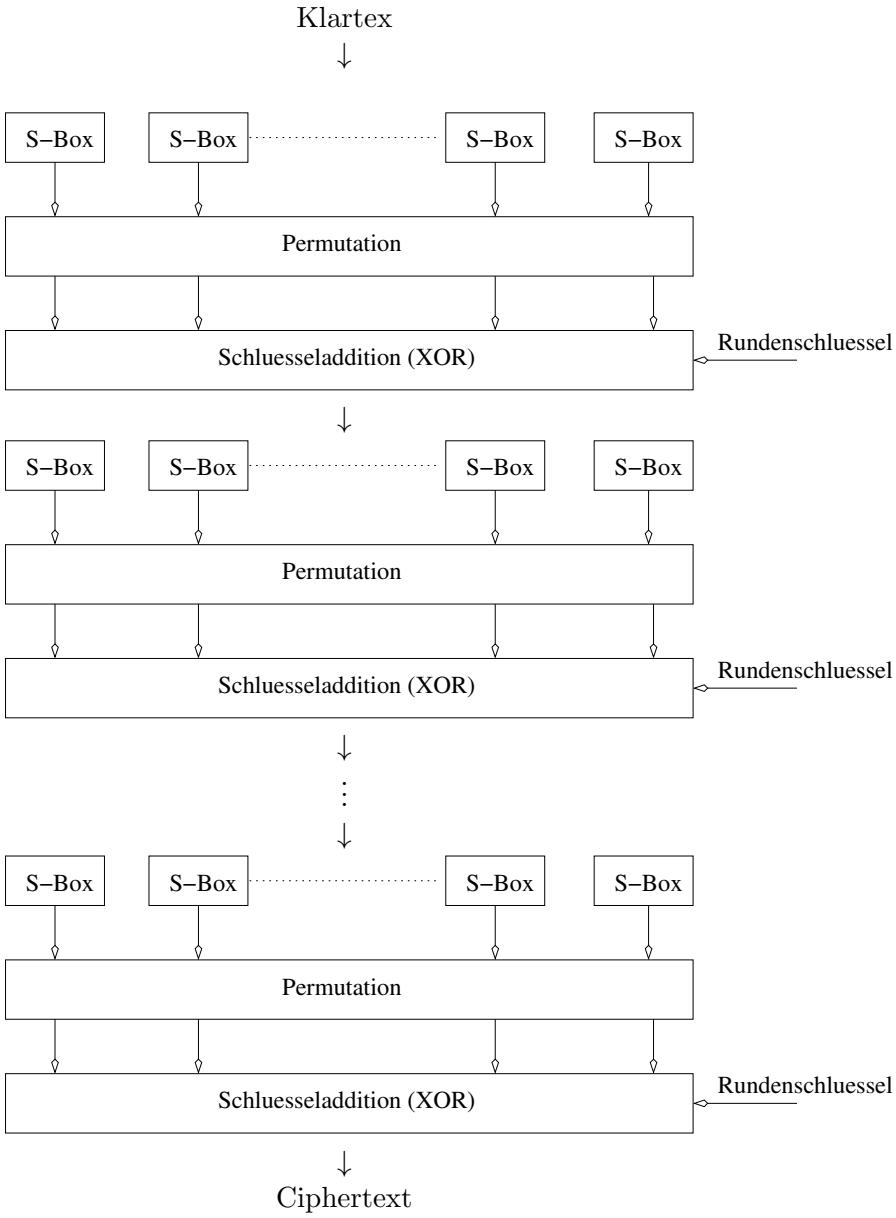


Tabelle 2.2: Aufbau eines Substitutions-Permutations-Netzwerkes (SPN)

Wir nennen eine nach den obigen Prinzipien konstruierte Chiffre auch Substitutions-Permutations-Netzwerk (SPN). Das Chiffrierverfahren Advanced Encryption Algorithm

(AES) ist ein typisches Beispiel für ein SPN. Darüber hinaus gibt es weitere Konstruktionsverfahren, wie z.B. Feistel-Chiffren. Ein Beispiel für solch ein Verfahren ist der Vorgänger von AES, der Data Encyption Standard (DES).

Nach dem derzeitigen Wissensstand gelten die folgenden Blockchiffren als sicher:

1. AES-128, AES-192, AES-256, siehe [30],
2. SERPENT-128, SERPENT-192, SERPENT-256, siehe [3], und
3. Twofish-128, Twofish-192, Twofish-256, siehe [59].

Tabelle 2.3: Empfohlene Blockchiffren

2.2 Betriebsarten

Wie in Abschnitt 2.1 bereits beschrieben, werden für Klartexte, deren Bitlänge die Blockgröße der eingesetzten Blockchiffre übersteigen, sogenannte Betriebsarten verwendet. Dazu wird zunächst der Klartext in Blöcke partitioniert, deren Bitlänge der Blockgröße der eingesetzten Blockchiffre entsprechen. Der letzte Block hat eventuell eine kleinerer Bitgröße und muss entsprechend aufgefüllt werden, siehe Abschnitt 2.4 für geeignete Verfahren.

Die einfachste Möglichkeit, den Klartext zu verschlüsseln, ist nun, jeden Klartextblock mit dem selben Schlüssel zu verschlüsseln (diese Betriebsart heißt auch Electronic Code Book (ECB), siehe Abbildung 2.4).

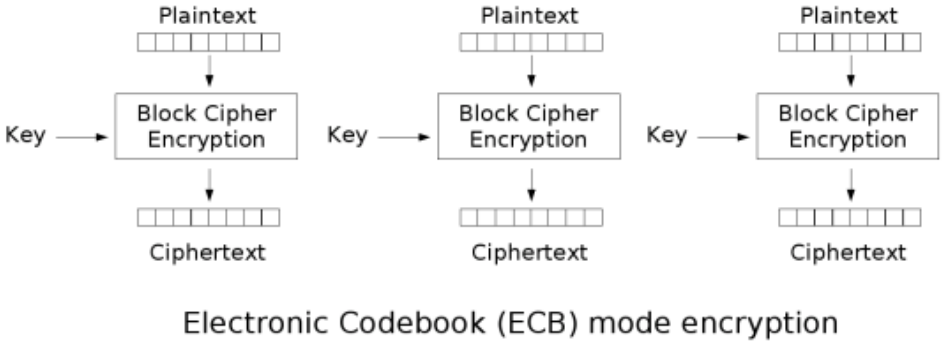


Tabelle 2.4: Schematische Darstellung der Betriebsart ECB

Dies führt aber dazu, dass gleiche Klartextblöcke zu gleichen Chiffretextblöcken verschlüsselt werden. Der Chiffretext liefert damit zumindest Informationen über die Struktur des Klartextes. Um dies zu verhindern, sollte der $n-$te Chiffreblock nicht nur vom $n-$ten Klartextblock und dem eingesetzten Schlüssel abhängen, sondern von einem weiteren Wert, wie zum Beispiel dem $(n-1)-$ten Chiffretextblock oder einem Zähler (auch Counter genannt).

Die Betriebsart Cipher Block Chaining weißt die Schwäche der ECB-Betriebsart nicht auf. Hier hängt der $n-$te Ciphertext nicht nur vom Schlüssel, sondern auch vom $(n-1)-$ten Cipherblock ab. Genauer wird, bevor der $n-$te Klartextblock m_n verschlüsselt wird, zunächst der $(n-1)-$te Ciphertextblock c_{n-1} auf m_n aufaddiert und der so erhaltene Block $m_n \oplus c_{n-1}$ verschlüsselt, siehe

auch Abbildung 2.5. Für die Verschlüsselung des ersten Klartextblockes (hier liegt ja noch kein Ciphertextblock vor, der aufaddiert werden kann) wird ein sogenannter Initialisierungsvektor (IV) benötigt.

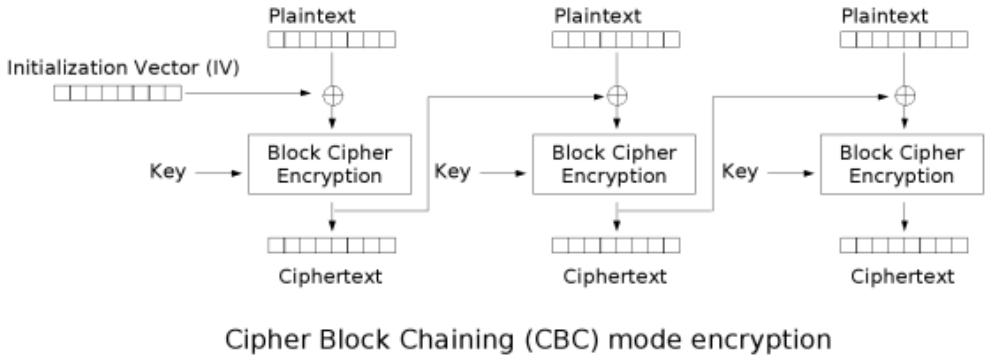


Tabelle 2.5: Schematische Darstellung der Betriebsart CBC

Der Counter Mode arbeitet, wie der Name schon sagt, mit einem Zähler als Initialisierungsvektor. Hier wird eine nonce (Abkürzung für number used once, eine zufällige, nur einmal benutzte Zeichenfolge) mit einem Zähler verknüpft, siehe Abbildung 2.6.

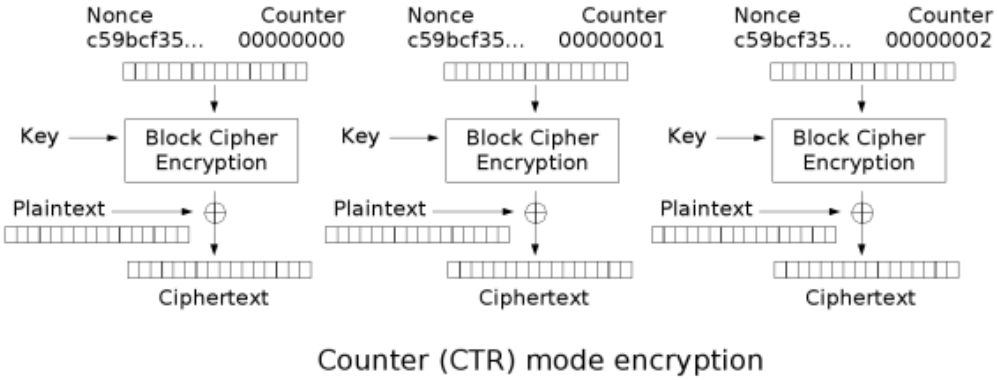


Tabelle 2.6: Schematische Darstellung der Betriebsart CTR

Folgende Betriebsarten sind für die unter 2.1 aufgeführten Blockchiffren geeignet:

1. Cipher-Block Chaining (CBC), siehe [51],
2. Galois-Counter-Mode (GCM), siehe [54], und
3. Counter Mode (CTR), siehe [51].

Tabelle 2.7: Empfohlene Betriebsarten für Blockchiffren

Bemerkung 2.1. Der Galois-Counter-Mode (GCM) liefert zusätzlich eine Datenauthentisierung und wird entsprechend genauer in Abschnitt 10.1 behandelt.

2.3 Initialisierungsvektoren

Für die unter 2.2 aufgeführten Betriebsarten werden Initialisierungsvektoren benötigt. Diese Werte müssen zwar nicht vertraulich behandelt werden, unterliegen aber bestimmten Nebenbedingungen, die in der folgenden Tabelle zusammengefasst sind.

1. Für CBC: Es sind nur unvorhersagbare Initialisierungsvektoren zu verwenden (vgl. Übungsaufgabe 3).
2. Für GCM: Die Zählerstände im Zähleranteil des Initialisierungvektors dürfen sich bei gleichem Schlüssel nicht wiederholen, vergleiche Bemerkung 10.2.
3. Für CTR: Die Zählerstände dürfen sich bei gleichem Schlüssel nicht wiederholen.

Tabelle 2.8: Empfohlene Verfahren für die Erzeugung von Initialisierungsvektoren

2.4 Paddingverfahren

Wie bereits in Abschnitt 2.2 erläutert, kann es bei der Partitionierung eines zu verschlüsselnden Klartextes geschehen, dass der letzte Klartextblock kleiner als die Blockgröße der eingesetzten Chiffre ist. Das Auffüllen dieses letzten Blocks zu der geforderten Größe heißt auch *Padding*.

Folgende Paddingverfahren werden empfohlen:

1. ISO-Padding, siehe [46] und [54],
2. Padding gemäß [56], und
3. ESP-Padding, siehe [57].

Tabelle 2.9: Empfohlene Paddingverfahren für Blockchiffren

2.5 Aufgaben

Übungsaufgabe 1. Berechnen Sie die Größe des Speicherplatzes eines Arrays, das eine bijektive Funktion auf $\mathbb{F}^{128}$ beschreibt.

Übungsaufgabe 2. Beschreiben Sie die Entschlüsselung bei der Nutzung der Betriebsarten CBC und CTR.

Übungsaufgabe 3. Formulieren Sie Angriffe bei der Nutzung der Betriebsart CBC mit vorhersagbaren Initialisierungsvektoren (Hinweis: Watermark-Angriff).

3 Asymmetrische Verschlüsselungsverfahren

Asymmetrische Verschlüsselungsverfahren werden in der Praxis lediglich zur Übertragung symmetrischer Schlüssel eingesetzt, siehe auch Kapitel 7. Die zu verschlüsselnde Nachricht (d. h. der symmetrische Schlüssel) wird mit dem öffentlichen Schlüssel des Empfängers verschlüsselt. Der Empfänger kann dann die Verschlüsselung mit dem zum öffentlichen Schlüssel assoziierten geheimen Schlüssel wieder entschlüsseln. Dabei darf es praktisch nicht möglich sein, den Klartext ohne Kenntnis des geheimen Schlüssels aus dem Chiffretext zu rekonstruieren. Dies impliziert insbesondere, dass der geheime Schlüssel nicht aus dem öffentlichen Schlüssel konstruiert werden kann. Um eine Zuordnung des öffentlichen Schlüssels zum Besitzer des zugehörigen geheimen Schlüssels zu garantieren, wird üblicherweise eine Public Key Infrastruktur benötigt, siehe Abschnitt 11.

Für die Spezifizierung von asymmetrischen Verschlüsselungsverfahren sind folgende Algorithmen festzulegen:

1. Ein Algorithmus zur Generierung von Schlüsselpaaren (inklusive Systemparameter).
2. Ein Algorithmus zum Verschlüsseln und ein Algorithmus zum Entschlüsseln der Daten.

Zusätzlich geben wir Empfehlungen für minimale Schlüssellängen an.

3.1 Empfohlene asymmetrische Verschlüsselungsverfahren

Tabelle 3.1 gibt einen Überblick über die im Folgenden empfohlenen asymmetrischen Verschlüsselungsverfahren. Wir behandeln im Rahmen dieser Vorlesung aber nur das RSA-Verfahren genauer.

1. RSA, siehe [58],
2. DLIES, siehe [32], und
3. ECIES, siehe [32].

Tabelle 3.1: Empfohlene asymmetrische Verschlüsselungsverfahren

Bemerkung 3.1. Bei der Auswahl der empfohlenen asymmetrischen Verschlüsselungsverfahren wurde darauf geachtet, dass lediglich sogenannte probabilistische Algorithmen[1] zum Einsatz kommen. Hier wird also bei jeder Berechnung eines Chiffretextes ein **neuer** Zufallswert benötigt. Anforderungen an diese Zufallswerte werden in den entsprechenden Abschnitten angegeben.

[1] Der RSA-Algorithmus selbst ist nicht probabilistisch, dafür aber das hier empfohlenen Paddingverfahren zu RSA.

3.2 RSA-Verschlüsselungsverfahren

Das RSA-Verfahren wurde 1977 von Ronald L. Rivest, Adi Shamir und Leonard Adleman am MIT entwickelt. Es galt damals als das erste asymmetrische Verschlüsselungsverfahren.

Die Sicherheit des Verfahrens beruht auf der vermuteten Schwierigkeit des Faktorisierungsproblems ganzer Zahlen.

Faktorisierungsproblem:
Gegeben: Eine zusammengesetzte natürliche Zahl $n = p \cdot q \in \mathbb{N}$ mit zwei Primzahlen p, q.
Lösung: Finde die beiden Primfaktoren p und q.

Bevor wir das Verfahren erläutern, benötigen wir einige mathematische Grundlagen.

Mathematische Grundlagen Sei $n \in \mathbb{N}$ eine natürliche Zahl. Dann bezeichnet

$$\mathbb{Z}_n := \{0, 1, \ldots, n-1\}$$

die Menge der ersten $n-1$ natürlichen Zahlen. Für jede natürliche Zahl $a \in \mathbb{Z}$ sei $r = a \bmod n$ die eindeutige Zahl aus $\mathbb{Z}_n$, für die es ein $k \in \mathbb{Z}$ mit $r = a - k \cdot n$ gibt. Beispielsweise ist $3 = 23 \bmod 10$.

Die Menge $\mathbb{Z}_n$ bildet zusammen mit den Operationen

$$\begin{aligned} a + b &:= a + b \bmod n \\ a \cdot b &:= a \cdot b \bmod n \end{aligned}$$

einen kommutativen Ring mit Einselement, d.h. $(\mathbb{Z}_n, +)$ ist eine abelsche Gruppe, es gilt das Distributivgesetz und $(\mathbb{Z}_n \backslash \{0\}, \cdot)$ ist eine abelsche Halbgruppe mit Einselement. Mit $\mathbb{Z}_n^*$ bezeichnen wir die Menge der Elemente aus $\mathbb{Z}_n$, die ein multiplikativ Inverses besitzen. Beispielsweise gilt

- $\mathbb{Z}_{10}^* = \{1, 3, 7, 9\}$.
- $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$ für jede Primzahl $p \in \mathbb{N}$ (dies folgt sofort aus Satz 3.2, insbesondere ist in diesem Fall $\mathbb{Z}_p^*$ bzgl. der Multiplikation eine Gruppe).
- $|\mathbb{Z}_{pq}^*| = (p-1)(q-1)$ für zwei Primzahlen $p, q \in \mathbb{N}$ (auch diese Aussage folgt sofort aus Satz 3.2).

Der Ring $(\mathbb{Z}_p, +, \cdot)$ bildet also einen Körper für jede Primzahl p, d.h. zusätzlich zu den Ringeigenschaften ist $(\mathbb{Z}_p \backslash \{0\} = \mathbb{Z}_p^*, \cdot)$ eine Gruppe.

Der folgende Satz charakterisiert die Menge der Elemente aus $\mathbb{Z}_n$, die ein multiplikativ inverses Element besitzen.

