Technische Universität Braunschweig

Diplomarbeit

Sichere Log-Dateien auf Grundlage kryptographisch verketteter Einträge

Stefan Konst

August 2000

Institut für Theoretische Informatik

Prof. Dr. Dietmar Wätjen

Erklärung

Ich versichere, die vorliegende Arbeit selbständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Braunschweig, den 9. August 2000

Kurzfassung

Diese Arbeit wird von der Frage geleitet, wie die Authentizität, Reihenfolge und Vollständigkeit der Einträge einer Log-Datei sichergestellt werden kann, wobei die Log-Datei um weitere Einträge erweiterbar sein muß.

Dazu wird zunächst eine allgemeine Theorie für kryptographische Verkettungen erarbeitet, bei der die Ecken eines beliebigen Graphen, entsprechend den zwischen ihnen existierenden Kanten, mittels kryptographischer Verfahren miteinander verkettet werden. Dadurch soll kein Unberechtigter Ecken unbemerkt manipulieren oder Ecken bzw. Kanten unbemerkt aus dem Graphen entfernen können. Die Konstruktionsvorschrift für die kryptographische Verkettung des Graphen kann dabei so angelegt werden, daß der Graph und entsprechend dessen kryptographische Verkettung um weitere Ecken und Kanten erweitert werden können.

Auf Grundlage der Theorie wird schließlich ein spezielles Schema zur kryptographischen Verkettung eines gerichteten Hamiltonschen Weges vorgestellt. Mit Hilfe dieses Schemas kann die Authentizität und Reihenfolge der Einträge einer Log-Datei überprüft und das Fehlen von Einträgen erkannt werden. Da sich das Schema in Kombination mit verschiedenen kryptographischen Hilfsfunktionen sowohl mittels symmetrischer Kryptosysteme als auch mittels Public-Key-Kryptosystemen realisieren läßt, werden dadurch verschiedene Wege für die kryptographische Verkettung der Einträge von Log-Dateien aufgezeigt.

Abstract

This work is conduced by the question of how authenticity, order and completeness of the entries of a log file can be guaranteed, concerning the fact that an enlargement of the log file must be possible.

First of all a theory of cryptographic chaining is worked out for responding this question. Therefore the vertices of a graph, corresponding to the edges between them, are chained by cryptographic methods in such a way that without being noticed, no attacker can manipulate vertices or remove vertices or edges out of the graph. The construction scheme of the cryptographic chaining of the graph can be realized in such a way that the graph and correspondingly its cryptographic chaining are enlargable by other vertices and edges.

By the foundation of this theory a special pattern for cryptographic chaining of a directed Hamiltonian way is presented. By means of this pattern, the authenticity and the order of the entries of a log file can be checked and missing entries can be recognized. Since the pattern is realizable in combination with various cryptographic support functions by symmetric-key cryptosystems and public key cryptosystems, various ways for a cryptographic chaining of the entries of log files are shown.

Inhaltsverzeichnis

Abbildungsverzeichnis

Einleitung

Viele automatisierte Prozesse legen Statusmeldungen oder bestimmte Zwischenergebnisse, die während der Abläufe der Prozesse entstehen, in Log-Dateien ab. Die Motivation dafür besteht darin, diese Informationen zu einem späteren Zeitpunkt auswerten zu können, weil etwa Probleme aufgetreten sind, die den Rechnerbetrieb stören.

Zum Beispiel können in einem vernetzten Rechner die Namen der Benutzer, die sich dort anmelden, in einer Log-Datei gespeichert werden und ebenso die Namen und Optionen der Befehle, die sie aufrufen. Treten nun während des Betriebs Störungen oder Unstimmigkeiten auf, kann man versuchen, die Ursache mit Hilfe der Log-Dateien zu ergründen. Beispielsweise könnte man feststellen, daß sich jemand unter dem Namen eines Benutzers angemeldet hat, von dem man annimmt, daß dieser zu diesem Zeitpunkt nicht dazu in der Lage sein sollte. Dies wäre ein erster Hinweis darauf, daß eventuell jemand in den Rechner eingedrungen ist und die Störungen oder Unstimmigkeiten verursacht hat. Stellt sich heraus, daß es sich tatsächlich um einen Eindringling handelt, kann man mit entsprechenden Gegenmaßnahmen reagieren.

Für die Log-Datei muß allerdings die Frage gestellt werden, die auch die grundlegende Fragestellung für diese Arbeit darstellt:

> *Wie kann die Authentizität, Reihenfolge und Vollständigkeit der Einträge einer Log-Datei sichergestellt werden?*

Denn wenn es jemand geschafft hat, in einen Rechner einzudringen, so könnte er auch in der Lage und von dem Willen beseelt sein, seine Spuren zu verwischen, indem er einfach die Log-Dateien des Rechners entsprechend manipuliert.

Ziel Im folgenden werden von mir Verfahren erarbeitet, mit deren Hilfe die Authentizität und Reihenfolge der Einträge einer Log-Datei sichergestellt werden kann. Ein weiteres Ziel ist, das Fehlen von Einträgen erkennbar zu machen, da die Vollständigkeit letztendlich nicht garantiert werden kann. Die Verfahren werden unter verschiedenen Gesichtspunkten miteinander verglichen, wobei der Schwerpunkt auf der *kryptographischen Verkettung* der Einträge liegt. Um die Praxistauglichkeit testen und demonstrieren zu können, werden ausgewählte Verfahren in einem Beispielprogramm für sichere Log-Dateien implementiert.