Satz 3.2. *Ein Element $a \in \mathbb{Z}_n$ ist genau dann multiplikativ invertierbar in $\mathbb{Z}_n$, wenn* $\mathrm{ggt}(a, n) = 1$ *gilt. Dabei bezeichnet* $\mathrm{ggt}(a, n)$ *den größten gemeinsamen Teiler von a und n (*$\mathrm{ggt}(0, 0) := 0$*).*

Beweis. (i) Sei zunächst $x \in \mathbb{Z}_n$ das Inverse von a (d.h. $ax = 1 \bmod n$) und sei $g = \mathrm{ggt}(a, n)$. Dann ist g ein Teiler von n und damit auch von $ax - 1$. Da g auch a und damit ax teilt, muss $g = 1$ sein.

(ii) Gelte umgekehrt $\mathrm{ggt}(a, n) = 1$. Wir zeigen gleich, dass es für je zwei ganze Zahlen $a, n \in \mathbb{Z}$ zwei ganze Zahlen $x, y \in \mathbb{Z}$ mit $xa + yn = g := \mathrm{ggt}(a, n)$ gibt. Hieraus folgt also $xa + yn = 1$ und damit $xa = 1 - yn$, d.h. $xa = 1 \bmod n$.

Für $a = n = 0$ folgt die Behauptung sofort. Sei also im Folgenden mindestens eine der beiden Zahlen ungleich Null. Sei weiter $I = a\mathbb{Z} + n\mathbb{Z}$ und g die kleinste positive Zahl in I. Wir zeigen $I = g\mathbb{Z}$. Sei dazu $c \in I\backslash\{0\}$. Mit $q := \lfloor c/g \rfloor$ und $r := a - gq$ folgt dann $c = gq + r$ und $0 \leq r < g$. Weiter ist $r = c - gq \in I$. Da g die kleinste positive Zahl in I ist, folgt $r = 0$ und damit $c = gq \in g\mathbb{Z}$. Wir haben also $I \subseteq g\mathbb{Z}$ gezeigt. $g\mathbb{Z} \subseteq I$ gilt offensichtlich.

Wir müssen jetzt nur noch $g = \text{ggt}(a, n)$ zeigen. Da sowohl a als auch n aus I sind, folgt aus $I = g\mathbb{Z}$, dass g ein Teiler von a und n ist. Da weiter $g \in I$, gibt es $x, y \in \mathbb{Z}$ mit $g = xa + yn$. Ein gemeinsamer Teiler von a und n ist somit auch immer ein Teiler von g, d.h. g ist der größte gemeinsame Teiler. □

Zwei Zahlen $a, n \in \mathbb{Z}$ mit $\text{ggt}(a, n) = 1$ heißen auch teilerfremd. Mit $\phi(n) := |\mathbb{Z}_n^*|$ bezeichnen wir die Anzahl der Elemente in $\mathbb{Z}_n$, die teilerfremd zu n sind, also ein multiplikatives Inverses in $\mathbb{Z}_n$ besitzen. $\phi(n)$ ist auch als Eulerzahl bekannt. Ist p eine Primzahl, so gilt, wie wir bereits gesehen haben, $\phi(p) = p - 1$.

Wir sind nun in der Lage, den kleinen Satz von Fermat zu beweisen.

Satz 3.3. (Kleiner Satz von Fermat) *Sind $a, n \in \mathbb{Z}$ teilerfremd, dann gilt $a^{\phi(n)} = 1 \bmod n$.*

Die Aussage folgt sofort aus dem folgenden Lemma.

Lemma 3.4. *(i) Sei G eine endliche abelsche Gruppe und $g \in G$. Dann gilt $g^{|G|} = 1$.*
(ii) Für jede Untergruppe U von G ist $|U|$ ein Teiler von $|G|$.

Beweis. *(i)* Die Abbildung $G \longrightarrow G; h \mapsto gh$ ist bijektiv. Damit gilt

$$\prod_{h \in G} h = \prod_{h \in G} gh = g^{|G|} \prod_{h \in G} h.$$

Teilen beider Seiten durch $\prod_{h \in G} h$ zeigt die Behauptung.

(ii) Sei $g \in G$. Dann ist $gU := \{gu; u \in U\}$. Für zwei Elemente $g, g' \in G$ gilt: entweder ist $gU = g'U$, oder $gU \cap g'U = \emptyset$. Diese Aussage beweisen wir gleich im Anschluss. Da weiter $|gU| = |U|$ und $\bigcup_{g \in G} = G$, folgt die Behauptung
Warum gilt nun $gU = g'U$ oder $gU \cap g'U = \emptyset$? Sei h ein gemeinsames Element von gU und $g'U$. Dann gibt es $u, u' \in U$ so, dass

$$gu \quad = g'u'. \tag{3.1}$$

Für jedes Element $k \in gU$ existiert $v \in U$ mit $k = gv$. Aus Gleichung 3.1 folgt dann $k = g' \cdot (u'u^{-1}v)$. Da $(u'u^{-1}v) \in U$, hat k also auch eine Darstellung in $g'U$. Damit haben wir $gU \subseteq g'U$ gezeigt. Die restliche Behauptung zeigt man analog. □

Wir sind nun in der Lage, das RSA-Verschlüsselungsverfahren zu erläutern.

Schlüsselgenerierung

1. Wähle zwei verschiedene Primzahlen p und q zufällig und unabhängig voneinander.
2. Wähle den öffentlichen Exponenten $e \in \mathbb{N}$ (e für Encryption=Verschlüsselung) unter der Nebenbedingung
$$\text{ggT}(e, \phi(n)) = 1.$$

3. Berechne den geheimen Exponenten $d \in \mathbb{N}$ (d für Decryption=Entschlüsselung) in Abhängigkeit von e unter der Nebenbedingung

$$e \cdot d = 1 \bmod \phi(n).$$

(Nach Satz 3.2 existiert $d \in \mathbb{N}$ mit dieser Eigenschaft tatsächlich.)

Mit $n = p \cdot q$ (dem sogenannten Modul) ist dann (n, e) der öffentliche Schlüssel und d der geheime Schlüssel. Weiter müssen natürlich auch die beiden Primzahlen p, q und $\phi(n)$ geheim gehalten werden, da sonst jeder aus dem öffentlichen Schlüssel (n, e) wie unter Punkt 3. den geheimen Exponenten berechnen kann.

Verschlüsselung Um eine Nachricht $m < n$ zu verschlüsseln, berechnet der Absender

$$c = m^e \mod n.$$

Entschlüsselung Um den Ciphertext c zu entschlüsseln, berechnet der Empfänger

$$m = c^d \mod n.$$

Für die Ver- und Entschlüsselung siehe auch [58].

Der folgende Satz zeigt, dass das Verfahren korrekt ist, d.h. dass wir mit der Entschlüsselung tatsächlich die ursprünglich verschlüsselte Nachricht erhalten.

Satz 3.5. *Sind n, e und d wie oben gewählt, dann gilt für alle $m \in \mathbb{Z}_n$*

$$(m^e)^d \bmod n = m.$$

Beweis. Wegen $e \cdot d = 1 \bmod \phi(n)$ existiert $k \in \mathbb{N}$ mit $ed = 1 + k\phi(n)$. Da weiter $\phi(n) = (p-1)(q-1)$, gilt

$$(m^e)^d = m^{ed} = m^{1+k\phi(n)} = m \cdot ((m^{p-1})^{q-1})^k.$$

Gilt nun $\mathrm{ggt}(m, p) = 1$, so folgt aus dem kleinen Satz von Fermat (Satz 3.3) $(m^e)^d = m \bmod p$. Ist $\mathrm{ggt}(m, p) \neq 1$, so ist p ein Teiler von m und es gilt ebenfalls $(m^e)^d = m \bmod p$.

Genauso zeigt man $(m^e)^d = m \bmod q$. Damit existieren $s, t \in \mathbb{N}$ mit

$$m + sp = (m^e)^d = m + tq,$$

also $sp = tq$. Damit ist $s = tq/p$ eine natürliche Zahl. Da $p \neq q$ und p und q Primzahlen sind, folgt, dass p ein Teiler von t ist. Damit existiert $r \in \mathbb{N}$ mit $t = rp$. Insgesamt folgt daraus

$$(m^e)^d = m + tq = m + rpq = m + rn$$

und damit $(m^e)^d = m \bmod n$. □

Ohne zusätzliche Absicherungen ist das RSA-Verfahren aber nicht sicher, wie das folgende Beispiel zeigt.

Beispiel 3.6. Die RSA-Funktion $x \mapsto x^d \mod n$ ist multiplikativ, d.h. $(xy)^d = x^d y^d \mod n$. Dank dieser Eigenschaft ist eine choosen ciphertext Attacke möglich: Angenommen, ein Angreifer möchte den Ciphertext $c = m^e \bmod n$ entschlüsseln. Er generiert den String $c \cdot r^e \bmod n$ (beachte, dass e öffentlich ist) und fragt den Besitzer des geheimen Schlüssels d an, diese Nachricht zu entschlüsseln. Das Resultat ist mr, durch Multiplikation mit dem Inversen von r modulo n erhält der Angreifer die Nachricht m.

Ein weiterer Angriff funktioniert bei der Wahl kleiner Verschlüsselungsexponenten e ohne die Mithilfe des Schlüsselinhabers, wie das folgende Beispiel zeigt (siehe auch [20]):

Beispiel 3.7. Sei $e = 3$. Kennt ein Angreifer die Ciphertexte $c_1 = m^3 \bmod n$ und $c_2 = (m + 1)^3 \bmod n$, so kann er ohne Kenntnis des Entschlüsselungsschlüssels d die Klartexte bestimmen. Es gilt nämlich:

$$\begin{aligned}\frac{c_2 + 2c_1 - 1}{c_2 - c_1 + 2} &= \frac{(m+1)^3 + 2m^3 - 1}{(m+1)^3 - m^3 + 2} \\ &= \frac{(m^3 + 3m + 3m^2 + 1) + 2m^3 - 1}{(m^3 + 3m + 3m^2 + 1) - m^3 + 2} \\ &= \frac{3m^3 + 3m + 3m^2}{3m + 3m^2 + 3} \\ &= m\end{aligned}$$

Der im obigen Beispiel beschriebene Angriff lässt sich auf die Situationen erweitern, in denen $m = \alpha \cdot m_1 + \beta$ gilt und für beliebige Exponenten e anwenden. In diesem Fall beträgt die Laufzeit des Angriffes $\mathcal{O}(e^2)$, ist also nur für kleine Exponenten praktikabel.

Eine Übersicht über weitere Angriffe gegen RSA liefert [8].

Um diesen und ähnliche Angriffe zu verhindern, kann das RSA-Verfahren nur mit einigen Modifikationen eingesetzt werden. Wir werden im Laufe der Vorlesung sehen, das asymmetrische Verfahren (wie z.B. für die Verschlüsselung, Authentisierung und Signatur) in der Regel probabilistische Algorithmen sind, d.h. das Ergebnis hängt nicht nur von der Eingabe (dem zu verschlüsselnden oder zu signierenden Text ab), sondern darüber hinaus auch von einem Zufallswert. Die grundlegende Idee ist dabei, dass die kryptographische Sicherheit durch eine oder mehrere der folgenden Methoden erhöht wird:

- Die effektive Größe des Klartextraumes wird vergrößert.
- Indem der Klartext auf viele Weisen (eben in Abhängigkeit vom zusätzlich eingebrachten Zufall) in den Chiffretext überführt wird, lassen sich Angriffe ausschließen oder zumindest deren Effizienz deutlich verringern.
- Weiter werden statistischen Angriffe verhindert oder deutlich erschwert, indem die a-priori-Wahrscheinlichkeitsverteilung der Eingaben der Gleichverteilung angenähert wird.

Insbesondere bei dem in diesem Abschnitt behandelten RSA-Verfahren führt der Zufall dazu, dass der multiplikative Zusammenhang zwischen zwei Klartexten nicht mehr zu einem ausnutzbaren Zusammenhang zwischen den Chiffretexten (und umgekehrt) führt.

Das eingesetzte probabilistische Verfahren ist dabei sorgfältig zu wählen. Wir erläutern im Folgenden das OAEP (Optimal asymmetric encryption padding), das heute allgemein als sicher gilt, siehe Abbildung 3.2.

Eine Nachricht m der Länge $n - k_0 - k_1$ wird wie folgt aufgefüllt:

- Die Nachricht m wird zunächst mit k_1 Nullen zu $m||0 \cdots 0$ aufgefüllt.
- r ist ein Zufallswert der Länge k_0.
- Die Funktion G erweitert r auf $n - k_1$ Bits.
- Die Funktion H reduziert $x := (m||0 \cdot 0) \oplus G(r))$ auf k_0 Bits.
- $Y := H(x) \oplus r$.
- Die Ausgabe ist $x||y$.

Dabei sollte die Länge der Zufallszahl mindestens 100 Bit betragen und G und H sollten sichere Hashfunktionen sein, siehe Abschnitt 4.

Um aus der Formatierung $x||y$ die Nachricht m zu erhalten:

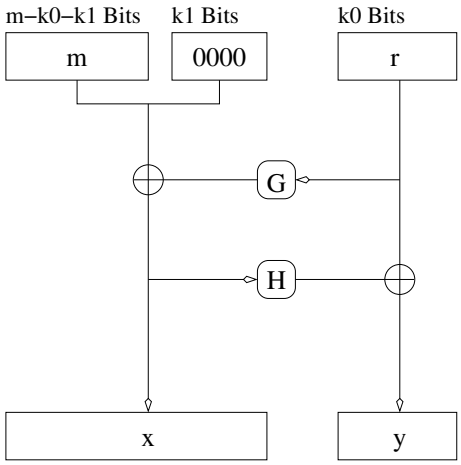


Tabelle 3.2: Das Formatierungsverfahren OAEP

- Berechne $r = y \oplus H(x)$ und
- $m||0\cdots 0 = x \oplus G(r)$.

EME-OAEP, siehe [58].

Tabelle 3.3: Empfohlenes Formatierungsverfahren für den RSA-Verschlüsselungsalgorithmus

Schlüssellänge Um die Sicherheit des RSA-Verfahrens beurteilen und insbesondere die notwendige Schlüssellänge angeben zu können, diskutieren wir zunächst die zu diesem asymmetrischen Verschlüsselungsverfahren in Bezug stehenden Probleme.

1. Zu gegebenen n, e und der verschlüsselten Nachricht $c = m^e \bmod n$ berechne m.
2. Zu gegebenen n, e berechne d mit $ed = 1 \bmod \phi(n)$.
3. Zu gegebenen $n = pq$ finde die beiden Primzahlen p und q.

Offensichtlich lässt sich aus der Faktorisierung des Moduls n bei bekanntem öffentlichen Exponenten e der geheime Exponent d berechnen, dieser Schritt wird ja auch im Rahmen der Schlüsselgenerierung durchgeführt. D.h., gibt es einen effizienten Algorithmus für das Faktorisierungsproblem, dann gibt es auch einen effizienten Algorithmus für Problem 2. Weiter lässt sich auch aus einem effizienten Algorithmus für Problem 2 ein effizienter Algorithmus für 1 konstruieren, aus der Kenntnis des Entschlüsselungsschlüssels d lässt sich natürlich die Nachricht entschlüsseln.

Wie sieht es nun mit den Umkehrungen aus? Es ist bekannt, dass sich auch aus einem effizienten Algorithmus für Problem 2 einer für Problem 3 konstruieren lässt, die beiden Probleme sind also gleich schwer. Bis heute weiß man allerdings nicht, ob dies auch für das Problem 1 gilt, d.h. dieses Problem könnte leichter zu lösen sein als Problem 2 (und wäre damit auch leichter als Problem 3), man kennt aber derzeit keine andere Möglichkeit, die Nachricht $c = m^e \bmod n$ zu entschlüsseln, als c mit d zu potenzieren.

Die Sicherheit des RSA-Verfahrens beruht also nach derzeitigem Kenntnisstand auf der vermuteten Schwierigkeit des Faktorisierens großer Zahlen. Der heute schnellste Faktorisierungsalgorithmus hat eine Laufzeit von $\mathcal{O}(e^{\sqrt{\log n \log\log n}})$. Daraus ergibt sich eine minimale Schlüssellänge, d.h. die Größe des Moduls n von 2048 Bit.

4 Hashfunktionen

Hashfunktionen bilden einen Bitstring $m \in \{0,1\}^*$ beliebiger Länge auf einen Bitstring $h \in \{0,1\}^n$ fester Länge $n \in \mathbb{N}$ ab. Diese Funktionen spielen in vielen kryptographischen Verfahren eine große Rolle, so zum Beispiel bei der Ableitung kryptographischer Schlüssel oder bei der Datenauthentisierung.

Hashfunktionen $H : \{0,1\}^* \longrightarrow \{0,1\}^n$, die in kryptographischen Verfahren eingesetzt werden, müssen je nach Anwendung die folgenden drei Bedingungen erfüllen:

Preimage resistance: Für gegebenes $h \in \{0,1\}^n$ ist es praktisch unmöglich einen Wert $m \in \{0,1\}^*$ mit $H(m) = h$ zu finden.

Second preimage resistance: Für gegebenes $m \in \{0,1\}^*$ ist es praktisch unmöglich einen Wert $m' \in \{0,1\}^* \backslash \{m\}$ mit $H(m) = H(m')$ zu finden.

Collision resistance: Es ist praktisch unmöglich zwei Werte $m, m' \in \{0,1\}^*$ so zu finden, dass $m \neq m'$ und $H(m) = H(m')$ gilt.

Eine Hashfunktion H, die alle obigen Bedingungen erfüllt, heißt *kryptographisch stark*.

Der Begriff "praktisch unmöglich" bedeutet hier, dass es keinen Algorithmus A gibt, der eine effiziente Laufzeit hat und den Wert bzw. die Werte berechnet (Sicherheitsniveau 100 Bit).

Ähnlich wie bei Blockchiffren müssen Hashfunktionen zum Teil große Datenmengen effizient verarbeiten können. Daher sind auch diese Funktionen aus einfach zu berechnenden Grundfunktionen zusammengesetzt.

Die meisten Hashfunktionen folgen der Merkle-Damgård-Konstruktion, siehe Abbildung 4.1. Hierzu wird eine Kompressionsfunktion

$$f : \mathbb{F}_2^a \times \mathbb{F}_2^b \longrightarrow \mathbb{F}_2^b$$

genutzt, die die oben formulierten Eigenschaften für Hashfunktionen erfüllen muss. Daraus lässt sich dann eine Hashfunktion konstruieren, die Bitstrings beliebiger Länge als Eingabe akzeptiert. Dazu wird die zu hashende Nachricht m zunächst in Blocks der Länge b aufgeteilt (der letzte Teil muss eventuell aufgefüllt werden).

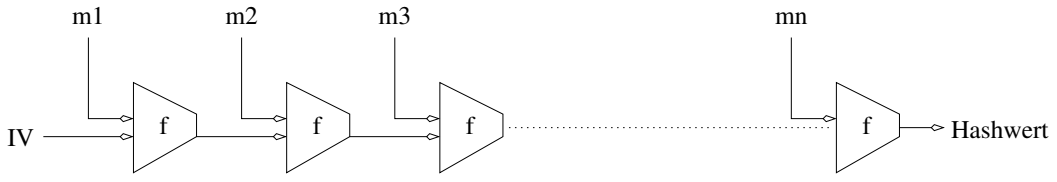


Tabelle 4.1: Hashfunktion nach Merkle-Damgård

Als Kompressionsfunktionen können beispielsweise die in Abschnitt 2 behandelten Blockchiffren genutzt werden. Die einfachste Art, eine Blockchiffre

$$E : \mathbb{F}_2^{128} \times \mathbb{F}_2^{128} \longrightarrow \mathbb{F}^{128}$$

als Kompressionsfunktion zu nutzen, sieht wie folgt aus: Der IV und die Bilder der Funktion werden als Schlüssel interpretiert, die Nachrichtenblöcke für die Hashfunktion als Nachrichtenblöcke

der Chiffre, d.h. für eine Nachricht $m = (m_1, \ldots, m_n)$ ist

$$h(m) = E(m_n, \ldots E(m_2, E(m_1, IV)) \cdots).$$

Diese Konstruktion ist aber unsicher, wie der folgende Satz zeigt.

Satz 4.1. *Die oben konstruierte Funktion h erfüllt selbst bei einer sicheren Blockchiffre nicht die Eigenschaft second preimage resistance.*

Beweis. Wir zeigen, dass wir für einen beliebigen Wert $h \in \mathbb{F}^{128}$ eine Nachricht beliebiger Länge $m = (m_1, \ldots, m_n)$ mit $h(m) = h$ finden. Dabei können sogar die Werte $m_1, \ldots, m_{n-1}$ beliebig gewählt werden. Sei nun

$$k := E(m_{n-1}, \ldots E(m_2, E(m_1, IV)) \cdots)$$

und

$$m_n := E^{-1}(h, k),$$

d.h. wir dechiffrieren h mittels des Schlüssels k. Offensichtlich gilt dann $h(m) = h$. □

Die modifizierte Funktion $f(x, y) := E(x, y) \oplus x$ weißt diese Schwäche nicht auf. Die Konstruktion im Beweis von Satz 4.1 zum Finden einer Kollision lässt sich auf diese Funktion offensichtlich nicht übertragen.

4.1 Geburtstagsparadoxon

Ziel dieses Abschnittes ist, eine untere Schranke für das Sicherheitsniveau für Hashfunktionen anzugeben. Genauer wollen wir berechnen, welche Größe der Bildraum einer Hashfunktion haben muss, um insbesondere die Eigenschaft Collision resistance zu erfüllen. Offensichtlich lässt sich um so leichter eine Kollision finden, je kleiner der Bildraum ist.

Dazu betrachten wir zunächst das sogenannte Geburtstagsparadoxon, ein Beispiel aus der Wahrscheinlichkeitstheorie dafür, dass Wahrscheinlichkeiten häufig intuitiv falsch eingeschätzt werden.

Hier geht es um folgende Fragestellung: *Wie groß ist die Wahrscheinlichkeit dafür, dass von* 23 *zufällig ausgewählte Personen mindestens zwei am selben Tag Geburtstag haben (der Jahrgang spielt keine Rolle)?*

Häufig wird eine Wahrscheinlichkeit von $5 - 10$ Prozent angegeben, tatsächlich beträgt sie aber über 50 Prozent.

Wir berechnen zunächst die Wahrscheinlichkeit dafür, dass n Personen an unterschiedlichen Tagen Geburtstag haben.

- Die Wahrscheinlichkeit dafür, dass die erste Person an irgendeinem Tag Geburtstag hat, beträgt 365/365.
- Die Wahrscheinlichkeit dafür, dass die zweite Person an einem anderen Tag als die erste Geburtstag hat, beträgt 364/365.
- Die Wahrscheinlichkeit dafür, dass die dritte Person an einem anderen Tag als die ersten beiden Geburtstag hat, beträgt 363/365.
- usw.
- Die Wahrscheinlichkeit dafür, dass die $n-$te Person an einem anderen Tag als die ersten $n - 1$ Geburtstag hat, beträgt $(365 - n + 1)/365$.

Insgesamt ist also die Wahrscheinlichkeit dafür, dass n Personen an unterschiedlichen Tagen Geburtstag haben genau

$$1 \cdot \frac{364}{365} \cdot \frac{363}{365} \cdots \frac{365-n+1}{365} = \prod_{i=1}^{n} \frac{365-i+1}{365}$$

und damit die Wahrscheinlichkeit dafür, dass mindestens zwei Personen am selben Tag Geburtstag haben genau

$$1 - \prod_{i=1}^{n} \frac{365-i+1}{365} = 1 - \prod_{i=1}^{n-1} \frac{365-i}{365}.$$

Ausrechnen zeigt folgende Werte:

- Für $n = 10$ beträgt die Wahrscheinlichkeit $0,117$.
- Für $n = 23$ beträgt die Wahrscheinlichkeit $0,507$.
- Für $n = 36$ beträgt die Wahrscheinlichkeit $0,832$.

Warum versagt hier die Intuition. Ein Grund ist wahrscheinlich, dass die Fragestellung mit folgender Aufgabenstellung verwechselt wird: *Wie groß ist die Wahrscheinlichkeit dafür, dass von 23 Personen eine am selben Tag wie ich Geburtstag hat (wieder ohne Beachtung des Geburtsjahres)?*

Auch hier bestimmen wir zunächst die Wahrscheinlichkeit dafür, dass n Personen nicht an einem bestimmten Tag Geburtstag haben. Diese beträgt offensichtlich $(364/365)^n$. Damit beträgt also für die gesuchte Wahrscheinlichkeit $1 - (364/365)^n$, für $n = 23$ also $0,06$.

Wir kommen nun auf unsere eigentliche Fragestellung zurück. Sei dazu eine Hashfunktion $h : \mathbb{F}_2^m \longrightarrow \mathbb{F}_2^n$ mit $m >> n$ gegeben. Wir nehmen im Folgenden an, dass die Werte im Bildraum $\mathbb{F}_2^n$ ungefähr gleichverteilt sind. Es gilt also $|h^{-1}(z)| \approx m/n$ für alle $z \in \mathbb{F}_2^n$.

Wir wählen gemäß der Gleichverteilung k unterschiedliche Elemente $x_1, \ldots, x_k \in \mathbb{F}_2^m$. Da unter unserer Annahme die Urbilder ungefähr die gleiche Größe haben, können wir voraussetzen, dass die Bilder $(z_1, \ldots, z_k) = (h(x_1), \ldots, h(x_k))$ ebenfalls gemäß der Gleichverteilung zufällig sind. Wir wollen nun die Wahrscheinlichkeit dafür berechnen, dass mindestens zwei dieser Hashwerte gleich sind, also eine Kollision bilden.

Die obige Rechnung für den Spezialfall des Geburtstagsparadoxons lässt sich offensichtlich wie folgt verallgemeinern: Die Wahrscheinlichkeit dafür, dass eine Kollision gefunden wurde, beträgt

$$1 - \prod_{i=1}^{k-1} \frac{2^n - i}{2^n} = 1 - \prod_{i=1}^{k-1} 1 - \frac{i}{2^n}.$$

Für kleine Zahlen $x \in \mathbb{R}$ lässt sich die Funktion $x \mapsto 1 - x$ durch die Funktion $x \mapsto e^{-x}$ relativ gut approximieren. Wir erhalten also

$$1 - \prod_{i=1}^{k-1} 1 - \frac{i}{2^n} \approx 1 - \prod_{i=1}^{k-1} e^{-i/2^n} = 1 - e^{\sum_{i=1}^{k-1} -i/2^n} = 1 - e^{-(k(k-1))/(2 \cdot 2^n)} =: p.$$

Der Wert p ist nun die Wahrscheinlichkeit dafür, dass eine Kollision gefunden wurde. Wir interessieren uns dafür, wie viele Werte $x_1, \ldots, x_k$ zufällig gewählt werden müssen, um mit hoher Wahrscheinlichkeit eine Kollision zu finden. Wir wollen also k in Abhängigkeit von p beschreiben.

Es gilt nun:

$$\begin{aligned}
1 - e^{-(k(k-1))/(2\cdot 2^n)} = p &\iff \\
e^{-(k(k-1))/(2\cdot 2^n)} = 1 - p &\iff \\
\frac{-k(k-1)}{2\cdot 2^n} = \ln(1-p) &\iff \\
k^2 - k = 2^{n+1} \ln\frac{1}{1-p} &\iff \\
k^2 \approx 2^{n+1} \ln\frac{1}{1-p} &\iff \\
k \approx \sqrt{2^{n+1} \ln\frac{1}{1-p}} &\iff \\
k \approx 2^{(n+1)/2} \cdot \sqrt{\ln\frac{1}{1-p}} &\iff
\end{aligned}$$

Für $p = 1/2$ ergibt sich somit $k \approx 2^{(n+1)/2}\sqrt{\ln(2)} \approx 0.83 \cdot 2^{(n+1)/2}$.

Für ein geforderte Sicherheitsniveau von 100 Bit muss daher für eine Hashfunktion $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ mindestens die Bedingung $n \geq 200$ gelten.

Bemerkung 4.2. (*i*) Die Kollisionsangriffe der Arbeitsgruppe um die chinesische Kryptologin X. Wang (siehe [61]) auf die Hashfunktion SHA-1 (spezifiziert in [28]) haben eine dynamische Entwicklung bei der Analyse von Hashfunktionen ausgelöst. Zusätzlich bildet SHA-1 (wie auch die in [33] spezifizierte Funktion RIPEMD-160) Bitstrings beliebiger Länge auf Bitstrings der Länge 160 ab. Allein aus diesem Grund erfüllen diese Funktionen das geforderte Sicherheitsniveau von 100 Bit nicht.

(*ii*) Man beachte, dass schon eine einzige Kollision einer Hashfunktion zu einer Unsicherheit bei Signaturverfahren führen kann, vergleiche [23] und [31].

4.2 Empfohlene Hashfunktionen

Nach heutigem Kenntnisstand gelten die folgenden Hashfunktionen als kryptographisch stark und sind damit für alle verwendeten Verfahren einsetzbar:

1. SHA-224, SHA-256, SHA-384, SHA-512, siehe [28].

Tabelle 4.2: Empfohlene Hashfunktionen

Bemerkung 4.3. Nicht für alle der hier empfohlenen kryptographischen Protokolle, in denen Hashfunktionen eingesetzt werden, ist es erforderlich, dass diese kryptographisch stark sind. So reicht es zum Beispiel bei deterministischen Zufallszahlengeneratoren, die auf Hashfunktionen basieren, aus, dass die erste Eigenschaft erfüllt ist, dass es also praktisch unmöglich ist, Urbilder zu finden.

4.3 Übungsaufgaben

Übungsaufgabe 4. Überlegen Sie sich weitere Modifikationen von Blockchiffren, um eine sichere Kompressionsfunktion zu erhalten. Begründen Sie Ihre Wahl.

Übungsaufgabe 5. Fassen Sie die Ergebnisse aus [31] insbesondere in Bezug auf Bemerkung 4.2 zusammen.

5 Datenauthentisierung

Unter Datenauthentisierung verstehen wir kryptographische Verfahren, die garantieren, dass übersandte oder gespeicherte Daten nicht verändert wurden. Genauer benutzt ein Beweisender (üblicherweise der Sender der Daten) einen kryptographischen Schlüssel zur Berechnung der Prüfsumme der zu authentisierenden Daten. Ein Prüfer (üblicherweise der Empfänger der Daten) prüft dann, ob die empfangene Prüfsumme der zu authentisierenden Daten mit der übereinstimmt, die er bei Unverfälschtheit der Daten und Verwendung des richtigen Schlüssels erwarten würde.

Man unterscheidet symmetrische und asymmetrische Verfahren. Bei symmetrischen Verfahren benutzen Beweisender und Prüfer den selben kryptographischen Schlüssel, ein Dritter kann also in diesem Fall nicht überprüfen, wer die Prüfsumme berechnet hat. Bei asymmetrischen Verfahren wird der private Schlüssel für die Berechnung der Prüfsumme benutzt und mit dem zu diesem Schlüssel assoziierten öffentlichen Schlüssel überprüft.

5.1 Message Authentication Code (MAC)

Message Authentication Codes sind symmetrische Verfahren zur Datenauthentisierung, die sich üblicherweise auf Blockchiffren oder Hashfunktionen stützen. Beweisender und Prüfer müssen also vorab einen gemeinsamen symmetrischen Schlüssel vereinbart haben. Diese Verfahren werden üblicherweise dann eingesetzt, wenn große Datenmengen authentisiert werden müssen. Häufig muss sowohl die Vertraulichkeit als auch die Authentizität der Daten gewährleistet werden, siehe Abschnitt 10.1 für solche Verfahren. Siehe weiter Kapitel 7 für Verfahren, mit denen Schlüssel über unsichere Kanäle ausgetauscht werden können.

MAC-Verfahren erfüllen das Schutzziel Authentizität, nicht aber das Schutzziel Nichtabstreitbarkeit. Da beide Kommunikationspartner den selben Schlüssel nutzen, kann gegenüber einem Dritten nicht nachgewiesen werden, wer die Daten authentisiert hat, die Berechnung der Prüfsumme kann von allen Besitzern des symmetrischen Schlüssels durchgeführt werden.

Die einfachste Idee, ein MAC-Verfahren aus einer Hashfunktion zu konstruieren, ist, aus dem Schlüssel k und der Nachricht m den Wert $m_k := k||m$ zu bilden und $h = h(m_k)$ zu berechnen. Der Empfänger der Daten überprüft dann die Korrektheit von h, indem er die gleiche Berechnung durchführt und beide Werte vergleicht. Allerdings ist dieses Verfahren nicht sicher. Um dies einzusehen, betrachten wir die in Abschnitt 4 diskutierte Konstruktion von Hashfunktionen nach Merkle-Damgård, siehe Abbildung 5.1.

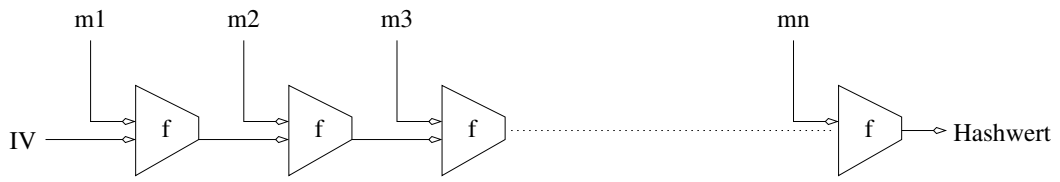


Tabelle 5.1: Hashfunktion nach Merkle-Damgård

Ein Angreifer kann damit also einfach weitere Blöcke an die Nachricht m anhängen und den Hashwert weiterrechnen. Dies ist ein typisches Beispiel dafür, dass ein auf den ersten Blick

offensichtlich sicheres Verfahren bei genaue Betrachtung extrem unsicher ist. Aus diesem Grund sollte man bei der Auswahl kryptographischer Verfahren auf etablierte Verfahren setzen.

Ein sicheres MAC-Verfahren, das auf einer Hashfunktion basiert, ist das folgende.

$$\mathrm{HMAC}_k(m) := h((k \oplus opad)||h((k \oplus ipad)||m)).$$

Die innere Komponente sichert, wie wir oben gesehen haben, nur den Anfang der Nachricht ab, die äußere Komponente auch das Ende der Nachricht.

Dabei sind

$$opad := \underbrace{0x5C \cdots 0x5C}_{b-\text{mal}} \quad \text{und} \quad ipad := \underbrace{0x36 \cdots 0x36}_{b-\text{mal}}$$

Konstanten.

Grundsätzlich gelten die folgenden Verfahren als sicher, wenn unter 1. eine aus Tabelle 2.3 aufgeführte Blockchiffre einsetzt wird (das Verfahren CMAC behandeln wir in den Übungen), bzw. unter 2. eine aus Tabelle 4.2 aufgeführte Hashfunktion eingesetzt wird und die Länge des Schlüssels für beide Verfahren mindestens 16 Byte beträgt:

1. CMAC, siehe [53],
2. HMAC, siehe [13].

Tabelle 5.2: Empfohlene MAC-Verfahren

5.2 Signaturverfahren

In Signaturalgorithmen werden die zu signierenden Daten zunächst gehasht und dann aus diesem Hashwert die Prüfsumme bzw. die Signatur mit dem geheimen Schlüssel des Beweisenden berechnet. Der Prüfer verifiziert dann die Signatur anhand der Daten mit dem zu dem geheimen Schlüssel assoziierten öffentlichen Schlüssel. Wie schon bei asymmetrischen Verschlüsselungsverfahren darf es dabei praktisch nicht möglich sein, die Signatur ohne Kenntnis des geheimen Schlüssels zu berechnen. Dies impliziert insbesondere, dass der geheime Schlüssel nicht aus dem öffentlichen Schlüssel konstruiert werden kann.

Man beachte, dass hierfür die zum Einsatz kommende Hashfunktion h kollisionsresistent sein muss, um die Sicherheit zu gewährleisten. Gibt es eine Kollision, d.h. zwei Werte x, y mit $h(x) = h(y)$, so könnte ein Angreifer die Signatur von x berechnen lassen und hätte so auch eine gültige Signatur für y.

Im Gegensatz zu MAC-Verfahren, die nur die Authentizität der Daten gewährleisten, erfüllen Signaturverfahren auch das Schutzziel Nichtabstreitbarkeit. Die Prüfsumme kann nur vom Inhaber des geheimen Schlüssels berechnet werden.

Für die Zuordnung des öffentlichen Schlüssels zum Beweisenden wird üblicherweise eine Public Key Infrastruktur benötigt, siehe Abschnitt 11.

Für die Spezifizierung von Signaturverfahren sind also folgende Algorithmen festzulegen:

1. Eine Hashfunktion, die die zu signierenden Daten auf eine Bitlänge fester Länge abbildet.
2. Ein Algorithmus zur Generierung von Schlüsselpaaren.

3. Ein Algorithmus zum Signieren der gehashten Daten und ein Algorithmus zum Verifizieren der Signatur.

Zusätzlich geben wir Empfehlungen für minimale Schlüssellängen an.