Kryptographische Verkettung Im allgemeinen Sinne soll unter einer kryptographischen Verkettung verstanden werden, daß die Ecken eines beliebigen Graphen, entsprechend den zwischen ihnen existierenden Kanten, mittels

kryptographischer Verfahren miteinander verkettet werden. Dadurch kann beispielsweise verhindert werden, daß ein Angreifer Ecken oder Kanten unerkannt entfernen oder hinzufügen kann. Die eigentliche Idee der kryptographischen Verkettung besteht dabei darin, daß die Ecken als eigenständige Objekte erhalten bleiben und der Graph nicht in Form eines monolithischen Gebildes gesichert wird.

Eine Log-Datei kann nun, wie in Abbildung 1 zu sehen ist, durch einen gerichteten Graphen dargestellt werden. Die Ecken des Graphen repräsentieren dabei jeweils einen Eintrag der Log-Datei, während die Kanten jeweils auf die Ecke des nächsten Eintrags zeigen. Dieser Graph hat einen gerichteten Hamiltonschen Weg, der mit *Eintrag* 1 beginnt. Dadurch kann die *Aufgabenstellung* an die kryptographischen Verkettung für die oben genannten Ziele weiter verfeinert werden:

1. Kein Unberechtigter soll Ecken unbemerkt manipulieren können, um die Authentizität der Einträge sicherzustellen.

2. Kein Unberechtigter soll Ecken oder Kanten unbemerkt aus dem Weg entfernen oder in den Weg einfügen können, um die Reihenfolge der Einträge sicherzustellen und das Fehlen von Einträgen erkennbar zu machen.

3. Ein Berechtigter soll die Einhaltung der Bedingungen 1 und 2 überprüfen können.

4. Neue Ecken und Kanten sollen an das Ende des Weges angefügt werden können, wobei danach wieder die Bedingungen 1 und 2 gelten müssen. Damit können neue Informationen in der Log-Datei abgelegt und gleichfalls geschützt werden.

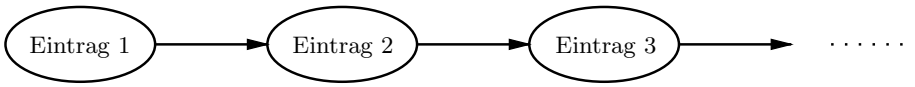

Abbildung 1: Darstellung einer Log-Datei als gerichteter Graph

Übertragen auf die Datenstrukturen der Informatik bedeutet dies, eine einfach verkettete Liste zu schützen, wobei die Elemente der Liste die Ecken des Weges darstellen und die Zeiger auf das jeweils nächste Element die entsprechenden Kanten des Weges.

Grenzen und Einordnung Eine Grenze der Verwendbarkeit besteht darin, daß die kryptographische Verkettung nicht den Erhalt aller Einträge sicherstellen kann (s. a. [SK98]). Es geht also primär nicht darum, daß alle Einträge erhalten bleiben. Vielmehr soll gerade die Manipulation oder das Fehlen einiger oder aller Einträge mittels der kryptographischen Verkettung erkennbar werden. Daher kann das Ganze auch für den Bereich der *Intrusion Detection*[1] eingesetzt werden (s. a. [SK99]). Ob und in welchem Umfang Einträge nach einem Einbruch erhalten bleiben, hängt zumeist einzig und allein vom Willen des

[1] Erkennen von Einbrüchen in einen Rechner.

Eindringlings ab, wenn einmal davon abgesehen wird, daß dieser eine Log-Datei übersieht. Die Existenz aller korrekten Einträge wäre nur eine zusätzliche Hilfe, mit der der Umfang des Schadens eines Einbruchs analysiert und eingegrenzt werden könnte.

Außerdem kann die kryptographische Verkettung nicht verhindern, daß manipulierte Einträge an das Ende der Log-Datei angefügt werden (s. a. [SK98]). Das heißt, daß die Authentizität nur für die Einträge sichergestellt werden kann, die vor einem Einbruch in die Log-Datei aufgenommen wurden. Im Idealfall stellt der letzte authentische Eintrag denjenigen Befehl dar, der die weitere authentische Protokollierung der Befehle beendet. Auch wenn es nicht gelingt, in den Rechner einzubrechen, kann durch das ungehemmte Anfügen neuer Einträge ein *Denial-of-Service*-Angriff[2] gegen die Log-Datei geführt werden. So könnte zum einen der Speicherplatz für die Log-Datei aufgebraucht werden oder zum anderen die Auswertung aller Einträge verlangsamt/erschwert werden. Beide Fälle sind allerdings relativ leicht bemerkbar und sollen hier nicht weiter behandelt werden.

Gliederung Nach dieser Einleitung wird in Kapitel 1 der Begriff Log-Datei formalisiert und auf weitere benötigte Grundlagen eingegangen. Kapitel 2 stellt die Begriffe Sicherheit und Authentizität zusammenfassend dar. In Kapitel 3 wird von mir eine allgemeine Theorie für kryptographische Verkettungen vorgestellt, aus der von mir in Kapitel 4 verschiedene Möglichkeiten für sichere Log-Dateien abgeleitet werden. Danach bildet Kapitel 5 mit einer Zusammenfassung und einem Ausblick auf mögliche Ergänzungen den Abschluß des theoretischen Teils, während in Anhang A noch das bereits erwähnte Beispielprogramm beschriebenen wird.

Voraussetzungen Alle wichtigen Begriffe werden im Verlauf der weiteren Ausführungen dargestellt. Trotzdem wird es sicherlich vorteilhaft sein, sich bereits mit kryptographischen Verfahren und den verschiedenen Blickwinkeln ihrer Anwendung beschäftigt zu haben.