Für die Berechnung des Hashwertes sind grundsätzlich alle der in Tabelle 4.2 aufgelisteten Hashfunktionen geeignet. Wir müssen also im Folgenden jeweils nur noch die unter Punkt 2. und 3. aufgeführten Algorithmen und Schlüssellängen angeben. Im übrigen können alle empfohlenen Verfahren sowohl zur Signierung von Daten, als auch zum Ausstellen von Zertifikaten genutzt werden.

Tabelle 5.3 gibt einen Überblick über sichere Signaturverfahren.

1. RSA, siehe [36],
2. DSA, siehe [36] und [29],
3. DSA-Varianten auf elliptischen Kurven:
   a) ECDSA, siehe [9],
   b) ECKDSA, ECGDSA, siehe [38], und
   c) Nyberg-Rueppel-Signaturen, siehe [40].
4. Merkle-Signaturen, siehe [18].

Tabelle 5.3: Empfohlene Signaturverfahren

Vergleiche auch den sogenannten Algorithmenkatalog zum deutschen Signaturgesetz, siehe [14].

Bemerkung 5.1. Bei der Auswahl der empfohlenen asymmetrischen Signaturverfahren wurde darauf geachtet, dass lediglich sogenannte probabilistische Algorithmen[1] zum Einsatz kommen. Hier wird also bei jeder Berechnung einer Signatur ein **neuer** Zufallswert benötigt. Anforderungen an diese Zufallswerte werden in den entsprechenden Abschnitten angegeben.

5.2.1 RSA

Das RSA-Verfahren haben wir bereits in Unterabschnitt 3.2 kennen gelernt. Eine Nachricht m wurde mittels eines öffentlichen Schlüssels e zu $m^e \bmod n$ verschlüsselt. Die Nachricht kann dann nur vom Besitzer des geheimen Schlüssels d via $(m^e)^d = m \mod n$ entschlüsselt werden.

Dieses Verfahren kann ähnlich auch als Signaturverfahren genutzt werden. Dazu wird die Nachricht m zunächst gehasht, d.h. $h = h(m)$ für eine kryptographisch starke Hashfunktion h berechnet. Der Inhaber des geheimen Schlüssels d berechnet dann die Signatur s vom m via $s = h^d \bmod n$. Jeder Kommunikationspartner kann unter Zuhilfenahme des öffentlichen Schlüssels e die Korrektheit der Signatur s wie folgt berechnen: Der Empfänger der Daten und der Signatur (m, s) berechnet $v = s^e \bmod n$ und $h = h(m)$ und prüft, ob $v = h$ gilt. Ist dies der Fall, so ist die Signatur korrekt, d.h. s wurde tatsächlich vom Inhaber des geheimen Schlüssels erzeugt.

Signaturerzeugung und Signaturverifikation Für die Signaturerzeugung und -verifikation siehe auch [36]. Wie schon beim Verschlüsselungsverfahren muss der zu verschlüsselnde Text (hier

[1] Der RSA-Algorithmus selbst ist nicht probabilistisch, dafür aber die hier empfohlenen Paddingverfahren zu RSA.

der Hashwert der Nachricht) vor Anwendung des öffentlichen Schlüssels e auf die Bitlänge des Moduls n formatiert werden. Das Formatierungsverfahren ist dabei sorgfältig zu wählen, siehe zum Beispiel [21]. Die folgenden Verfahren werden empfohlen:

1. EMSA-PSS, siehe [58],
2. Digital Signature Scheme (DS) 2 und 3, siehe [42].

Tabelle 5.4: Empfohlene Formatierungsverfahren für den RSA-Signaturalgorithmus

Schlüssellänge Wie schon im Fall der RSA-Verschlüsselung sollte die Länge des Moduls n mindestens 2048 Bit betragen.

5.2.2 Digital Signature Algorithm (DSA)

Die Sicherheit dieses Verfahrens beruht auf der vermuteten Schwierigkeit des Diskreten Logarithmusproblems in bestimmten endlichen Gruppen.

Diskretes Logarithmus Problem:
Gegeben: Zwei Elemente $g, h \in G$.
Lösung: Der Logarithmus von h zur Basis g, d.h. $x \in \mathbb{N}$ so, dass $g^x = h$.

Bevor wir den Algorithmus formulieren können, benötigen wir aber noch einige mathematische Grundlagen.

Eine Kategorie von endlichen Gruppen, in denen dieses Problem als schwierig gilt, sind die multiplikativen Gruppen der Körper $\mathbb{Z}_p$, wobei p eine sehr große Primzahl ist (in Abschnitt 3.2 haben wir gesehen, dass in diesem Fall $\mathbb{Z}_p$ tatsächlich einen Körper bildet). Die Größe von p hängt vom geforderten Sicherheitsniveau ab, für ein Sicherheitsniveau von 100 Bit muss p mindestens 2048 groß sein. Wir kommen am Ende dieses Unterabschnitts darauf noch zurück.

Üblicherweise schreibt man, wenn p eine Primzahl ist, $\mathbb{F}_p$ anstelle von $\mathbb{Z}_p$, um klarzumachen, dass es sich um einen Körper ($\mathbb{F}$, da mathematische Körper im englischen field heißen).

Für jede Primzahl p ist die multiplikative Gruppe $(\mathbb{F}_p^*, \cdot)$ zyklisch, d.h. es gibt ein Element $a \in \mathbb{F}_p$ so, dass $\mathbb{F}_p^* = \{a^n; n \in \mathbb{N}\}$. Solch ein Element a heißt auch Erzeuger der Gruppe.

Beispiel 5.2. Wir betrachten die Gruppe $\mathbb{F}_5^* = \{1, 2, 3, 4\}$. Hier gilt

$$\mathbb{F}_5^* = \{2^1 = 2, 2^2 = 4, 2^3 = 3 \bmod 5, 2^4 = 1 \bmod 5\}.$$

Also ist 2 ein Erzeuger der Gruppe. Dass nicht jedes Element ein Erzeuger ist, sehen wir, wenn wir die gleiche Konstruktion mit dem Element 4 wiederholen. Hier gilt $\{4^n; n \in \mathbb{N}\} = \{4, 1\}$.

Ist nun $g \in \mathbb{F}_p^*$ ein Erzeuger der multiplikativen Gruppe, so ist das Logarithmusproblem zur Basis g eindeutig lösbar: Für jedes Element $h \in \mathbb{F}_p^*$ gibt es genau ein $x \in \mathbb{N}$ mit $g^x = h$. Die natürliche Zahl x ist dann die eindeutige Lösung.

Sei weiter $h \in \mathbb{F}_p^*$. Dann ist $\mathbb{F}_p^*(g) := \{g^n; n \in \mathbb{N}\}$ eine Untergruppe von $\mathbb{F}_p^*$ und per Definition ebenfalls zyklisch.

Wir sind nun in der Lage, den Signaturalgorithmus einzuführen.

Schlüsselgenerierung

1. Wähle zwei Primzahlen p und q so, dass

$$q \text{ teilt } p-1$$

   gilt.
2. Wähle x in $\mathbb{F}_p^*$ und berechne $g := x^{(p-1)/q} \bmod p$.
3. Falls $g = 1$, gehe zu 2.
4. Wähle eine Zahl $a \in \{1, \ldots, q-1\}$ und setze $A := g^a$.

Dann ist (p, q, g, A) der öffentliche Schlüssel und a der geheime Schlüssel.

Die Wahl von g in Schritt 2 sorgt dafür, dass die Untergruppe $\mathbb{F}_p^*(g)$ die Mächtigkeit q besitzt, siehe Satz 5.3. Die Größen der beiden Primzahlen p und q sind die Sicherheitsparamter des Verfahrens. Um den geheimen Schlüssel a zu ermitteln, kann ein Angreifer das diskrete Logarithmusproblem entweder in der Gruppe $\mathbb{Z}_p^*$ (die die Mächtigkeit $p-1$ besitzt) oder in $\mathbb{Z}_p^*(g)$ (die die Mächtigkeit q besitzt) versuchen zu lösen. Je größer die beiden Primzahlen, um so größer also die Sicherheit des Signaturverfahrens.

Satz 5.3. *Für alle $g = x^{(p-1)/q} \bmod p \in \mathbb{F}_p$ hat die Untergruppe $\{g^n; n \in \mathbb{N}\}$ genau q Elemente.*

Beweis. Es gilt $g^q = (x^{(p-1)/q})^q = x^{p-1} = 1 \bmod p$, wobei die letzte Gleichung aus Lemma 3.4 (i) folgt. Also besitzt die Untergruppe höchstens q Elemente.

Ist nun $z \leq q$ das minimale Element mit $g^z = 1 \bmod p$, so ist z ein Teiler von q. Angenommen, z ist kein Teiler von q Dann existieren $t, i \in \mathbb{N}$ mit $zt + i = q$ mit $i < z$. Es gilt nun

$$1 = g^q \bmod p = g^{zt+i} \bmod p = (g^z)^t g^i \bmod p = g^i \bmod p,$$

ein Widerspruch zur Minimalität von z. Also ist z ein Teiler von q. Da q eine Primzahl ist, folgt $z = q$ und damit die Behauptung. □

Signaturerzeugung Um ein Dokument m zu signieren, wird zunächst der Hashwert $h = h(m)$ gebildet. Aus diesem Hashwert berechnet man nun die Signatur wie folgt:

1. Wähle eine zufällige Zahl $1 < s < q$
2. Berechne $s_1 = (g^s \bmod p) \bmod q$
3. Berechne $s_2 = s^{-1}(h + s_1 \cdot a) \bmod q$

Der Wert (s_1, s_2) ist dann die Signatur von m. Die Zufallszahl s muss geheim gehalten werden. Bei Kenntnis von s kann ein Angreifer den geheimen Schlüssel berechnen, siehe Bemerkung 5.5. Aus dem gleichen Grund muss die Entropie, d.h. die Unvorhersagbarkeit, von s groß sein.

Signaturverifikation Zur Verifikation der Signatur benötigt der Prüfer den öffentlichen Schlüssel A. Zunächst wird der Hashwert $h = h(m)$ berechnet. Danach wird die Signatur wie folgt verifiziert:

1. Prüfe, ob $0 < s_1, s_2 < q$ gilt.
2. Berechne $w = s_2^{-1} \bmod q$
3. Berechne $u_1 = h \cdot w \bmod q$
4. Berechne $u_2 = s_1 \cdot w \bmod q$
5. Berechne $v = (g^{u_1} \cdot A^{u_2} \bmod p) \bmod q$

6. Ist $v = s_1$, so akzeptiere die Signatur.

Satz 5.4. *Ist die Signatur korrekt, so wird sie auch vom Prüfer akzeptiert.*

Beweis. Sei (s_1, s_2) eine korrekte Signatur vom m. Aus $s_2 = s^{-1}(h + s_1 \cdot a) \bmod q$ folgt durch Multiplikation auf beiden Seiten der Gleichung mit $sw \bmod q$

$$s_2 sw = (hw + s_1 aw) \bmod q.$$

Aus der Definition von w erhalten wir damit

$$s = (hw + s_1 aw) \bmod q$$

und aus der Wahl von u_1, u_2

$$s = u_1 + u_2 a \bmod q.$$

Es existiert also $n \in \mathbb{N}$ mit

$$s + nq = u_1 + u_2 a.$$

Da g die Ordnung q in $\mathbb{F}_p$ besitzt (d.h. $g^q = 1 \bmod p$), folgt

$$g^s \bmod p = g^{s+nq} \bmod p = g^{u_1+u_2 a} \bmod p = g^{u_1}(g^a)^{u_2} \bmod p = g^{u_1} A^{u_2} \bmod p.$$

Daraus folgt nun

$$s_1 = (g^s \bmod p) \bmod q = (g^{u_1} \cdot A^{u_2} \bmod p) \bmod q = v,$$

also $s_1 = v$. □

Für Signaturerzeugung und Signaturverifikation siehe auch [36] und [29].

Schlüssellänge Der derzeit einzige Angriff auf dieses Verfahren ist, den geheimen Signaturschlüssel a zu berechnen. Da der öffentliche Schlüssel $(p, q, g, A = g^a)$ bekannt ist, reduziert sich die Sicherheit des DSA-Verfahrens auf die Schwierigkeit der Lösbarkeit des diskreten Logarithmusproblems in den zwei Gruppe $\mathbb{F}_p^*$ und $\mathbb{F}_p^*(g)$. Um die Schlüssellänge bestimmen zu können, benötigen wir also die nach heutigem Kenntnisstand schnellsten Algorithmen zum Lösen dieses Problems. Der derzeit schnellste Algorithmus zur Berechnung des diskreten Logarithmus in der multiplikativen Gruppe $\mathbb{F}_p^*$ hat eine Laufzeit von

$$\mathcal{O}(e^{\sqrt{\log n \log\log n}})$$

und in $\mathbb{F}_p^*(g)$ eine Laufzeit von

$$\mathcal{O}(\sqrt{n}),$$

mit $n = |\mathbb{F}_p^*(g)|$. Die Primzahl p sollte also eine Mindestgröße von 2024 Bit und die Ordnung der zyklischen Untergruppe $\mathbb{F}_p^*(g)$ (deren Größe nach Satz 5.3 gerade q ist) eine Mindestgröße von 200 Bit.

Bemerkung 5.5. Das DSA-Verfahren ist ein sogenannter probabilistischer Algorithmus, d.h. das für die Berechnung der Signatur eine Zufallszahl s benötigt wird. Hier ist $s \in \{1, \ldots, q-1\}$ und sollte bezüglich der Gleichverteilung auf $\{1, \ldots, q-1\}$ gewählt werden, da es ansonsten Angriffe gibt, vergleiche [29, Change Notice 1]. Siehe auch Abschnitt 12.2 für einen empfohlenen Algorithmus zur Berechnung des Zufallswertes s.

(i) Ist beispielsweise der Zufallswert s, die Signatur (s_1, s_2) und die zu signierende Nachricht m bekannt, so lässt sich der geheime Schlüssel a wie folgt berechnen:

$$a = (\underbrace{(s_2 \cdot s)}_{=h(m)+s_1 \cdot a \bmod q} - h(m)) \cdot s_1^{-1} \bmod q.$$

Ist weiter der Algorithmus zur Berechnung der Zufallszahl s schwach, d.h. ist die Entropie von s so klein, dass ein Angreifer alle Möglichkeiten von s durchprobieren kann, so kann er dank der obigen Formel ebenfalls den geheimen Schlüssel berechnen.

(*ii*) Weiter muss bei jeder Signaturberechnung ein anderer Zufallswert s genutzt werden, da sonst der geheime Schlüssel a wie folgt berechnet werden kann: Wir nehmen also an, dass zwei verschiedene Nachrichten m, m' mittels des selben geheimen Schlüssels a zu (s_1, s_2) und (s_1', s_2') signiert wurden. Damit gilt dann

$$s_1 = (s^{-1} \bmod p) \bmod q = s_1'$$

und

$$\begin{aligned} s_2 &= s^{-1}(m + s_1 a) \bmod q \\ s_2' &= s^{-1}(m' + s_1' a) \bmod q = s^{-1}(m' + s_1 a) \bmod q, \end{aligned}$$

also

$$\begin{aligned} s_2 - s_2' &= s^{-1}(m + s_1 a) - s^{-1}(m' + s_1 a) \bmod q \\ &= s^{-1}(m - m') \bmod q, \end{aligned}$$

d.h.

$$s = (m - m')(s_2 - s_2')^{-1}.$$

Mit den Ergebnissen aus (*i*) lässt sich dann aus s der geheime Schlüssel a ermitteln.

5.3 Übungen

Übungsaufgabe 6. Beschreiben Sie das MAC-Verfahren CMAC.

6 Instanzauthentisierung

Instanzauthentisierungen sind Verfahren, in denen ein Beweisender einem Prüfer den Besitz eines Geheimnisses nachweist. Bei symmetrischen Verfahren ist dies ein symmetrischer Schlüssel, der vorab ausgetauscht werden muss. In asymmetrischen Verfahren zeigt der Beweisende, dass er im Besitz eines geheimen Schlüssels ist. Hier wird eine Public Key Infrastruktur benötigt, damit der Prüfer den zugehörigen öffentlichen Schlüssel dem Beweisenden zuordnen kann. Passwortbasierte Verfahren dienen in erster Linie der Freischaltung von Chipkarten oder anderen kryptographischen Komponenten. Hier beweist der Inhaber der Komponente, dass er im Besitz einer PIN ist.

Die Authentisierung erfolgt meist gegenseitig und geht mit einer Schlüsseleinigung einher, um die Vertraulichkeit und Integrität einer anschließenden Kommunikation zu gewährleisten, siehe Kapitel 7 für empfohlene Schlüsselaustausch- und Schlüsseleinigungsverfahren und Abschnitt 10.2 für empfohlene Protokolle, die beide Verfahren kombinieren.

Deshalb werden in diesem Kapitel für die ersten beiden Verfahren (Abschnitte 6.1 und 6.2) nur allgemeine Ideen zur Instanzauthentisierung angegeben und lediglich die entsprechenden kryptographischen Primitive empfohlen. Für die benötigten kryptographischen Protokolle sei auf Abschnitt 10.2 verwiesen. Insbesondere werden auch nur dort Empfehlungen für Schlüssellängen usw. angegeben.

6.1 Symmetrische Verfahren

Für den Nachweis des Beweisenden (B) gegenüber einem Prüfer (P), dass B im Besitz des geheimen symmetrischen Schlüssels ist, sendet P einen Zufallswert r an B. B berechnet dann mittels des gemeinsamen geheimen Schlüssels K eine Prüfsumme von r und schickt diese zurück an P. P prüft dann diese Prüfsumme. Solche Verfahren heißen auch *Challenge-Response-Verfahren*, siehe Tabelle 6.1 für eine schematische Darstellung.

Beweisender (B)		**Prüfer (P)**
		Wähle Zufallswert r
	$\xleftarrow{r}$ (Challenge)	
Berechne Prüfsumme c		
	$\xrightarrow{c}$ (Response)	
		Verifiziere Prüfsumme

Tabelle 6.1: Schematische Darstellung eines Challenge-Response-Verfahren zur Instanzauthentisierung

Die Berechnung und Verifikation der Prüfsumme hängt vom gewählten Verfahren ab. Grundsätzlich können alle in Kapitel 2 empfohlenen Verschlüsselungsverfahren und alle in Abschnitt 5.1 empfohlenen MAC-Verfahren eingesetzt werden. Für empfohlene Bitlängen und Nebenbedingungen an die benutzten Zufallswerte siehe Abschnitt 10.2.

6.2 Asymmetrische Verfahren

Wie schon im letzten Abschnitt werden auch für asymmetrische Verfahren Challenge-Response-Protokolle zur Instanzauthentisierung eingesetzt. Hier berechnet der Beweisende eine Prüfsumme zu einem vom Prüfer gesendeten Zufallswert r mit seinem geheimen Schlüssel. Der Prüfer verifiziert dann die Prüfsumme mit Hilfe des zugehörigen öffentlichen Schlüssels. Grundsätzlich können hierfür alle in Abschnitt 5.2 empfohlenen Verfahren eingesetzt werden. Für empfohlene Bitlängen und Nebenbedingungen an die benutzten Zufallswerte siehe ebenfalls Abschnitt 10.2.

Bemerkung 6.1. Auch wenn die in Abschnitt 5.2 empfohlenen Signaturverfahren zur Datenauthentisierung auch zur Instanzauthentisierung genutzt werden können, muss bei konkreten Implementierungen darauf geachtet werden, dass die eingesetzten Schlüssel verschieden sind. D.h., dass ein Schlüssel zur Erzeugung von Signaturen nicht zur Instanzauthentisierung eingesetzt werden darf. Dies muss in den entsprechenden Zertifikaten für die öffentlichen Schlüssel kenntlich gemacht werden, siehe auch Abschnitt 11. Andernfalls lässt sich folgender Angriff durchführen:

- Der Angreifer erzeugt eine Nachricht m und bildet $h = h(m)$.
- Im zweiten Schritt sendet der Angreifer im Rahmen eines Challenge-Response-Verfahrens in der Rolle des Prüfers anstelle des Zufallswertes r den Hashwert h.
- Der Beweisende kann nicht feststellen, ob es sich bei h um einen Zufallswert oder den Hashwert einer Nachricht handelt und signiert h im Glauben, ein Challenge-Response-Verfahren durchzuführen.
- Im Ergebnis erhält der Angreifer eine vom Beweisenden korrekt signierte Nachricht.

6.3 Passwortbasierte Verfahren

Passwörter zum Freischalten der auf kryptographischen Komponenten, wie zum Beispiel Signaturkarten, zur Verfügung gestellten kryptographischen Schlüssel sind meist kurz, damit sich der Inhaber der Komponente das Passwort auch merken kann. Um trotzdem ein ausreichendes Sicherheitsniveau zu erreichen, wird die Anzahl der Zugriffsversuche üblicherweise begrenzt.

Folgende Nebenbedingungen werden empfohlen:

1. Die Entropie des Passwortes muss mindestens $\log_2(10^6)$ Bit betragen (z. B. sechs Ziffern),
2. Die Anzahl der Zugriffsversuche muss auf drei beschränkt sein.

Tabelle 6.2: Empfohlene Passwortlängen und empfohlene Anzahl der Zugriffsversuche für den Zugriffsschutz kryptographischer Komponenten

Für kontaktbehaftete Chipkarten zum Beispiel ist das Verfahren sehr einfach. Das Passwort wird auf dem Pinpad des Kartenlesers eingegeben und ohne zusätzliche kryptographische Absicherung zur Chipkarte übertragen. Um trotzdem einen ausreichenden Schutz zu gewährleisten (Schutz vor Erraten der PIN über elektromagnetische Strahlung), muss ein Kartenleser eingesetzt werden, der entsprechend zertifiziert ist.

Bei kontaktlosen Chipkarten kann die Kommunikation zwischen Kartenleser und Chipkarte auch noch aus einiger Entfernung mitgelesen werden. Hier kann das Passwort zur Freischaltung der Chipkarte also nicht einfach vom Kartenleser zur Chipkarte gesendet werden.

Die Hauptidee des folgenden Protokolls ist, zwischen Chip und Kartenlesegerät ein verschlüsseltes Diffie-Hellman-Schlüsseleinigungsverfahren ablaufen zu lassen, siehe Abbildung 6.3.

		PIN-Eingabe
Wähle Zufallswert r	$\mathsf{ENC}_{\mathsf{PIN}}(r)$ ⟶	Entschlüssle r
Beide berechnen aus r einen Diffie-Hellman Basispunkt g		
	⟵ g^y	Wähle Zufallswert y
Wähle Zufallswert x	g^x ⟶	
Berechne $K = (g^y)^x = g^{xy}$		Berechne $K = (g^x)^y = g^{xy}$
Berechne $T_1 = \mathrm{MAC}(K, g^y)$	T_1 ⟶	Verifiziere T_1
Verifiziere T_2	⟵ T_2	Berechne $T_2 = \mathrm{MAC}(K, g^x)$

Tabelle 6.3: Das Protokoll PACE

Nach erfolgreicher Beendigung des Protokolls wissen beide Partner, dass sie die selbe PIN genutzt haben. Die PIN wird dabei nicht im Klartext übertragen.

Folgendes passwortbasiertes Verfahren wird für den Zugriffsschutz auf kontaktlose Chipkarten empfohlen:

PACE: Password Authenticated Connection Establishment, siehe [43].

Tabelle 6.4: Empfohlenes passwortbasiertes Verfahren für den Zugriffsschutz auf kontaktlose Chipkarten

Das in Tabelle 6.4 empfohlene Verfahren beweist der kontaktlosen Chipkarte nicht nur, dass der Benutzer im Besitz des korrekten Passwortes ist, sondern führt gleichzeitig ein Schlüsseleinigungsverfahren durch, so dass im Anschluss eine vertrauliche und authentisierte Kommunikation durchgeführt werden kann.

Bemerkung 6.2. (*i*) Wie schon bei kontaktbehafteten Komponenten muss auch hier die Anzahl der Versuche beschränkt sein. Empfohlen wird, nach drei erfolglosen Versuchen die Chipkarte zu sperren.

(*ii*) Um Denial-of-Service Attacken zu verhindern, muss ein Mechanismus zum Aufheben der Sperrung vorgesehen sein. Die Entropie des Schlüssels zur Entsperrung sollte mindestens 100 Bit betragen.

7 Schlüsseleinigungsverfahren

Schlüsseleinigungsverfahren dienen dem Austausch eines Verschlüsselungsschlüssels über einen unsicheren Kanal. Diese Verfahren müssen unbedingt mit Instanzauthentisierungsverfahren kombiniert werden. Ansonsten besteht keine Möglichkeit zu entscheiden, mit welcher Partei die Schlüsseleinigung durchgeführt wird (Datenauthentisierung allein genügt hier nicht, da ein Angreifer eine in der Vergangenheit durchgeführte Kommunikation mitgeschnitten haben könnte um die aufgezeichneten Daten für einen Angriff zu nutzen). Aus diesem Grund geben wir, wie schon in Kapitel 6, in diesem Kapitel nur ganz allgemeine Ideen für Schlüsseleinigungsverfahren an und verweisen für konkrete Verfahren, d.h. für Schlüsseleinigungsverfahren, die auch eine Instanzauthentisierung beinhalten, auf Abschnitt 10.2.

Nach erfolgreicher Schlüsseleinigung befinden sich beide Parteien im Besitz eines gemeinsamen Geheimnisses. Für empfohlene Verfahren zur Generierung symmetrischer Schlüssel aus diesem Geheimnis siehe Abschnitt 12.1.

Es wird empfohlen, nur Schlüsseleinigungsverfahren zu verwenden, in denen beide Kommunikationspartner Anteile für den neuen Schlüssel bereitstellen.

7.1 Symmetrische Verfahren

Grundsätzlich können alle der in Kapitel 2 empfohlenen symmetrischen Verschlüsselungsverfahren zum Aushandeln neuer Sitzungsschlüssel verwendet werden.

7.2 Asymmetrische Verfahren

Grundsätzlich können alle der in Kapitel 3 empfohlenen asymmetrischen Verschlüsselungsverfahren zum Aushandeln neuer Sitzungsschlüssel verwendet werden.

Zusätzlich zu diesen werden die folgenden beiden Verfahren empfohlen:

1. Diffie-Hellman, siehe [50],
2. EC Diffie-Hellman, siehe [50].

Tabelle 7.1: Empfohlene asymmetrische Schlüsseleinigungsverfahren

Wir behandeln in der Vorlesung lediglich das erste Verfahren: Das Diffie-Hellman-Schlüsseleinigungsverfahren wurde von Martin Hellman gemeinsam mit Whitfield Diffie und Ralph Merkle an der Stanford-Universität (Kalifornien) entwickelt und 1976 veröffentlicht. Wie erst 1997 bekannt wurde, hatte James Ellis, Clifford Cocks und Malcolm J. Williamson im Auftrag des britischen Government Communications Headquarters (GCHQ) schon in den frühen 1970er Jahren ein ähnliches Verfahren entwickelt.

Die Sicherheit des Diffie-Hellman-Schlüsseleinigungsverfahrens beruht, wie schon der Algorithmus DSA, auf der vermuteten Schwierigkeit des Diskreten Logarithmusproblems in den multiplikativen Gruppen der Körper $\mathbb{F}_p$, p eine Primzahl.

Für die Spezifizierung sind folgende Algorithmen festzulegen:

1. Ein Algorithmus zum Festlegen der Systemparameter.
2. Ein Algorithmus zur Schlüsseleinigung.

Systemparameter

1. Wähle eine Primzahl p.
2. Wähle einen Erzeuger g der multiplikativen Gruppe $\mathbb{F}_p^*$.

Das Paar (p, g) muss vorab **authentisch** zwischen den Kommunikationspartnern ausgetauscht werden.

Schlüsselvereinbarung

1. A wählt gleichverteilt einen Zufallswert $x \in \{1, \ldots, p-1\}$ und sendet $g^x \bmod p$ an B.
2. B wählt gleichverteilt einen Zufallswert $y \in \{1, \ldots, p-1\}$ und sendet $g^y \bmod p$ an A.
3. A berechnet $(g^y \bmod p)^x = g^{xy} \bmod p$.
4. B berechnet $(g^x \bmod p)^y = g^{xy} \bmod p$.

Das ausgehandelte Geheimnis ist dann $K = g^{xy} \bmod p$, siehe auch Abbildung 7.2.

A		B
Wähle Zufallswert x und berechne $g^x \bmod p$	$g^x \bmod p$ ⟶	
	⟵ $g^y \bmod y$	Wähle Zufallswert y und berechne $g^y \bmod p$
Berechne $K = (g^y \bmod p)^x \bmod p = g^{xy} \bmod p$		Berechne $K = (g^x \bmod p)^y = g^{xy} \bmod p$

Tabelle 7.2: Das Diffie-Hellman-Schlüsseleinigungsverfahren

Schlüssellänge Die Länge von p sollte mindestens 2048 Bit betragen, vergleiche die Ausführungen zum Signaturverfahren DSA aus Abschnitt 5.2.2.

Die Wahl eines Erzeugers der Gruppe sorgt dafür, dass die Schlüsseleinigung nicht in einer zu kleinen Untergruppe stattfindet und das Verfahren damit unsicher wird.

Beispiel 7.1. Sei q ein Primfaktor von $p-1$. In Satz 5.3 haben wir gezeigt, dass dann für $g = x^{(p-1)/q}$, mit $x \in \mathbb{F}_p$ die zyklische Untergruppe $\mathbb{F}_p(g)$ genau q Elemente hat. Damit gilt dann $\mathbb{F}_p(g) = \{g^n; n \leq q\}$. Ist q klein, ist in dieser Untergruppe der diskrete Logarithmus natürlich einfach zu berechnen. Ein Angreifer kennt g und g^x und findet (mittels Durchprobieren aller Möglichkeiten) $x' \leq q$ mit $g^x = g^{x'}$. Da er auch g^y kennt, kann er $(g^y)^{x'} = (g^{x'})^y = (g^x)^y = g^{xy}$, also das gemeinsame Geheimnis, berechnen.

8 Secret Sharing

Häufig müssen kryptographische Schlüssel über einen langen Zeitraum gespeichert werden. Dies erfordert insbesondere, dass Kopien dieser Schlüssel angelegt werden müssen, um einen Verlust der Schlüssel zu verhindern. Je mehr Kopien allerdings generiert werden, um so größer ist die Wahrscheinlichkeit dafür, dass das zu schützende Geheimnis kompromittiert wird.

Wir geben deshalb in diesem Kapitel ein Verfahren an, welches erlaubt, ein Geheimnis, wie zum Beispiel einen kryptographischen Schlüssel K, so in n Teilstücke $K_1, \ldots, K_n$ aufzuteilen, dass beliebige $t \leq n$ dieser Teilstücke genügen, um das Geheimnis zu rekonstruieren, aus $t-1$ Teilstücken aber nicht wieder auf K geschlossen werden kann.

Eine weitere Anwendung dieses Verfahrens ist ein Vieraugenprinzip oder allgemeiner ein t-aus-n-Augenprinzip zu gewährleisten, um zum Beispiel das Passwort für eine kryptographische Komponente so auf n verschiedene Anwender zu verteilen, dass mindestens t Anwender benötigt werden, um das Passwort zu rekonstruieren.

Das hier vorgestellte Secret-Sharing-Verfahren wurde von A. Shamir entwickelt, siehe [60]. Wir nehmen im Folgenden an, dass das zu verteilende Geheimnis als natürliche Zahl $K \in \mathbb{N}$ interpretiert werden kann. (Soll zum Beispiel ein 128-Bit Schlüssel $K = (K_0, \ldots, K_{127})$ verteilt werden, so interpretieren wir K als $K = \sum_{i=0}^{127} K_i \cdot 2^i$).)

Zur Berechnung der verteilten Geheimnisse auf n Benutzer, so dass t Benutzer das Geheimnis K wieder rekonstruieren können, geht man wie folgt vor:

1. Wähle eine Primzahl $p \geq \max(K, n)$ und setze $a_0 := K$.
2. Wähle unabhängig voneinander $t-1$ zufällige Werte $a_1, \ldots, a_{t-1} \in \{0, 1, \ldots, p-1\}$. Die Werte $a_0, a_1, \ldots, a_{t-1}$ definieren dann ein zufälliges Polynom
$$f(x) = \sum_{j=0}^{t-1} a_j x^j$$
über $\mathbb{F}_p$, für das $f(0) = K$ gilt.
3. Berechne die Werte $K_i := f(i)$ für alle $i \in \{1, \ldots, n\}$.

Tabelle 8.1: Berechnung der Teilgeheimnisse im Shamir Secret-Sharing-Verfahren

Die Teilgeheimnisse K_i werden dann, zusammen mit i dem $i-$ten Benutzer übergeben.

Bemerkung 8.1. Die Koeffizienten $a_0, \ldots, a_{t-1}$ eines unbekannten Polynoms f vom Grad $t-1$ können mit Hilfe der sogenannten *Lagrange-Interpolations-Formel* aus t Punkten $(x_i, f(x_i))$ wie folgt gefunden werden:

$$f(x) = \sum_{i=1}^{t} f(x_i) \prod_{1 \leq j \leq t, i \neq j} \frac{x - x_j}{x_i - x_j}.$$

Insbesondere lässt sich damit $K = f(0)$ aus t gegebenen Punkten berechnen. Dies ist genau die Grundlage für das obige Verfahren.

Um nun aus t Teilgeheimnissen $K_{j_1}, \ldots, K_{j_t}$ das Geheimnis K zu rekonstruieren, geht man wie folgt vor:

1. Berechne für alle $i \in \{j_1, \ldots, j_t\}$ den Wert $c_i = \prod_{1 \leq l \leq t, K_j \neq K_i} \frac{K_{j_l}}{K_{j_l} - K_{j_i}}$.
2. Berechne $K = \sum_{i=1}^{t} c_i K_{j_i}$.

Tabelle 8.2: Zusammensetzen der Teilgeheimnisse im Shamir Secret-Sharing-Verfahren

Der Wert K ist dann das zusammengesetzte Geheimnis.

Bemerkung 8.2. Die Wahl der Primzahl p unter der Bedingung $p \geq \max(K, n)$ garantiert, dass dieses Verfahren mindestens ein Sicherheitsniveau der Bitlänge des zu schützendes Geheimnisses hat. Zusätzliche Schlüssellängen müssen also nicht angegeben werden.

9 Zufallszahlengeneratoren

Zufallszahlen werden für eine Reihe von Anwendungen in kryptographischen Verfahren benötigt, wie zum Beispiel für kryptographische Schlüssel, Systemparameter für Signatur-, Authentisierungs- und asymmetrischen Verschlüsselungsverfahren, Instanzauthentisierung, probabilistische Signatur- bzw. Authentisierungsverfahren und Paddingverfahren.

In Anhang 12 geben wir für eine Vielzahl von Anwendungen Algorithmen an, wie aus Zufallszahlen die gewünschten Werte berechnet werden können.

Dabei spielt die Unvorhersagbarkeit, auch Entropie genannt, die entscheidende Rolle. Die Entropie gibt, einfach gesprochen, an, wie viele Versuche ein Angreifer mindestens benötigt, um alle Möglichkeiten durchzuprobieren. Wir messen im Folgenden die Entropie in Bit, d.h. zum Beispiel, besitzt ein symmetrischer Schlüssel eine Entropie von n Bit, so benötigt ein Angreifer mindestens 2^n Schritte, um den gesamten Schlüsselraum zu durchsuchen.

Es wird dringend empfohlen, einen physikalischen Zufallszahlengenerator zur Erzeugung von Zufallswerten zu benutzen. Zumindest sollte der Startwert (bzw. *Seed*, siehe Abschnitt 9.2), für einen eingesetzten deterministischen Zufallszahlengeneratoren mit Hilfe eines physikalischen Zufallszahlengenerator erzeugt und der innere Zustand des deterministischen regelmäßig mit Zufallsfolgen des physikalischen Zufallszahlengenerators erneuert werden.

9.1 Physikalische Zufallszahlengeneratoren

Physikalische Zufallszahlengeneratoren beziehen Zufallszahlen aus physikalischen Rauschquellen, die beispielsweise auf elektromagnetischen, elektromechanischen oder quantenmechanischen Effekten beruhen. Häufig ist eine deterministische Nachbearbeitung der digitalen Rauschsignale nötig, um eventuell auftretende Schiefen in der Ausgabe des physikalischen Zufallszahlengenerators nicht in die für kryptographische Verfahren verwendeten Zufallswerte einfließen zu lassen.

Es wird empfohlen, einen P2-Generator mit Stärke der Mechanismen bzw. Funktionen HOCH im Sinne der AIS 31, siehe [2], einzusetzen. Dies bedeutet unter anderem:

1. Die Eigenschaften der digitalisierten Rauschsignale müssen sich hinreichend gut durch ein stochastisches Modell beschreiben lassen.
2. Der durchschnittliche Entropiezuwachs pro Zufallsbit liegt oberhalb einer gegebenen Mindestschranke.
3. Die digitalisierten Rauschsignale müssen im laufenden Betrieb in regelmäßigen Abständen statistischen Tests unterzogen werden, die geeignet sind, nicht akzeptable statistische Defekte oder Verschlechterungen der statistischen Eigenschaften in angemessener Zeit zu erkennen.
4. Auf eine fehlerhafte Rauschsignalfolge muss zum Beispiel durch Stilllegen der Rauschquelle reagiert werden.