Von besonderem Vorteil dürfte die Kenntnis der Arbeiten von Haber und Stornetta [HS91, BHS93, HS97], Anderson [And96] sowie Schneier und Kelsey [KS96, KS99, SK97a, SK97b, SK98, SK99] sein, wobei vor allem [HS91] und [SK98] für Zeitstempel und Log-Dateien hervorzuheben sind. Dabei ist die in Kapitel 3 von mir erarbeitete Theorie dazu geeignet, diese Arbeiten in einen allgemeinen Zusammenhang einzuordnen, der so nicht aus diesen Arbeiten hervorgeht.

Erwähnt werden muß sicherlich auch die Arbeit von Merkle [Mer89], die eigentlich aus dem Jahre 1979 stammt. In ihr wird, zur Sicherung der Authentizität von öffentlichen Schlüsseln für Public-Key-Kryptosysteme, mittels einer Einwegfunktion ein *Authentication Tree*[3] aufgebaut.

Für die Beschreibung des Beispielprogramms wird auf die in [Str98] eingesetzte Terminologie zurückgegriffen.

[2] Überlastung eines Dienstes durch einen Angreifer mit dem Ziel, daß eine normale Nutzung des Dienstes nicht mehr möglich ist.

[3] S. [MOV97] auf den Seiten 556–559.

Kapitel 1

Grundlagen

In diesem Kapitel werden die allgemeinen Grundlagen dargestellt, auf die im weiteren Verlauf Bezug genommen wird. Dazu wird in Abschnitt 1.1 zunächst an den Unterschied zwischen Listen und Mengen erinnert und danach in Abschnitt 1.2 der Begriff Log-Datei formalisiert. Es folgt Abschnitt 1.3 über die Transformation eines Wortes in eine Zahl und wieder zurück, die für die Implementierung des in der Einleitung erwähnten Beispielprogramms für eine sichere Log-Datei (s. S. 1 und Anhang A) verwendet wird. Anschließend folgen in den Abschnitten 1.4, 1.5 und 1.6 Definitionen und Sätze aus den Bereichen der Graphentheorie, die für die Einordnung der kryptographischen Verkettung benötigt werden, sowie der Codierungstheorie und der Kryptologie, die für die Formalisierungen der Begriffe Sicherheit und Authentizität gebraucht werden.

1.1 Listen und Mengen

Wie in [AU96] beschrieben, ist eine *Liste* eine endliche Folge von null oder mehr Elementen eines gegebenen Typs. Im Gegensatz zu einer *Menge* ist die Reihenfolge der Elemente bei einer Liste von Bedeutung. So unterscheiden sich die Listen $(1, 2, 3)$ und $(3, 2, 1)$, während $\{1, 2, 3\}$ und $\{3, 2, 1\}$ die gleiche Menge auf zwei verschiedene Arten darstellen. Weiterhin kann sich in einer Liste ein Element beliebig oft wiederholen, während es in einer Menge nur einmal vorkommen kann. Kommt ein Element mehrfach in einer Menge vor, spricht man von einer *Multimenge*. Wird eine *Liste als Menge* verwendet, werden die Elemente der Menge anhand ihrer Positionen innerhalb der Liste unterschieden. Eigens für die *Menge der natürlichen Zahlen* wird das Symbol $\mathbb{N}$ verwendet, und es sei $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

1.2 Log-Datei

Die folgende Definition 1.2.1 liefert eine Formalisierung des Begriffs Log-Datei. In den nachfolgenden Absätzen werden dann auf Grundlage dieser Definition verschiedene Aspekte einer Log-Datei diskutiert.

Definition 1.2.1. Ein *Eintrag* e sei ein Paar (t_e, d_e), bestehend aus einem Zeitpunkt t_e und beliebigen Daten d_e. Eine *Log-Datei* $\mathcal{L}$ sei eine nach der Zeit t geordnete Liste $(e_1, ..., e_n)$ von Einträgen $e_i,\ i = 1, ..., n,\ n \in \mathbb{N}_0$, mit $t_{e_i} < t_{e_{i+1}}$. $\mathcal{L}$ hat die *Länge* $|\mathcal{L}| = n$, und speziell für $n = 0$ erhält man die *leere Log-Datei* $\varepsilon_{\mathcal{L}} = ()$ mit $|\varepsilon_{\mathcal{L}}| = 0$. □

Zeitpunkt und Daten In t_e wird der Zeitpunkt festgehalten, zu welchem der Eintrag an das Ende der Log-Datei angehängt wird. d_e enthält die eigentlichen (Nutz-)Daten, wie eine Statusmeldung oder ein Zwischenergebnis. In dieser Definition wurden die Datentypen von t_e und d_e bewußt nicht näher spezifiziert, da sie für die hier vorgestellten Verfahren nicht von Bedeutung sind. Entscheidend ist vielmehr, daß die in t_e und d_e gespeicherten Informationen erkenn- bzw. auswertbar sind und sie von völlig willkürlich (zufällig) gewähltem Rauschen unterscheidbar sind. Sind die Datentypen einer konkreten Implementierung der zu speichernden Informationen inkompatibel zu denen von t_e und d_e, müßten sie, wie zum Beispiel im nachfolgenden Abschnitt 1.3 beschrieben, konvertiert werden.

Logische und physische Repräsentation Weiterhin muß zwischen logischer und physischer Repräsentation einer Log-Datei unterschieden werden. Logisch gesehen sind Log-Dateien wegen $t_{e_i} < t_{e_{i+1}}$ immer aufsteigend nach der Zeit geordnet, während sich die einzelnen Einträge an physisch völlig ungeordneten Positionen im Speicher befinden können. Die Verbindung zwischen logischer und physischer Repräsentation ist dabei völlig von der Implementierung abhängig. Für den Rest der Arbeit soll daher davon ausgegangen werden, daß die logische Repräsentation der Log-Datei gemeint ist, wenn nur von „Log-Datei“ die Rede ist. Sollte die physische Repräsentation gemeint sein, so wird diese explizit erwähnt. Außerdem ist bei dieser Unterscheidung zu beachten, daß eine Log-Datei logisch gesehen unendlich viele Einträge $e_i,\ i = 1, ..., \infty$, aufnehmen kann, die selbst wieder unendlich groß sein können. Physisch gesehen wird die Anzahl der Einträge und deren Größe allerdings durch die Speicherkapazität des Rechners begrenzt, auf der sich die Log-Datei befindet.