Die Entwicklung und sicherheitskritische Bewertung von physikalischen Zufallszahlengeneratoren setzt eine umfassende Erfahrung auf diesem Gebiet voraus.

9.2 Deterministische Zufallszahlengeneratoren

Deterministische bzw. Pseudozufallszahlengeneratoren berechnen aus einem Zufallswert fester Länge, dem sogenannten *Seed*, eine pseudozufällige Bitfolge praktisch beliebiger Länge. Dazu wird der sogenannte innere Zustand des Pseudozufallszahlengenerators zunächst mit dem Seed initialisiert. In jedem Schritt wird dann der innere Zustand erneuert und hieraus eine Zufallszahl (meist eine Bitfolge fester Länge) abgeleitet. Der innere Zustand muss natürlich gegen Auslesen und Manipulation entsprechend geschützt werden.

In [1] sind verschiedene Sicherheitsklassen von Pseudozufallszahlen definiert. Insbesondere sollten aber zumindest die folgenden beiden Kriterien durch solche Generatoren erfüllt sein:

1. Es ist einem Angreifer praktisch unmöglich, zu einer bekannten Zufallszahlenfolge Vorgänger oder Nachfolger dieser Teilfolge oder einen inneren Zustand zu berechnen.
2. Es ist einem Angreifer praktisch unmöglich, aus Kenntnis eines inneren Zustandes Vorgänger der Teilfolge oder Vorgängerzustände zu berechnen.

Zusätzlich muss die Entropie des Seed mindestens 100 Bit betragen.

Ein einfacher Pseudozufallszahlengenerator, der auf einer kryptographisch starken Hashfunktion basiert und die beiden obigen Bedingungen erfüllt, ist in Abbildung 9.1 dargestellt.

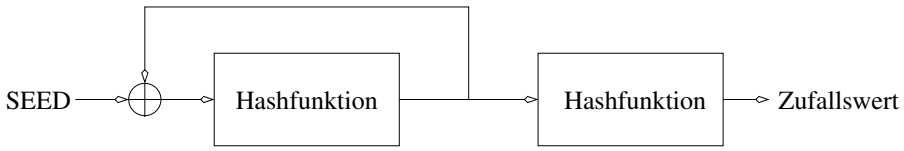


Abbildung 9.1: Ein einfacher Pseudozufallszahlengenerator

Weitere Beispiele für Pseudozufallszahlengeneratoren finden sich in [1].

9.3 Seedgenerierung

Wie bereits im letzten Abschnitt beschrieben, wird für die Initialisierung eines Pseudozufallszahlengenerators ein Seed mit ausreichend hoher Entropie (hier mindestens 100 Bit) benötigt.

Die Erzeugung dieses Seeds sollte mit physikalischen Zufallszahlengeneratoren erfolgen. In vielen Anwendungen steht dieser aber nicht zur Verfügung. Typische Szenarien sind der Einsatz kryptographischer Verfahren für E-Government- und E-Business-Anwendungen. Hier werden für viele Verfahren, wie zum Beispiel Verschlüsselung, Schlüsseleinigung oder Challenge-Response-Protokolle, Zufallszahlen mit hoher Entropie benötigt. Allerdings werden diese Anwendungen häufig auf Computern ohne zusätzliche kryptographische Hardware genutzt. Wir geben aus diesem Grund für die zwei wichtigsten Betriebssysteme geeignete Verfahren zur Seedgenerierung an.

9.3.1 GNU/Linux

Folgendes Verfahren wird zur Seedgenerierung unter dem Betriebsystem GNU/Linux empfohlen.

Bemerkung 9.1. Die Funktion /dev/random wurde bisher nur in der Kernelversion 2.6.21.5 vom BSI untersucht. Solange keine gravierenden Änderungen bei folgenden Kernelversionen in dieser Funktion vorgenommen werden, sollte die Funktion /dev/random auch für aktuellere Kernel ein Sicherheitsniveau von 100 Bit erreichen.

/dev/random (in der Kernelversion 2.6.21.5)

Tabelle 9.1: Empfohlenes Verfahren zur Seedgenerierung unter GNU/Linux

9.3.2 Windows

Im Gegensatz zum System GNU/Linux gibt es derzeit keine vom BSI untersuchte Funktion im System Windows, das ein ausreichendes Sicherheitsniveau von 100 Bit gewährleistet. Zur Erzeugung sicherer Seeds sollten daher verschiedene Systemaufrufe in geeigneter Weise kombiniert werden. Dazu eignen sich die beiden Funktionen

1. ReadTimeStampCounter():
   gibt die seit Systemstart durchlaufenen Prozessorzyklen an, bei einem Prozessor mit einer Taktfrequenz von mindestens 1 GHz gibt es pro Sekunde also mindestens 2^{30} verschiedene Werte.
2. KeQuerySystemTime():
   gibt die aktuelle Systemzeit an und hat eine Auflösung von 100 ns, nimmt also pro Sekunde mindestens 2^{23} verschiedene Werten an.

Kombiniert man diese beiden Funktionen in geeigneter Weise so, dass die Zeitpunkte, zu denen die Funktionen aufgerufen werden, paarweise voneinander weitestgehend unabhängig sind, so lässt sich ein Seed mit ausreichender Entropie erzeugen. Beispielhaft sei folgendes Verfahren angegeben. Dabei muss gewährleistet sein, dass alle Prozesse (inklusive das Starten des Rechners) von einem Benutzer ausgeführt werden und nicht automatisiert sind.

1. Starten des Rechners
2. Starten des Programms
   a) A:=ReadTimeStampCounter(),
   b) B:=KeQuerySystemTime(),
3. Verifizieren eines Logins
   a) C:=ReadTimeStampCounter(),
4. Erzeugen des Seeds
   a) D:=ReadTimeStampCounter(),
   b) SEED:=A||B||C||D,

Geht man davon aus, dass ein Angreifer die obigen Zeitpunkte nur auf höchstens eine Sekunde genau schätzen kann, so hat der Wert SEED eine Entropie von mindestens 113 Bit.

10 Anwendung kryptographischer Verfahren

Die in den vorangegangenen Kapiteln erläuterten Verfahren müssen häufig miteinander kombiniert werden, um den erforderten Schutz sensitiver Daten zu gewährleisten. Insbesondere sollten zu übertragende sensitive Daten nicht nur verschlüsselt, sondern zusätzlich authentisiert werden, damit eine etwaige Veränderung vom Empfänger erkannt wird.

Weiter muss eine Schlüsseleinigung immer mit einer Instanzauthentisierung einher gehen, damit sich beide Parteien sicher sind, mit wem sie kommunizieren. Andernfalls kann durch eine sogenannte Man-in-the-Middle-Attacke die Kommunikation kompromittiert werden. Für beide Anwendungen geben wir in diesem Kapitel geeignete Verfahren an.

10.1 Verschlüsselungsverfahren mit Datenauthentisierung (Secure Messaging)

Grundsätzlich können bei der Kombination von Verschlüsselung und Datenauthentisierung alle in Kapitel 2 bzw. Abschnitt 5.1 empfohlenen Verfahren eingesetzt werden.

Allerdings müssen die folgenden beiden Nebenbedingungen eingehalten werden:

1. Authentisiert werden ausschließlich die verschlüsselten Daten.
2. Verschlüsselungs- und Authentisierungsschlüssel müssen verschieden und nicht voneinander ableitbar sein.

Bemerkung 10.1. Es besteht die Möglichkeit, Verschlüsselungs- und Authentisierungsschlüssel aus einem gemeinsamen Schlüssel abzuleiten. Empfohlene Verfahren sind in Abschnitt 12.1 zusammengefasst.

Bemerkung 10.2. Bei der in Abschnitt 2.2 empfohlenen Betriebsart Galois Counter Mode (GCM) wird in [54] ein Verfahren zur Erzeugung des Authentisierungsschlüssels aus dem Verschlüsselungsschlüssel beschrieben. Dies bedeutet insbesondere, dass der Authentisierungsschlüssel aus dem Verschlüsselungsschlüssel ableitbar ist. Es sollte stattdessen ein in Abschnitt 12.1 empfohlenes Verfahren zum Einsatz kommen.

10.2 Schlüsselvereinbarung mit Instanzauthentisierung

Wie bereits in der Einleitung zu diesem Kapitel erwähnt, muss eine Schlüsselvereinbarung immer mit einer Instanzauthentisierung kombiniert werden. Wir geben in den folgenden beiden Unterabschnitten sowohl Verfahren an, die sich auf rein symmetrische Algorithmen, als auch auf rein asymmetrische Algorithmen stützen. Für benötigte Hintergrundsysteme siehe Kapitel 11.

Nach dem erfolgreichen Ablauf der vorgestellten Protokolle befinden sich beide Kommunikationspartner im Besitz eines gemeinsamen Geheimnisses. Für die Berechnung symmetrischer Schlüssel für Verschlüsselungs- und Datenauthentisierungsverfahren aus diesem Geheimnis siehe Abschnitt 12.1.

Bemerkung 10.3. Bei der konkreten Umsetzung der vorgestellten Verfahren müssen die folgenden beiden Bedingungen erfüllt werden.

1. Die für die Authentisierung benutzten Zufallswerte müssen bei jeder Durchführung des Protokolls mit großer Wahrscheinlichkeit verschieden sein. Dies kann zum Beispiel dadurch erreicht werden, dass jedes Mal ein Zufallswert der Länge mindestens 100 Bit bezüglich der Gleichverteilung aus $\{0,1\}^{100}$ gewählt wird.
2. Die für die Schlüsselvereinbarung benutzten Zufallswerte müssen mindestens eine Entropie erreichen, die den gewünschten Schlüssellängen der auszuhandelnden Schlüssel entspricht[1].

10.2.1 Symmetrische Verfahren

Grundsätzlich kann jedes Verfahren aus Abschnitt 6.1 zur Instanzauthentisierung mit jedem Verfahren aus Abschnitt 7.1 zum Schlüsselaustausch miteinander kombiniert werden. Die Kombination muss dabei so erfolgen, dass die ausgetauschten Schlüssel tatsächlich authentisiert sind, Man-in-the-Middle-Attacken also ausgeschlossen werden können.

Folgendes Verfahren wird für diese Anwendung empfohlen:

Schlüsselvereinbarung mit Instanzauthentisierung gemäß [27], Abschnitt 8.7.

Tabelle 10.1: Empfohlenes symmetrisches Verfahren zur Schlüsselvereinbarung mit Instanzauthentisierung

Bemerkung 10.4. In [27] wird als Blockchiffrierverfahren TDES und ein auf DES basierendes MAC-Verfahren empfohlen. Es sollte allerdings als Blockchiffrierverfahren eines der in Kapitel 2 empfohlenen Verfahren (inklusive empfohlener Betriebsarten, Paddingverfahren und Verfahren zur Erzeugung von Initialisierungsvektoren) und als MAC eines der in Kapitel 5 empfohlenen Verfahren verwendet werden.

10.2.2 Asymmetrische Verfahren

Wie schon für symmetrische Verfahren kann auch hier grundsätzlich jedes Verfahren aus Abschnitt 6.2 zur Instanzauthentisierung mit jedem Verfahren aus Abschnitt 7.2 zur Schlüsselvereinbarung miteinander kombiniert werden.

Um allerdings Fehler in selbst konstruierten Protokollen auszuschließen, werden die in Tabelle 10.2 aufgelisteten Verfahren zur Schlüsselvereinbarung mit Instanzauthentisierung basierend auf asymmetrischen Verfahren empfohlen.

Bemerkung 10.5. In einigen der in Tabelle 10.2 empfohlenen Standards werden Signaturverfahren und asymmetrische Schlüsselaustauschverfahren vorgeschlagen, die in diesem Dokument nicht empfohlen werden. Dies betrifft vor allem Hashfunktionen und Paddingverfahren. Bei der konkreten Umsetzung der Protokolle sollte darauf geachtet werden, dass lediglich die in diesem Dokument empfohlenen Verfahren zur Anwendung kommen.

Bemerkung 10.6. Bei dem Verfahren EC-KAEG findet keine gegenseitige Authentisierung statt, hier beweist nur eine Partei der anderen im Besitz eines privaten Schlüssels zu sein.

[1] Hier wird davon ausgegangen, dass nur symmetrische Schlüssel ausgehandelt werden

1. Elliptic Curve Key Agreement of ElGamal Type (EC-KAEG), siehe [9].
2. Instanzauthentisierung mit RSA und Schlüsselvereinbarung mit RSA, siehe [25], Annex D, A.14,
3. Instanzauthentisierung mit DSA und Schlüsselvereinbarung mit Diffie-Hellman, siehe [25], Annex D, A.15,
4. Instanzauthentisierung mit ECDSA und Schlüsselvereinbarung mit EC-Diffie-Hellman, siehe [25], Annex D, A.16,
5. Instanzauthentisierung und Schlüsselvereinbarung mit Diffie-Hellman, siehe [27], Anhang A.3,
6. MTI A(0), siehe [35], Key agreement mechanism 5, genauer Annex B.6.

Tabelle 10.2: Empfohlene asymmetrische Verfahren zur Schlüsselvereinbarung mit Instanzauthentisierung

11 Public Key Infrastrukturen

Um einen öffentlichen Schlüssel für ein Signatur- Instanzauthentisierungs- oder asymmetrisches Verschlüsselungsverfahren einer bestimmten Person oder Instanz zuordnen zu können, müssen diese den jeweiligen Kommunikationspartnern über vertrauenswürdige Kanäle bekannt gemacht werden.

Eine Möglichkeit ist, diese öffentlichen Schlüssel paarweise auszutauschen, z.B. im Rahmen eines persönlichen Treffens oder über E-Mail mit anschließender telefonischer Verifikation. Bei zwei Personen bedeutet dies einen Austausch, bei drei Personen drei, bei vier Personen, die untereinander authentisch kommunizieren müssen, schon sechs. Insgesamt werden in einem Netzwerk von n Kommunikationspartnern insgesamt $(n \cdot (n-1))/2$-mal Schlüssel ausgetauscht.

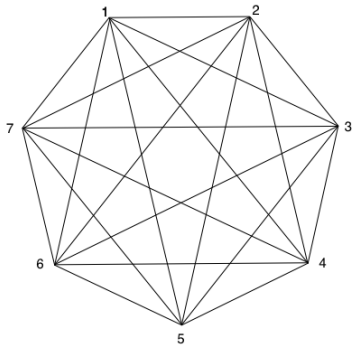


Abbildung 11.1: Schlüsselaustausch zwischen sieben Kommunikationsinstanzen

Deutlich effektiver ist, eine sogenannte Public Key Infrastruktur zu nutzen, um allen Kommunikationspartnern die jeweiligen öffentlichen Schlüssel zuordnen zu können. Zusätzlich ist es in einer solchen Infrastruktur deutlich einfacher, neue Personen oder Instanzen aufzunehmen.

Innerhalb einer Public Key Infrastruktur gibt es eine vertrauenswürdige Instanz, die sogenannte Root Certification Authority (CA), die die öffentlichen Schlüssel aller Beteiligten zertifiziert. Vertraut man dieser Instanz, so kann man auch allen von dieser Root zertifizierten öffentlichen Schlüssel vertrauen.

Die von der Root ausgestellten Zertifikate für die öffentlichen Schlüssel sind elektronische Dateien, die in der Regel folgende Daten enthalten

- Angaben zum Zertifikatsinhaber
- öffentlicher Schlüssel des Zertifikatsinhabers
- verwendeter Algorithmus des Zertifikatsinhabers
- Gültigkeitszeitraum
- Schlüsselnutzung (z.B. Signatur, Authentisierung, Verschlüsselung)

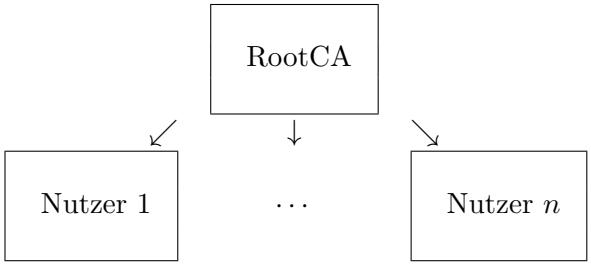


Tabelle 11.1: Schematischer Aufbau einer Public Key Infrastruktur

- Angaben zum Aussteller des Zertifikates
- verwendeter Algorithmus zum Ausstellen des Zertifikates
- Signatur über alle Datengruppen

Die Signatur über alle Datengruppen berechnet der Aussteller des Zertifikates mit seinem geheimen Signaturschlüssel. Mit Hilfe des zugehörigen öffentlichen Schlüssels können dann alle Beteiligten das Zertifikat prüfen. Zertifikate sind also Datenstrukturen, die eine Bindung zwischen einem öffentlichen Schlüssel für ein asymmetrisches Kryptoverfahren (Verschlüsselung, Signatur, Instanzauthentisierung) und dem Schlüsselinhaber herstellen.

Beispiel 11.1. Nutzer 1 signiert ein Dokument und nutzt dazu seinen geheimen Signaturschlüssel. Der zugehörige öffentliche Schlüssel besitzt ein von der Root ausgestelltes Zertifikat. Alle Daten, d.h. das Dokument, die Signatur des Dokuments und sein Zertifikat werden nun an Nutzer 2 geschickt. Um die Signatur zu prüfen, geht dieser wie folgt vor:

- Das Zertifikat wird geprüft.
  - Für die Überprüfung der kryptographischen Gültigkeit nutzt der Prüfer den öffentlichen Schlüssel des Zertifikatsausstellers.
  - Danach prüft er den Namen des Zertifikatsinhabers und die Gültigkeitsdauer.
- Der Prüfer extrahiert aus dem Zertifikat den öffentlichen Schlüssel und prüft damit die Signatur des Dokuments.

Die oben erläuterte Infrastruktur ist aber nur ein vereinfacht dargestelltes Beispiel. Häufig besteht eine Public Key Infrastruktur nicht nur aus einer Root Certification Authority und Endnutzern, sondern aus einer Root, verschiedenen Teil-CAs, die die Zertifikate für Endnutzer bereitstellen und natürlich den eigentlichen Nutzern.

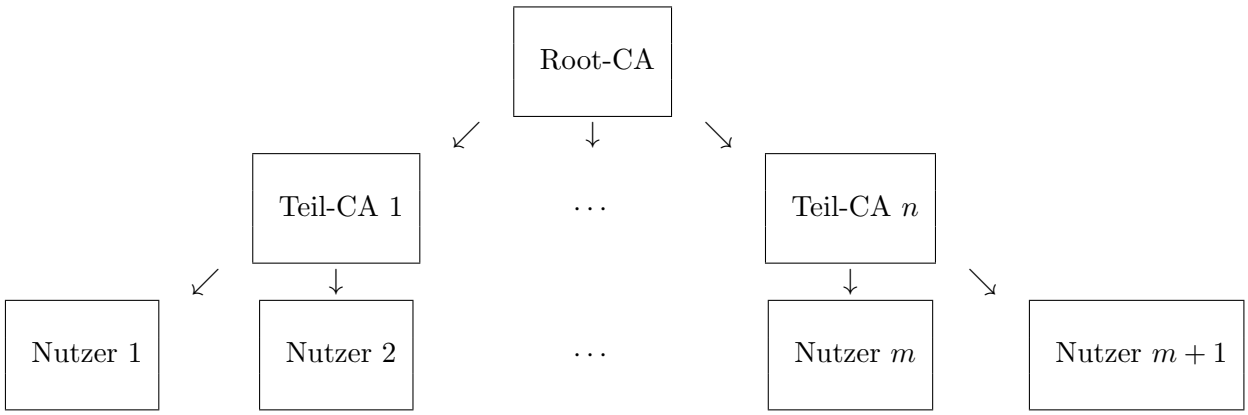


Tabelle 11.2: Schematischer Aufbau einer Public Key Infrastruktur

Die Zertifikatserstellung ist ähnlich zu der in Abbildung 11.1 dargestellten Infrastruktur. Die Root-CA stellt Zertifikate für alle Teil-CAs aus. Die Teil-CAs nutzen ihre von der Root ausge-

stellten Zertifikate, genauer die dazu assoziierten geheimen Schlüssel, um die Zertifikate ihrer Endnutzer auszustellen.

Auch das in Beispiel 11.1 erläuterte Verfahren zur Signaturverifikation muss für die in Abbildung 11.2 dargestellte Infrastruktur leicht angepasst werden. Der Signaturerzeuger sendet nicht nur sein Zertifikat zum Prüfer, sondern auch das Zertifikat der Teil-CA, die sein Zertifikat ausgestellt hat. Der Prüfer verifiziert dann das Zertifikat der Teil-CA mit Hilfe des Root-Zertifikats, überprüft dann das Zertifikat des Signaturerzeugers mit Hilfe des bereits verifizierten Zertifikats der Teil-CA und anschließend kann er die Signatur prüfen. Dies nennt man auch Prüfung einer Zertifikatskette.

Unter bestimmten Umständen müssen Zertifikate zurückgezogen werden. Abhängig vom Einsatz kann dies z.B. dann geschehen, wenn der Zertifikatsinhaber (Endnutzer aber auch Teil-CA) sich nicht an die Sicherheitsstandards hält, die genutzten Algorithmen als unsicher eingestuft werden müssen oder der zu einem Zertifikat assoziierte geheime Schlüssel kompromittiert wurde. Für diese Fälle werden sogenannte Certificate Revokation Lists geführt und von der Root-CA veröffentlicht.

11.1 Certificate Policy und Certificate Practise Statement

Eine Public Key Infrastruktur arbeitet üblicherweise innerhalb eines gemeinsamen Sicherheitsstandards. Dieser ist in einem Dokument, der sogenannter Certification Policy, die von der Root-CA herausgegeben wird, geregelt. Das Dokument beschreibt für alle Teilnehmer der PKI verbindlich Sicherheitsvorgaben für die gesamten Lebensdauer der eingesetzten Schlüssel und Zertifikate, insbesondere:

- Allgemeines
  - Genutzte Algorithmen
  - Schlüssellängen
  - Datenfelder in den Zertifikaten
- Initialisierung
  - Registrierung der Teilnehmer
  - Generierung der Schlüsselpaare
  - Generierung der Zertifikate
  - Schlüssel- und Zertifikatsverteilung
  - Schlüssel-Backup
- Nutzung
  - Zertifikatsneuausstellung
  - Zertifikatsvalidierung
  - Schlüsselupdate
  - Schlüssel-Recovery
- Aufhebung
  - Ablauf des Zertifikates
  - Zurückziehen des Zertifikates
  - Archivierung

Eine Certificate Policy beschreibt, welche Sicherheitsvorgaben eingehalten werden müssen. In einem Certificate Practise Statement wird dann beschrieben, wie diese Vorgaben umgesetzt werden.

Im RFC 3647 [19] wird Aufbau und Inhalt dieser beiden Dokumente im Detail beschrieben. Wir geben im Folgenden die in RFC 3647 empfohlenen Inhalte für diese beiden Dokumente im Überblick an.

1. Introduction
   a) Overview
   b) Document Name and Identification
   c) PKI Participants: Certification authorities, Registration authorities, Subscribers, Relying parties, Other participants
   d) Certificate Usage: Appropriate certificate uses, Prohibited certificate uses
   e) Policy Administration: Organization administering the document, Contact person, Person determining CPS suitability for the policy, CPS approval procedures
   f) Definitions and Acronyms
2. Publication and Repository Responsibilities: Repositories, Publication of certification information, Time or frequency of publication, Access controls on repositories
3. Identification and Authentication (I&A)
   a) Naming: Types of names, need for names to be meaningful; Anonymity or pseudonymity of subscribers; Rules for interpreting various name forms; Uniqueness of names; Recognition, authentication, and role of trademarks
   b) Initial Identity Validation: Method to prove possession of private key; Authentication of organization identity or individual identity; Non-verified subscriber information; Validation of authority; Criteria for interoperation
   c) I&A for Re-key Requests: Identification and authentication for routine re-key; Identification and authentication for re-key after revocation
   d) I&A for Revocation Requests
4. Certificate Life-Cycle Operational Requirements
   a) Certificate Application: Who can submit a certificate application; Enrollment process and responsibilities
   b) Certificate Application Processing: Performing identification and authentication functions; Approval or rejection of certificate applications; Time to process certificate applications
   c) Certificate Issuance: CA actions during certificate issuance; Notification to subscriber by the CA of issuance of certificate
   d) Certificate Acceptance: Conduct constituting certificate acceptance; Publication of the certificate by the CA; Notification of certificate issuance by the CA to other entities
   e) Key Pair and Certificate Usage: Subscriber private key and certificate usage; Relying party public key and certificate usage
   f) Certificate Renewal: Circumstance for certificate renewal; Who may request renewal; Processing certificate renewal requests; Notification of new certificate issuance to subscriber; Conduct constituting acceptance of a renewal certificate; Publication of the renewal certificate by the CA; Notification of certificate issuance by the CA to other entities

g) Certificate Re-key: Circumstance for certificate re-key; Who may request certification of a new public key; Processing certificate re-keying requests; Notification of new certificate issuance to subscriber; Conduct constituting acceptance of a re-keyed certificate; Publication of the re-keyed certificate by the CA; Notification of certificate issuance by the CA to other entities

h) Certificate Modification: Circumstance for certificate modification; Who may request certificate modification; Processing certificate modification requests; Notification of new certificate issuance to subscriber; Conduct constituting acceptance of modified certificate; Publication of the modified certificate by the CA; Notification of certificate issuance by the CA to other entities

i) Certificate Revocation and Suspension: Circumstances for revocation; Who can request revocation; Procedure for revocation request; Revocation request grace period; Time within which CA must process the revocation request; Revocation checking requirement for relying parties; CRL issuance frequency (if applicable); Maximum latency for CRLs (if applicable); On-line revocation/status checking availability; On-line revocation checking requirements; Other forms of revocation advertisements available; Special requirements re key compromise; Circumstances for suspension; Who can request suspension; Procedure for suspension request; Limits on suspension period

j) Certificate Status Services: Operational characteristics; Service availability; Optional features

k) End of Subscription

l) Key Escrow and Recovery: Key escrow and recovery policy and practices; Session key encapsulation and recovery policy and practices

5. Facility, Management, and Operational Controls

a) Physical Security Controls: Site location and construction; Physical access; Power and air conditioning; Water exposures; Fire prevention and protection; Media storage; Waste disposal; Off-site backup

b) Procedural Controls: Trusted roles, Number of persons required per task; Identification and authentication for each role; Roles requiring separation of duties

c) Personnel Controls: Qualifications, experience, and clearance requirements; Background check procedures; Training requirements; Retraining frequency and requirements; Job rotation frequency and sequence; Sanctions for unauthorized actions; Independent contractor requirements; Documentation supplied to personnel

d) Audit Logging Procedures: Types of events recorded; Frequency of processing log; Retention period for audit log; Protection of audit log; Audit log backup procedures; Audit collection system (internal vs. external); Notification to event-causing subject; Vulnerability assessments

e) Records Archival: Types of records archived; Retention period for archive; Protection of archive; Archive backup procedures; Requirements for time-stamping of records; Archive collection system (internal or external); Procedures to obtain and verify archive information

f) Key Changeover

g) Compromise and Disaster Recovery: Incident and compromise handling procedures; Computing resources, software, and/or data are corrupted; Entity private key compromise procedures; Business continuity capabilities after a disaster

h) CA or RA Termination

6. Technical Security Controls

a) Key Pair Generation and Installation: Key pair generation; Private key delivery to subscriber; Public key delivery to certificate issuer; CA public key delivery to relying parties; Key sizes; Public key parameters generation and quality checking; Key usage purposes (as per X.509 v3 key usage field)

b) Private Key Protection and Cryptographic Module Engineering Controls: Cryptographic module standards and controls; Private key (n out of m) multi-person control; Private key escrow; Private key backup; Private key archival; Private key transfer into or from a cryptographic module; Private key storage on cryptographic module; Method of activating private key; Method of deactivating private key; Method of destroying private key; Cryptographic Module Rating

c) Other Aspects of Key Pair Management: Public key archival; Certificate operational periods and key pair usage periods

d) Activation Data: Activation data generation and installation; Activation data protection; Other aspects of activation data

e) Computer Security Controls: Specific computer security technical requirements; Computer security rating

f) Life Cycle Security Controls: System development controls; Security management controls; Life cycle security controls

g) Network Security Controls

h) Timestamping

7. Certificate, CRL, and OCSP Profiles

a) Certificate Profile: Version number(s); Certificate extensions; Algorithm object identifiers; Name forms; Name constraints; Certificate policy object identifier; Usage of Policy Constraints extension; Policy qualifiers syntax and semantics; Processing semantics for the critical Certificate Policies extension

b) CRL Profile: Version number(s); CRL and CRL entry extensions

c) OCSP Profile: Version number(s); CRL and CRL entry extensions

8. Compliance Audit and Other Assessment: Frequency or circumstances of assessment; Identity/qualifications of assessor; Assessor's relationship to assessed entity; Topics covered by assessment; Actions taken as a result of deficiency; Communication of results

9. Other Business and Legal Matters

a) Fees: Certificate issuance or renewal fees; Certificate access fees; Revocation or status information access fees; Fees for other services; Refund policy

b) Financial Responsibility: Certificate issuance or renewal fees; Certificate access fees; Revocation or status information access fees; Fees for other services; Refund policy

c) Confidentiality of Business Information: Scope of confidential information; Information not within the scope of confidential information; Responsibility to protect confidential information

d) Privacy of Personal Information: Privacy plan; Information treated as private; Information not deemed private; Responsibility to protect private information; Notice and consent to use private information; Disclosure pursuant to judicial or administrative process; Other information disclosure circumstances

e) Intellectual Property Rights

f) Representations and Warranties: CA representations and warranties; RA representations and warranties; Subscriber representations and warranties; Relying party representations and warranties; Representations and warranties of other participants

g) Disclaimers of Warranties

h) Limitations of Liability

i) Indemnities

j) Term and Termination: Term; Termination; Effect of termination and survival

k) Individual notices and communications with participants

l) Amendments: Procedure for amendment; Notification mechanism and period; Circumstances under which OID must be changed

m) Dispute Resolution Procedures

n) Governing Law

o) Compliance with Applicable Law

p) Miscellaneous Provisions: Entire agreement; Assignment; Severability; Enforcement (attorneys' fees and waiver of rights); Force Majeure

q) Other Provisions

11.2 Zertifikate

In der Einleitung zu diesem Abschnitt haben wir gesehen, dass Zertifikate die Bindung zwischen einem öffentlichen Schlüssel für ein asymmetrisches Kryptoverfahren (Verschlüsselung, Signatur, Instanzauthentisierung) und dem Schlüsselinhaber herstellen. Ein Zertifikat muss daher den Inhaber des öffentlichen Schlüssels und die austellende Certification Authority eindeutig indentifizieren, die Schlüsselnutzung festlegen, Angaben zum Sicherheitsstandard enthalten (d.h. wo Certificate Policies oder Certificate Practise Statements zu finden sind) usw.

Im Wesentlichen gibt es heute zwei Standards für Zertifikate: X509 und Card Verifiable Certificates (cvc). CV-Zertifikate werden, wie der Name schon sagt, eingesetzt, wenn Komponenten mit beschränkten Resourcen, wie z.B. SmartCards, das Zertifikat prüfen. X.509-Zertifikate werden z.B. für die sichere Kommunikation im Internet, im Rahmen von tsl/ssl zur Serverauthentisierung, eingesetzt. Wir wollen die Struktur dieser Zertifikate im Folgenden kurz erläutern.

X509-Zertifikate werden nach ASN.1 (Abstract Syntax Notation Nr. 1) kodiert und haben die folgende Form:

```
Certificate::= SEQUENCE {
   tbsCertificate       TBSCertificate,
   signatureAlgorithm   AlgorithmIdentifier,
   signatureValue       BITSTRING }
```

TBS steht für To Be Signed, d.h. die ausstellende Certification Authority signiert nur den Inhalt von `TBSCertificate`. Das Feld `signatureAlgoritm` legt den Algorithmus fest, mit dem das Zertifikat signiert wurde und `signatureValue` ist die Signatur von `TBSCertificate`.

In `TBSCertificate` sind dann die eigentlichen oben beschriebenen Inhalte enthalten. Die einzelnen Datenfelder beschreiben wir im Folgenden im Detail.

```
TBSCertificate::= SEQUENCE {
   version                   EXPLICIT Version DEFAULT v1,
   serialNumber              CertificateSerialNumber,
   signature                 AlgorithmIdentifier,
   issuer                    Name,
```

```
   validity                  Validity,
   subject                   Name,
   subjectPublicKeyInfo      SubjectPublicKeyInfo,
   extensions                EXPLICIT Extensions OPTIONAL
                             --If present, version MUST be v3 }
```

Aktuell befindet sich der Standard in der dritten Version, die Versionen v1 und v2 sind aber weiterhin nutzbar. Die genutzte X.509-Version muss in jedem Zertifikat angegeben werden.

```
Version::= INTEGER { v1(0), v2(1), v3(2) }
```

Zertifikate, die von der selben Certification Authority (festgelgt im Feld `issuer`) ausgestellt werden, müssen jeweils unterschiedliche Seriennummern erhalten. Das Feld `CertificateSerialNumber` wird häufig einfach hochgezählt.

```
CertificateSerialNumber::= INTEGER
```

Dieses Feld spezifiziert den Algorithmus, der für die Signierung des vorliegenden Zertifikats genutzt wurde (und enhält somit den selben Eintrag wie das Feld `signatureAlgorithm` der Sequence `Certificate`. Der Object Identifier identifiziert über eine OID den genutzten Algorithmus (z.B. dsaWithSha256). das zweite Datenfeld enthält eventuell benötigte Systemparameter.

```
AlgorithmIdentifier::= SEQUENCE {
   algorithm       OBJECT IDENTIFIER,
   parameters      ANY DEFINED BY algorithm OPTIONAL }
```

Die Datenfelder `issuer` und `subject` beschreiben jeweils die Aussteller und den Inhaber des Zertifikates eindeutig über einen Namen `Name`, der eine Anzahl von Attributen enthält. Mögliche Attributtypen sind

- country
- organization
- organizational unit
- distinguished name qualifier
- state or province name
- common name (z.B. “Marian Margraf”)
- serial number

```
Name ::= CHOICE {
     rdnSequence  RDNSequence }

   RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

   RelativeDistinguishedName ::=
     SET SIZE (1..MAX) OF AttributeTypeAndValue

   AttributeTypeAndValue ::= SEQUENCE {
     type     AttributeType,
     value    AttributeValue }

   AttributeType ::= OBJECT IDENTIFIER

   AttributeValue ::= ANY -- DEFINED BY AttributeType
```

`Validity` beschreibt den Zeitraum, in dem der geheime Schlüssel, für den das Zertifikat ausgestellt wurde, genutzt werden kann.

```
Validity::= SEQUENCE {
   notBefore   Time,
   notAfter    Time }
```

`SubjectPublicKeyInfo` enthält den öffentlichen Schlüssel des Schlüsselpaares, für das das Zertifikat ausgestellt wurde. Wie oben bereits gesehen, beschreibt `AlgorithmIdentifier` den Algorithmus, mit dem das Schlüsselpaar genutzt wird über eine eindeutige OID sowie eventuell benötigte Systemparameter. `subjectPublicKey` enthält dann den eigentlichen öffentlichen Schlüssel.

```
SubjectPublicKeyInfo::= SEQUENCE {
   algorithm           AlgorithmIdentifier,
   subjectPublicKey    BIT STRING }
```

In Version v3 des X.509-Standards gibt es die Möglichkeit, weitere Angaben im Datenfeld `Extensions` aufzunehmen. Heute häufig genutzte Erweiterungen lassen sich gliedern in

- Informationen über Schlüssel (z.B. Schlüsselnutzung) und Sicherheitsrichtlinien (z.B. wo findet man die Sicherheitsrichtlinie Certificate Policy)
- Attribute von Zertifikatsbesitzer und signierender CA (alternative Namen von subject und issuer, mögliche Kontaktdaten für Rückfragen (E-Mailadresse, Fax, Telefon))
- Einschränkungen des Zertifikatspfades (z.B. maximale Pfadkette, d.h. wie viele Zertifikate müssen maximal bis zum Root-Zertifikat geprüft werden

```
Extensions::= SEQUENCE SIZE (1..MAX) OF Extension
```

Dabei hat Extension die folgende Form:

```
Extension::= SEQUENCE {
   extnID     OBJECT IDENTIFIER,
   critical   BOOLEAN DEFAULT FALSE,
   extnValue  OCTET STRING }
```

Das Feld `critical` gibt an, ob eine bestimmte Angabe geprüft werden muss oder das Zertifikat auch ohne die Überprüfung dieser Angabe als gültig akzeptiert werden kann.