Unterscheidung der Einträge Alle Einträge unterscheiden sich dadurch, daß es laut Definition ($t_{e_i} < t_{e_{i+1}}$) keine Einträge mit gleichem Zeitpunkt t_e gibt:

$$t_{e_i} \neq t_{e_j} \Longleftrightarrow i \neq j,\ i, j = 1, ..., n.$$

Für eine Identifizierung der einzelnen Einträge ist es deshalb unerheblich, welche Nutzdaten d_e die Einträge enthalten. So können die Nutzdaten auch überall gleich sein ($d_{e_i} = d_{e_j}$), und trotzdem sind die einzelnen Einträge unterscheidbar.

Auf der anderen Seite reicht für die Kennzeichnung des Zeitpunkts ein einfacher ganzzahliger Zähler, der für jeden neuen Eintrag um eins erhöht wird, da sich dadurch bereits die Anordnung $t_{e_i} < t_{e_{i+1}}$ festlegen läßt. Wenn die unveränderliche Reihenfolge der Elemente in der Log-Datei garantiert werden kann, kann soweit gegangen werden, daß auf die explizite Speicherung des Zeipunkts verzichtet wird und dafür die Position i des Eintrags innerhalb der Log-Datei als Zeitpunkt verwendet wird. Die Wahl des richtigen Typs für den

Zeitpunkt hängt demnach vor allem von den Anforderungen an den Informationsgehalt des Zeitpunkts ab.

Funktionen für eine Log-Datei Für die Arbeit mit einer Log-Datei $\mathcal{L}$ werden zumindest drei Funktionen benötigt:

Definition 1.2.2. Es sei:

- $INIT()$ eine *Initialisierungsfunktion*, die eine leere Log-Datei $\varepsilon_{\mathcal{L}}$ anlegt,
- $APPEND(e_{n+1})$ eine *Anfügefunktion*, die einen neuen Eintrag e_{n+1} an das Ende n von $\mathcal{L}$ anhängt, und
- $e_i = READ(i)$ eine *Lesefunktion*, die den Eintrag e_i von der Position i aus $\mathcal{L}$ zurückliefert. □

Zu $APPEND$ sei zu bemerken, daß sich die Position des Endes n nach dem Anfügen um eins erhöht (kurz: $n_{(neu)} = n_{(alt)} + 1$).

1.3 Wort- und Zahltransformation

Dieser Abschnitt definiert den Datentyp Wort auf Grundlage des Datentyps Buchstabe und beschreibt, wie der Datentyp Wort in den Datentyp Zahl transformiert werden kann. Eine Transformation ist notwendig, da diese beiden Datentypen inkompatibel zueinander sind. Nebenbei wird auf eine typische Verwendung dieser Transformation für eine Log-Datei eingegangen.

Definition 1.3.1. Eine endliche nichtleere Menge V von Buchstaben sei ein *Alphabet*, $V^* = \{b_1...b_n \mid b_i \in V,\ i = 1, ..., n,\ n \in \mathbb{N}_0\}$ die *Menge der Wörter über* V und $w = b_1...b_n$ ein *Wort über* V (eine Folge von Buchstaben, ein Vektor). w hat die *Länge* $|w| = n$ und für $n = 0$ erhält man das *leere Wort* ε_w mit $|\varepsilon_w| = 0$ (s. a. [Wät94, Wät99a]). □

Transformation Für eine konkrete Implementierung der im vorhergehenden Abschnitt 1.2 definierten Log-Datei $\mathcal{L}$ seien $t_e, d_e \in \mathbb{N}_0$ für alle $e \in \mathcal{L}$. Soll nun statt einer Zahl $x \in \mathbb{N}_0$ ein Wort $w \in V^*$ in d_e gespeichert werden, kann dies mittels einer zweistufigen Transformation

$$t = t_2 \circ t_1 :\ V^* \longrightarrow \mathbb{N}_0$$

erfolgen:

1. Zunächst wird w mittels einer bijektiven Abbildung

$$t_1 :\ V^* \longrightarrow \bigcup_{n=0}^{\infty} X^n \quad \text{mit} \quad w \longmapsto f = (x_1, ..., x_n),$$

wobei $X \subset \mathbb{N}_0,\ x_i \in X,\ i = 1, ..., n,\ n \in \mathbb{N}_0$ gilt, in eine Folge von Zahlen f transformiert. Für ε_w mit $n = 0$ ist dies eine leere Folge von Zahlen $\varepsilon_f = ()$. Ansonsten wird V bijektiv auf X abgebildet, wobei jedem b_i bijektiv ein x_i zugeordnet wird. Daher ist t_1 ein Monoidhomomorphismus (s. a. [Bos96]).

2. Danach wird die Folge von Zahlen f mittels einer weiteren bijektiven Abbildung

$$t_2 : \bigcup_{n=0}^{\infty} X^n \longrightarrow \mathbb{N}_0 \quad \text{mit} \quad (x_1, ..., x_n) \longmapsto d_e = t_2(x_1, ..., x_n)$$

in die Zahl d_e überführt.

Die Vorbereitung für t_1 kann zum Beispiel so aussehen, daß die Elemente (Buchstaben) von V von 0 bis $m-1$ mit $m = |V|$ und $m \in \mathbb{N}$ numeriert werden. Die eigentliche Transformation t_1 besteht dann darin, daß die Elemente durch ihre jeweilige Nummer (eine Zahl) ersetzt werden. Darauf aufbauend kann die Transformation t_2 auf Grundlage der g-adischen Entwicklung (s. a. [Bun96, Lei89]) erfolgen. Hierfür wird angenommen, daß f eine Zahl eines Zahlensystems zur Basis m darstellt, wobei $x_k, ..., x_n = 0$ mit $k = 1, ..., n$ und ε_f besonders berücksichtigt werden müssen. Daraus folgt:

$$d_e = \begin{cases} \left(\sum_{i=1}^{n} x_i m^{i-1}\right) + \left(\sum_{i=1}^{n} m^{i-1}\right) = \sum_{i=1}^{n} (x_i + 1) m^{i-1} & \text{falls} \quad n > 0, \\ 0 & \text{falls} \quad n = 0. \end{cases}$$

Umkehrtransformation Entsprechend muß bei einer Auswertung die Zahl d_e wieder in eine Folge von Buchstaben w mit

$$t^{-1} = t_1^{-1} \circ t_2^{-1} : \mathbb{N}_0 \longrightarrow V^*$$

zurücktransformiert werden:

1. Zunächst wird d_e mittels der Umkehrabbildung

$$t_2^{-1} : \mathbb{N}_0 \longrightarrow \bigcup_{n=0}^{\infty} X^n \quad \text{mit} \quad d_e \longmapsto (x_1, ..., x_n) = (t_2^{-1}(d_e)_1, ..., t_2^{-1}(d_e)_n)$$

wieder in eine Folge von Zahlen f zurücktransformiert.

2. Danach wird die Folge von Zahlen f mittels der Umkehrabbildung

$$t_1^{-1} : \bigcup_{n=0}^{\infty} X^n \longrightarrow V^*$$

wieder in w überführt, wobei für ε_f wieder ε_w eingesetzt wird und für jedes x_i das entsprechende b_i.

In diesem Fall folgt daraus:

$$f = \begin{cases} (x_1, ..., x_n) & \text{falls} \quad d_e > 0, \\ \varepsilon_f & \text{falls} \quad d_e = 0, \end{cases}$$

$$\text{mit} \quad x_i = t_2^{-1}(d_e)_i = \left(\frac{d_e - \sum_{j=1}^{i} m^{j-1}}{m^{i-1}} \right) \bmod m \quad \text{für} \quad i = 1, ..., n.$$

Bei der obigen in Klammern stehenden Division für x_i muß beachtet werden, daß es sich wegen $d_e, m \in \mathbb{N}_0$ um eine Division mit Rest handelt, wobei der Rest nicht weiter berücksichtigt wird. t_1^{-1} besteht dann nur noch darin, die jeweiligen Nummern (die Zahlen) wieder durch ihre entsprechenden Elemente zu ersetzen.

Abschluß Abschließend demonstrieren Beispiel 1.3.1 und Beispiel 1.3.2 anhand konkret durchgerechneter Werte die einzelnen Schritte der Transformation eines Wortes w in die Zahl d_e und der anschließenden Umkehrtransformation von d_e nach w.

Beispiel 1.3.1. Es sei $V = \{a, b\}$ und daher $m = 2$, woraus $X = \{0, 1\} \subset \mathbb{N}_0$ folgt. $s \in \mathbb{N}_0$ sei eine temporäre Variable. Daraus ergibt sich:

w	n	f	$(x_i + 1) \cdot 2^{i-1}$	d_e	$s = \sum_{j=1}^{i} 2^{j-1}$	$\frac{d_e - s}{2^{i-1}}$
ε_w	0	()		0		
a	1	(0)	1	1	1	0
b	1	(1)	2	2	1	1
aa	2	(0, 0)	1, 2	3	$1 + 2$	2, 0
ba	2	(1, 0)	2, 2	4	$1 + 2$	3, 0
ab	2	(0, 1)	1, 4	5	$1 + 2$	4, 1
bb	2	(1, 1)	2, 4	6	$1 + 2$	5, 1
aaa	3	(0, 0, 0)	1, 2, 4	7	$1 + 2 + 4$	6, 2, 0

□

Beispiel 1.3.2. Es sei $V = \{a, b, c\}$ und daher $m = 3$, woraus $X = \{0, 1, 2\} \subset \mathbb{N}_0$ folgt. $s \in \mathbb{N}_0$ sei eine temporäre Variable. Daraus ergibt sich:

w	n	f	$(x_i + 1) \cdot 3^{i-1}$	d_e	$s = \sum_{j=1}^{i} 3^{j-1}$	$\frac{d_e - s}{3^{i-1}}$
ε_w	0	()		0		
a	1	(0)	1	1	1	0
b	1	(1)	2	2	1	1
c	1	(2)	3	3	1	2
aa	2	(0, 0)	1, 3	4	$1 + 3$	3, 0
cc	2	(2, 2)	3, 9	12	$1 + 3$	11, 2
aaa	3	(0, 0, 0)	1, 3, 9	13	$1 + 3 + 9$	12, 3, 0

□

1.4 Graphentheorie

In der Literatur zur Graphentheorie sind Definitionen zu finden, die sich in Teilen unterscheiden. Daher werden als nächstes einige grundlegende Definitionen aufgeführt um festzulegen, wie sie hier zu verstehen sind. Sie werden für

die Einordnung der Aufgaben und Prinzipien der kryptographischen Verkettung benötigt. Für die Erstellung dieses Abschnitts wurden die Bücher [Aig93, Jun94, SS89] verwendet, die allgemein in die Graphentheorie einführen.