In der Vorlesung haben wir bereits gesehen, dass z.B. das RSA-Verfahren sowohl für Verschlüsselung, Datenauthentisierung (Signaturen) und Instanzauthentisierung genutzt werden kann. Insbesondere ist es technisch möglich, für alle drei Einsatzszenarien das selbe Schlüsselpaar zu nutzen. Das dies zu einfachen aber effektiven Angriffen führt, haben wir bereits auf Seite 37 gesehen. Daher sollte ein Schlüsselpaar auch nur für ein Verfahren eingesetzt werden. Dies sollte auch im Zertifikat für das Schlüsselpaar vermerkt werden.

```
KeyUsage::= BIT STRING {
   digitalSignature  (0),
   nonRepudiation    (1),
   keyEncipherment   (2),
   dataEncipherment  (3),
```

```
keyAgreement      (4),
keyCertSign       (5),
cRLSign           (6),
encipherOnly      (7),
decipherOnly      (8) }
```

Darüber hinaus gibt es die Möglichkeit, in Extended Key Usage weitere Nutzungen zu erlauben:

- Code signing
- OCSP signing
- Timestamping

Weitere mögliche Erweiterungen sind:

- Subject Key Identifier
- Authority Key Identifier
- Subject Alternative Name
- Issuer Alternative Name
- Extended Key Usage
- Certificate Policies
- Policy Mappings
- Policy Constraints
- Basic Constraints
- Name Constraints
- CRLDistributionPoints
- Inhibit Any-Policy
- Freshest CRL
- Authority Information Access
- Subject Information Access

12 Zusätzliche Funktionen und Algorithmen

Für einige kryptographische Verfahren werden zusätzliche Funktionen und Algorithmen benötigt, um zum Beispiel Systemparameter zu generieren, oder aus den Ergebnissen von Zufallszahlengeneratoren oder Schlüsseleinigungsverfahren symmetrische Schlüssel zu erzeugen. Diese Funktionen und Algorithmen sind sorgfältig zu wählen, um das geforderte Sicherheitsniveau zu erreichen und somit kryptoanalytische Angriffe zu verhindern.

12.1 Schlüsselableitung

Nach einem Schlüsseleinigungsverfahren sind beide Parteien im Besitz eines gemeinsamen Geheimnisses. Häufig müssen aus diesem Geheimnis mehrere symmetrische Schlüssel, zum Beispiel für die Verschlüsselung und zur Datenauthentisierung, abgeleitet werden.

Folgendes Verfahren zur Schlüsselableitung wird empfohlen:

Key Derivation Function aus [6], Abschnitt 5.6.3.

Tabelle 12.1: Empfohlenes Verfahren zur Schlüsselableitung

Bemerkung 12.1. (*i*) In [6], Abschnitt 5.6.3 wird die Hashfunktion SHA1 zur Ableitung der symmetrischen Schlüssel genutzt. Obwohl diese Hashfunktion in Kapitel 4 nicht empfohlen wird, kann sie für dieses Verfahren eingesetzt werden.

(*ii*) Anstelle der Hashfunktion SHA1 kann auch RIPEMD160 und alle in Kapitel 4 empfohlenen Hashfunktionen eingesetzt werden.

12.2 Generierung von Zufallszahlen für probabilistische asymmetrische Verfahren

Bei den in der Vorlesung vorgestellten asymmetrischen Verfahren wird häufig ein Zufallswert $k \in \{1, \ldots, q-1\}$ benötigt, wobei q die Ordnung der jeweiligen Gruppe ist, in der die Berechnung stattfindet. Wie bereits in Bemerkung 5.5 erwähnt, sollte k möglichst gleichverteilt gewählt werden.

Auf der anderen Seite liefern die in Kapitel 9 beschriebenen Zufallszahlengeneratoren gemäß der Gleichverteilung nur Zufallswerte aus $\{0, 1, \ldots, 2^n - 1\}$, $2^n - 1 \geq q$. Bei einer konkreten Implementierung muss darauf geachtet werden, dass sich die Gleichverteilung auf $\{0, 1, \ldots, 2^n - 1\}$ in $\{1, \ldots, q\}$ wiederfindet.

Wir empfehlen im Folgenden drei Verfahren. Sei dazu $n \in \mathbb{N}$ so, dass $2^{n-1} \leq q < 2^n - 1$ (q habe also eine Bitlänge von n).

Bemerkung 12.2. (*i*) Verfahren 2. und 3. liefern keine Gleichverteilung auf $\{0, \ldots, q-1\}$, die Abweichung ist allerdings so gering, dass dies nach derzeitigem Wissensstand von einem Angreifer nicht ausgenutzt werden kann.

Verfahren 1.

1. Wähle $k \in \{0, 1, \ldots, 2^n - 1\}$ gleichverteilt.
2. `if` $k < q$ `return` k
3. `else goto` 1.

Verfahren 2.

1. Wähle $k' \in \{0, 1, \ldots, 2^{n+64} - 1\}$ gleichverteilt.
2. $k = k' \bmod q$ (d.h. $0 \leq k < q$).
3. `return` k.

Verfahren 3.

1. Wähle $x, y \in \{0, 1, \ldots, 2^n - 1\}$ gleichverteilt.
2. `return` $k = x \cdot y \bmod q$.

Tabelle 12.2: Berechnung von Zufallswerten

(*ii*) Im Gegensatz dazu liefert Verfahren 1. eine Gleichverteilung, hat allerdings den Nachteil, dass unter Umständen mehrere Iterationen notwendig sind. Um zu garantieren, dass das Verfahren nach einer bestimmten Zeit terminiert, wird empfohlen, die Anzahl der Iterationen zu beschränken. Da die Wahrscheinlichkeit dafür, dass ein zufälliger Wert $k \in \{0, 1, \ldots, 2^n - 1\}$ durch 2^{n-1} (und damit auch durch q) von oben beschränkt ist (d.h. $k < 2^{n-1} \leq q$ gilt), gerade $2^{n-1}/2^n = 1/2$ ist, genügen in der Praxis 8 Iterationen. Die Wahrscheinlichkeit, dass das Verfahren tatsächlich einen Wert kleiner q zurückgibt, ist dann ungefähr $1 - 1/2^8$.

12.3 Probabilistische Primzahltests

Bei der Festlegung der Systemparameter für RSA-basierte asymmetrische Verfahren müssen zwei Primzahlen p, q gewählt werden. Für die Sicherheit dieser Verfahren ist es zusätzlich nötig, die Primzahlen geheim zu halten, dies setzt insbesondere voraus, dass p und q zufällig gewählt werden müssen.

Folgendes Verfahren wird zur Erzeugung zufälliger Primzahlen empfohlen. Dabei sei b die gewünschte Bitlänge der Primzahl (hier $b \geq 1024$).

1. Wähle eine zufällige ungerade Zahl p der Bitlänge b.
2. Falls p keine Primzahl ist, gehe zurück zu 1.
3. Gib p aus.

Tabelle 12.3: Empfohlenes Verfahren zur Erzeugung von Primzahlen

Für den zweiten Schritt des obigen Algorithmus werden aus Effizienzgründen meist probabilistische Primzahltests eingesetzt. Der folgende Algorithmus wird empfohlen:

Miller-Rabin, siehe [50], Algorithmus 4.24.

Tabelle 12.4: Empfohlener probabilistischer Primzahltest

Bemerkung 12.3. (*i*) Der Miller-Rabin-Algorithmus benötigt neben der zu untersuchenden Zahl p einen Zufallswert $a \in \{2, 3, \ldots, p-2\}$, die sogenannte Basis. Ist a bezüglich der Gleichverteilung auf $\{2, 3, \ldots, p-2\}$ gewählt, so ist die Wahrscheinlichkeit dafür, dass p zusammengesetzt ist, obwohl der Miller-Rabin-Algorithmus ausgibt, dass p eine Primzahl ist, höchstens $1/4$.

(*ii*) (Worst Case) Um die Wahrscheinlichkeit dafür, dass eine feste Zahl p mittels des Miller-Rabin-Algorithmus als Primzahl erkannt wird, obwohl sie zusammengesetzt ist, auf $1/2^{100}$ zu beschränken, muss der Algorithmus 50-mal mit jeweils unabhängig voneinander bezüglich der Gleichverteilung gewählten Basen $a_1, \ldots, a_{50} \in \{2, 3, \ldots, p-2\}$ aufgerufen werden, siehe Abschnitt 12.2 für empfohlene Verfahren zur Berechnung gleichverteilter Zufallszahlen aus $\{2, 3, \ldots, p-2\}$.

(*iii*) (Average Case) Um eine bezüglich der Gleichverteilung gewählte ungerade Zahl $p \in [2^{b-1}, 2^b - 1]$ auf ihre Primzahleigenschaft zu testen, reichen bei weitem weniger Iterationen des Miller-Rabin-Algorithmus aus, vergleiche [22], [52] und [41], Annex A. So sind für $b = 1024$ lediglich 4 Iterationen notwendig, um mit Wahrscheinlichkeit 2^{-109} auszuschließen, dass p zusammengesetzt ist, obwohl der Miller-Rabin-Algorithmus p als Primzahl erkennt. Auch hier müssen die Basen unabhängig und bezüglich der Gleichverteilung aus $\{2, 3, \ldots, p-2\}$ gewählt werden.

Literaturverzeichnis

[1] AIS 20 (W. Schindler). *Functionality classes and evaluation methodology for deterministic random number generators.* Bundesamt für Sicherheit in der Informationstechnik, 1999. Verfügbar unter `http://www.bsi.de/zertifiz/zert/interpr/aisitsec.htm`.

[2] AIS 31 (W. Schindler). *Functionality classes and evaluation methodology for physical random number generators.* Bundesamt für Sicherheit in der Informationstechnik, 2001. Verfügbar unter `http://www.bsi.de/zertifiz/zert/interpr/aisitsec.htm`.

[3] R. Anderson, E. Biham, and L. Knudsen. Serpent. Verfügbar unter `http://www.cl.cam.ac.uk/~rja14/serpent.html`.

[4] ANSI X9.62. *Triple Data Encryption Algorithm, Modes of Operation*, 1998.

[5] ANSI X9.62. *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 2005.

[6] ANSI X9.63. *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, 2001.

[7] R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Chapman & Hall/CRC, 2005.

[8] Dan Boneh. *Twenty Years of attacks on the RSA Cryptosystem.* Notices of the American Mathematical Society (AMS) 46 (2): 203–213, 1999. `http://crypto.stanford.edu/~dabo/abstracts/RSAattack-survey.html`.

[9] BSI (H. Baier, D. Kügler und M. Margraf). *Elliptic Curve Cryptography based on ISO 15946.* BSI Technical Guideline TR-03111, 2007.

[10] BSI. *Leitfaden für die Auswahl von IT-Sicherheitssystemen für sensible Infrastrukturen, deren Schutz im nationalen Sicherheitsinteresse liegt.* Technische Richtlinie BSI - L 04001,Bundesamt fur Sicherheit in der Informationstechnik, 2005, verfügbar unter `http://www.bsi.bund.de`.

[11] BSI. *Liste der zugelassenen IT-Sicherheitsprodukte -und Systeme.* Bundesamt für Sicherheit in der Informationstechnik, BSI-Schrift 7164, 2008, auf Anfrage vom BSI erhältlich.

[12] BSI. *Liste zertifizierter Produkte.* Bundesamt für Sicherheit in der Informationstechnik. BSI-Schrift 7148, 2008, verfügbar unter `http://www.bsi.bund.de`.

[13] M. Bellare, R. Canetti und H. Krawczyk. *HMAC: Keyed-Hashing for Message Authentication.* RFC 2104, 1997.

[14] BNetzA, Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (übersicht über geeignete Algorithmen).

[15] D. Boneh und G. Durfee. *Cryptanalysis of RSA with private key d less then* $N^{0.292}$. Eurocrypt 1999, LNCS 1592, 1999.

[16] S. Bradner. *Key words for use in RFCs to indicate requirement levels (RFC2119)*, 1999. Verfügbar unter `http://www.ietf.org/rfc/rfc2119.txt`.

[17] BSI: *IT-Grundschutzkataloge,* 2006.

[18] J. Buchmann, L.C. Coronado García, E. Dahmen, M. Döring und E. Klintsevich. *CMSS – An Improved Merkle Signature Scheme.* Progress in Cryptography – INDOCRYPT 2006, Springer LNCS 4329, pp. 349-363 (2006).

[19] S. Chokhani, W. Ford, R. Sabett, C. Merrill und S. Wu. *Internet X.509 Public Key Infrastructure - Certificate Policy and Certification Practices Framework.* RFC 3647, 2003.

[20] D. Coppersmith, M. Franklin, J. Patarin und M. Reiter. *Low-exponent RSA with related messages.* Technical Report IBM RC 20318 T.J. Watson Research Lab, December 27, 1995.

[21] J.-S. Coron, D. Naccache und J. Stern. *On the Security of RSA-Padding.* Crypto 99, LNCS 1666, 1999.

[22] I. Damgard, P. Landrock und C. Pommerance. *Average Case Error Estimates for the Strong Provable Prime Test,* Mathematics of Computation, v. 61, No. 203, pp. 177-194, 1993.

[23] M. Daum und S. Lucks. *The Story of Alice and Bob,* "Rump Session"-Vortrag Eurocrypt 2005, `http://www.cits.rub.de/imperia/md/content/magnus/rump_ec05.pdf`.

[24] DIN V66291. *Spezifikation der Schnittstelle zu Chipkarten mit Digitaler Signatur-Anwendung/Funktion nach SigG und SigV,* Annex A, 2.1.1, 1999.

[25] DIN V66291-4. *Chipkarten mit Digitaler Signatur-Anwendung/Funktion nach SigG und SigV, Teil 4: Grundlegende Sicherheitsdienste,* DIN NI-17.4, 2002.

[26] ECC Brainpool. *ECC Brainpool Standard Curves and Curve Generation*, 2005. Verfügbar unter `http://www.ecc-brainpool.org/ecc-standard.htm`, Version 1.0.

[27] European Committee for Standardization. *Application Interface for smart cards used as Secure Signature Creation Devices - Part 1: Basic requirements* 08.03.2004.

[28] Federal Information Processing Standards Publication 180-2 (FIPS PUB 180-2). *Secure Hash Standard*, 2002.

[29] Federal Information Processing Standards Publication 186-2 (FIPS PUB 186-2). *Digital Signature Standard (DSS)*, 2000.

[30] Federal Information Processing Standards Publication 197 (FIPS PUB 197). *Advances Encryption Standard (AES)*, 2001.

[31] M. Gebhardt, G. Illies und W. Schindler. *A Note on the Practical Value of Single Hash Collisions for Special File Formats,* Sicherheit 2006, Köllen-Verlag, LNI P-77 (2006), 333-344. Erweiterte Version: NIST Cryptographic Hash Workshop 2005, `http://www.csrc.nist.gov/pki/Hashworkshop/2005/Oct31_Presentations/Illies_NIST_05.pdf`.

[32] IEEE P1363. *Standard Specifications for Public Key Cryptography*, 2000.

[33] ISO/IEC 10118-3-2004. *Information technology – Security techniques – Hash functions – Part 3: Dedicated hash functions*, 2003.

[34] ISO/IEC 11770-3-1999. *Information technology – Security techniques – Key management – Part 2: Mechanisms using symmetric techniques*, 1996.

[35] ISO/IEC 11770-3-1999. *Information technology – Security techniques – Key management – Part 3: Mechanisms using asymmetric techniques*, 1999.

[36] ISO/IEC 14888-3-1999. *Information technology – Security techniques – Digital signatures with appendix – Part 3: Certificate-based mechanisms*, 1999.

[37] ISO/IEC 15946-1-2002. *Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 1: General*, 2002.

[38] ISO/IEC 15946-2-2002. *Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 2: Digital signatures*, 2002.

[39] ISO/IEC 15946-3-2002. *Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 3: Key establishment*, 2002.

[40] ISO/IEC 15946-4-2004. *Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 4: Digital signatures giving message recovery*, 2004.

[41] ISO/IEC 18032. *IT security techniques – Prime number generation,* 2005.

[42] ISO/IEC 9796-2-2002. *Information technology – Security techniques – Digital Signature Schemes giving message recovery – Part : Integer Factorization based mechanisms*, 2002.

[43] D. Kügler und M. Margraf. *PACE: Password Authenticated Connection Establishment.* preprint (2007).

[44] A.K. Lenstra und E.R. Verheul. *Selecting Cryptographic Key Sizes.* J. Cryptology 39, 2001.

[45] ISO/IEC 7816-8-2004. *Identification cards – Integrated circuit cards – Part 8: Commands for security operations*, 2004.

[46] ISO/IEC 7816-4-2005. *Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange*, 2005.

[47] M. Margraf. *Kryptographische Verfahren: Empfehlungen und Schlüssellangen.* Technische Richtlinie TR-02102, Bundesamt fur Sicherheit in der Informationstechnik, 2008, verfügbar unter `http://www.bsi.bund.de`.

[48] M. Margraf und D. Rappe. *Leitfaden zur Erstellung eines Kryptokonzeptes.*

[49] A. Menezes. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1993.

[50] A. Menezes, P. van Oorschot und O. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[51] NIST. Recommendation for Block Cipher Modes of Operation: Methods and Techniques, Special Publication SP800-38A, National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce, 2001.

[52] NIST. Two Methods to Calculate the Required Number of Rounds of Miller-Rabin Testing, Januar 2007. `csrc.nist.gov/groups/ST/toolkit/documents/dss/Miller-RabinTrails(12Jan07).doc`

[53] NIST. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, Special Publication SP800-38B, National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce, 2001.

[54] NIST. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) for Confidentiality and Authentication, Special Publication SP800-38D, National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce, 04.2006.

[55] P. Nguyen and I. Shparlinski. The insecurity of the elliptic curve signature algorithm with partially known nonces. *Designs, Codes and Cryptography*, 30(2):201–217, 2003.

[56] R. Housley. Cryptographic Message Syntax (CMS). *RFC 3852,* 2004.

[57] S. Kent. IP Encapsulating Security Payload (ESP). *RFC 4303*, 2005.

[58] PKCS #1 v2.1: RSA Cryptographic Standard, 14.06.2002.

[59] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. Twofish: A 128-Bit Block Cipher. Verfügbar unter `http://www.schneier.com/paper-twofish-paper.html.`

[60] A. Shamir. *How to share a secret.* Communications of the ACM, 22 (1979), 612-613.

[61] X. Wang, Y.L. Yin, and H. Yu. Collision search attacks on SHA-1. In *Proceedings of Crypto 2005.* Springer Verlag, 2005.

Abbildungsverzeichnis

Tabellenverzeichnis