1.4.1 Graphen

Definition 1.4.1. Ein *Graph* G sei ein Paar (E, K_G), bestehend aus einer (endlichen) Menge $E \neq \emptyset$ von *Ecken* und einer Menge $K_G \subseteq \binom{E}{2} = \{\{u, v\} \mid u, v \in E\}$ von *Kanten* $k = \{u, v\}$ (kurz: $k = uv$). □

Wörtlich kann $\binom{E}{2}$ als Menge aller 2-elementigen Teilmengen der Menge E beschrieben werden. Daraus folgt $u \neq v$. Die Ecken werden zum Beispiel durch einen Punkt oder einen Kreis dargestellt und die Kanten $k = \{u, v\}$ durch eine Linie, die die entsprechenden Punkte oder Kreise von u und v miteinander verbindet.

Definition 1.4.2. Eine Folge von Kanten $(k_1, ..., k_n)$ heißt *Kantenzug*, wenn $v_0, ..., v_n \in E$ existieren mit $k_i = v_{i-1}v_i$ für $i = 1, ..., n$. Ein Kantenzug heißt *Weg*, wenn $k_i \neq k_j$ gilt für $i \neq j,\ i, j = 1, ..., n$. Ein Weg heißt *einfacher Weg*, wenn $v_i \neq v_j$ ist für $i \neq j,\ i, j = 0, ..., n$. □

Anders ausgedrückt sind bei einem Weg die Kanten k_i paarweise verschieden und bei einem einfachen Weg zusätzlich die Ecken v_i.

Definition 1.4.3. Die *Länge* eines Kantenzugs, eines Wegs oder eines einfachen Wegs ist die Anzahl n der Kanten. v_0 heißt *Startpunkt* und v_n heißt *Endpunkt*. □

Definition 1.4.4. Ein *Hamiltonscher Weg* ist ein einfacher Weg, der alle Ecken des Graphen genau einmal enthält. □

Definition 1.4.5. Ein Kantenzug ist *geschlossen*, wenn $v_0 = v_n$ gilt. Ein geschlossener Weg ist ein *Kreis*, und ein *einfacher Kreis* ist ein einfacher Weg mit der Ausnahme $v_0 = v_n$. □

Definition 1.4.6. Eine Ecke $v \in E$ ist von einer Ecke $u \in E$ aus *erreichbar*, wenn es $(v_0v_1, ..., v_{n-1}v_n)$ gibt mit $v_0 = u$, $v_n = v$, $v_0, ..., v_n \in E$ und $v_{i-1}v_i \in K_G$ für $i = 1, ..., n$. □

Das heißt, v ist von u aus erreichbar, wenn es einen Kantenzug mit Startpunkt u und Endpunkt v gibt.

Definition 1.4.7. Ein Graph $G = (E, K_G)$ ist *zusammenhängend*, wenn für alle $u \neq v,\ u, v \in E$ ein Kantenzug $(v_0v_1, ..., v_{n-1}v_n)$ existiert mit $v_0 = u$, $v_n = v$, $v_0, ..., v_n \in E$ und $v_{i-1}v_i \in K_G$ für $i = 1, ..., n$. □

Intuitiv formuliert ist G zusammenhängend, wenn von jeder Ecke u zu jeder anderen Ecke v ein Kantenzug existiert bzw., wenn jede Ecke v von jeder anderen Ecke u aus erreichbar ist.

Definition 1.4.8. Ein zusammenhängender Graph ist ein *Baum*, wenn er keine Kreise enthält. □

1.4.2 Gerichtete Graphen

Definition 1.4.9. Ein *gerichteter Graph* $\vec{G}$ sei ein Paar $(E, K_{\vec{G}})$, bestehend aus einer (endlichen) Menge $E \neq \emptyset$ von *Ecken* und einer Menge $K_{\vec{G}} \subseteq E^2 = \{(u, v) \mid u, v \in E\}$ von Paaren, den *gerichteten Kanten* $k = (u, v)$ (kurz: $k = uv$) mit $u \neq v$. u heißt *Startpunkt* und v heißt *Endpunkt* von k. □

Eine gerichtete Kante $k = (u, v)$ wird durch einen Pfeil dargestellt, der von u nach v zeigt.

Jeder gerichtete Graph $\vec{G} = (E, K_{\vec{G}})$ hat einen *zugrundeliegenden* (ungerichteten) Graphen $G = (E, K_G)$, der in K_G für alle gerichteten Kanten $(u, v) \in K_{\vec{G}}$ und $(v, u) \in K_{\vec{G}}$, zwischen zwei Ecken $u \in E$ und $v \in E$, genau eine Kante $\{u, v\}$ enthält. Für G und $\vec{G}$ ist die Menge der Ecken E gleich. Abbildung 1.1 veranschaulicht dies in einem kleinen Beispiel. Außerdem wird vereinbart, daß der zugrundliegende Graph G gemeint ist, wenn die Definitionen 1.4.2 bis 1.4.8 im Zusammenhang mit einem gerichteten Graphen $\vec{G}$ gebraucht werden.

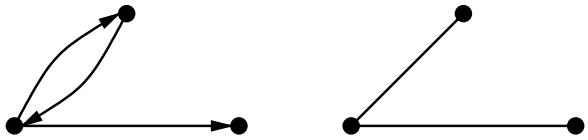

Abbildung 1.1: Gerichteter Graph mit zugrundeliegendem Graphen

Die Definitionen 1.4.2 bis 1.4.6 können allerdings auch auf gerichtete Graphen erweitert werden. Dabei wird eine Kante $\{u, v\}$ durch eine gerichtete Kante (u, v) ersetzt und jeweils entsprechend das Wort „gerichtet“ vorangestellt. Analog zu Definition 1.4.7 ist die nächste Definition zu sehen:

Definition 1.4.10. Ein gerichteter Graph $\vec{G}$ heißt *zusammenhängend*, wenn der zugrundeliegende Graph G zusammenhängend ist. $\vec{G} = (E, K_{\vec{G}})$ ist *stark zusammenhängend*, wenn von jeder Ecke $u \in E$ zu jeder anderen Ecke $v \in E$ ein gerichteter Kantenzug existiert. □

Jeder (ungerichtete) Graph $G = (E, K_G)$ kann in einen gerichteten Graphen $\vec{G} = (E, K_{\vec{G}})$ überführt werden, indem für jede Kante $\{u, v\} \in K_G$ die beiden gerichteten Kanten (u, v) und (v, u) in $K_{\vec{G}}$ eingesetzt werden. Dies wird in Abbildung 1.2 als Beispiel dargestellt. Die Menge der Ecken E ist auch hier wieder für G und $\vec{G}$ gleich.

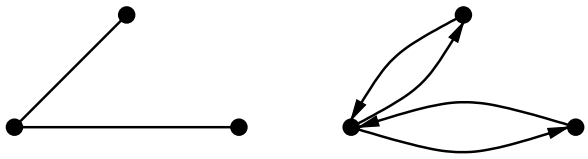

Abbildung 1.2: (Ungerichteter) Graph als gerichteter Graph

Definition 1.4.11. Eine Ecke $r \in E$ heißt *Wurzel*, wenn von r zu jeder anderen Ecke $w \in E \setminus \{r\}$ ein gerichteter Kantenzug $(v_0v_1, ..., v_{n-1}v_n)$ existiert mit $v_0 = r$, $v_n = w$, $v_0, ..., v_n \in E$ und $v_{i-1}v_i \in K_{\vec{G}}$ für $i = 1, ..., n$. □

Die Ecke r ist also eine Wurzel, wenn jede andere Ecke w von r aus über einen gerichteten Kantenzug erreichbar ist.

Definition 1.4.12. $P_v = \{u \mid (u, v) \in K_{\vec{G}}\}$ ist die *Menge der Vorgänger von* v (engl. predecessors), und $S_v = \{w \mid (v, w) \in K_{\vec{G}}\}$ ist die *Menge der Nachfolger von* v (engl. successors). □

Definition 1.4.13. Eine Ecke $s \in E$ heißt *Quelle*, wenn $P_s = \emptyset$ gilt, und eine Ecke $t \in E$ heißt *Senke*, wenn $S_t = \emptyset$ ist. □

Das bedeutet, daß für eine Quelle s in $K_{\vec{G}}$ keine Kante mit dem Endpunkt s existiert und für eine Senke t keine Kante mit dem Startpunkt t. Desweiteren kann eine Quelle eine Wurzel sein, muß sie aber nicht. Eine Senke kann dagegen keine Wurzel sein.

1.5 Codierungstheorie

Die Codierungstheorie wird für die Formalisierungen der Begriffe Sicherheit und Authentizität benötigt, wobei sie allerdings kein Schwerpunktthema dieser Arbeit darstellt. Dieser Abschnitt beschränkt sich daher auf einen sehr kurzen Überblick, der auf [Roh95] beruht.

Zusammengefaßt sind die Hauptaufgaben der Codierungstheorie die *Fehlererkennung* und die *Fehlerkorrektur* in einer Nachricht. In Abgrenzung zur Kryptographie geht es dabei nicht darum, die Fälschung einer Nachricht zu verhindern.

Um in einer Nachricht einen Fehler erkennen oder korrigieren zu können, wird ihr *Redundanz* hinzugefügt. Von Redundanz wird dann gesprochen, wenn eine Nachricht aus einer bestimmten Menge von Buchstaben vorhersagbar ist und die Betrachtung weiterer (redundanter) Buchstaben nichts an der Vorhersage ändert. Ein Fehler kann dann dadurch erkannt werden, daß sich die Vorhersage ändert, obwohl sie das nicht dürfte.

Beim Hinzufügen von Redundanz zu einer Nachricht ist zwischen *Blockcodes* und *Faltungscodes* zu unterscheiden. Dafür wird angenommen, daß eine Nachricht durch Bits aus $\{0, 1\}$ dargestellt wird.

Bei einem Blockcode werden einem n Bit langem Nachrichtenblock k Kontrollstellen hinzugefügt, so daß ein Codewort mit der Länge $N = n+k$ Binärstellen entsteht.

Beispiel 1.5.1. Bei einer Paritätskontrolle wird den n Nachrichtenbits jeweils ein Kontrollbit hinzugefügt, so daß $k = 1$ und $N = n+1$ ist. Danach können die Kontrollbits x_{n+1} durch eine Modulo-2-Addition der Nachrichtenbits $x_1, ..., x_n$ berechnet werden, so daß $x_1 \oplus x_2 \oplus x_3 \oplus ... \oplus x_n = x_{n+1}$ ist. Diese Bedingung muß dann auch bei der Überprüfung des Codewortes gelten. Dabei kann allerdings nur eine ungerade Anzahl von Fehlern erkannt werden, da die Bedingung bei einer geraden Anzahl von Fehlern trotzdem erfüllt wird. □

Im Gegensatz zu Blockcodes wird bei Faltungscodes eine Nachricht nicht in einzelne Blöcke zerlegt, sondern kontinuierlich als Bitfolge verarbeitet, worauf hier aber nicht weiter eingegangen wird.

1.6 Kryptographie

Neben der Codierungstheorie (s. Abschnitt 1.5) ist die Kryptographie für die Formalisierungen der Begriffe Sicherheit und Authentizität von zentraler Bedeutung. Daher wird hier die Kryptographie überblicksartig beschrieben. Für umfassendere Einführungen in die Kryptographie können beispielsweise die Bücher [Bau93, MOV97, Schn96] oder das Vorlesungsskript [Wät99b] herangezogen werden.

Zunächst einmal dient der Begriff *Kryptologie*, wie in Abbildung 1.3 veranschaulicht, dem Zusammenfassen der Bereiche *Kryptographie*, *Kryptanalyse* und *Steganographie* unter einem Dach. Verkürzt gesagt ist die Kryptographie die Wissenschaft vom geheimen Schreiben, die Kryptanalyse die Wissenschaft von der unberufenen Entzifferung und die Steganographie die Wissenschaft vom verdeckten Schreiben. Der Unterschied zwischen Kryptographie und Steganographie besteht darin, daß die Kryptographie Informationen sichern soll, die im gesicherten Zustand für einen Angreifer zugänglich sind, während die Steganographie Informationen gegenüber einem Angreifer verbergen soll, so daß er nicht bemerkt, daß überhaupt Informationen existieren. Eine Kombination beider Bereiche kann zum Beispiel dadurch erfolgen, daß kryptographisch gesicherte Informationen mittels der Steganographie versteckt werden.

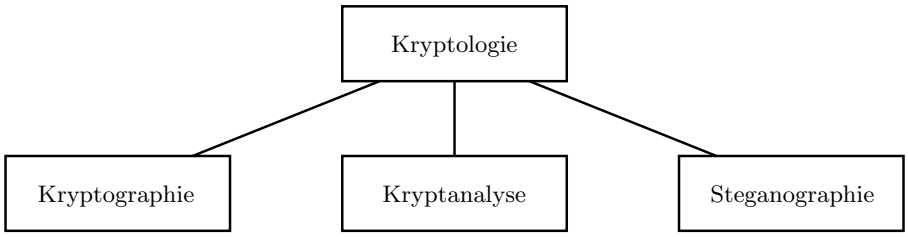

Abbildung 1.3: Bereiche der Kryptologie

In Abbildung 1.4 wird die Kryptographie in vier Bereiche unterteilt, die auf verschiedenste Arten voneinander abhängen. In den nachfolgenden Unterabschnitten werden diese Bereiche zusammenfassend beschrieben. Natürlich sind auch andere Einteilungen möglich, wobei die Motivation für diese Einteilung darin besteht, Protokolle von Verkettungen zu unterscheiden und Verkettungen als eigenständigen Teilbereich zu betrachten.

1.6.1 Grundaufgaben

Die Formalisierung der Grundaufgaben der Kryptographie basiert auf folgender grundlegender Definition:

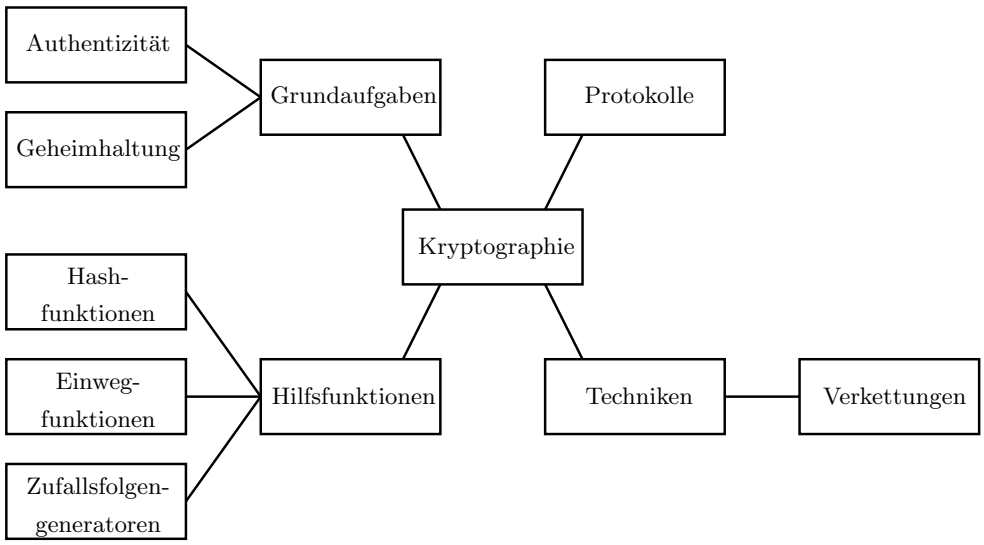

Abbildung 1.4: Bereiche der Kryptographie

Definition 1.6.1. $(\mathcal{M}, \mathcal{C}, \mathcal{K}, E_{K_1}, D_{K_2})$ sei ein *kryptographisches System* (kurz: *Kryptosystem*) mit

- einer *Menge von Klartexten* $\mathcal{M}$,
- einer *Menge von Chiffretexten* $\mathcal{C}$,
- einer *Menge von Schlüsseln* $\mathcal{K}$,
- einer Familie von *Chiffriertransformationen* $E_{K_1} : \mathcal{M} \longrightarrow \mathcal{C}$ mit $K_1 \in \mathcal{K}$ und
- einer Familie von *Dechiffriertransformationen* $D_{K_2} : \mathcal{C} \longrightarrow \mathcal{M}$ mit $K_2 \in \mathcal{K}$. □

Jede Chiffriertransformation E_{K_1} besteht aus einem Chiffrieralgorithmus E und einem Schlüssel K_1. Analog dazu besteht jede Dechiffriertransformation D_{K_2} aus einem Dechiffrieralgorithmus D und einem Schlüssel K_2. Dabei sind E und D für jede Transformation der jeweiligen Familie gleich. Mit E_{K_1} kann ein Klartext $M \in \mathcal{M}$ in einen Chiffretext $C \in \mathcal{C}$ überführt bzw. *verschlüsselt* werden:

$$E_{K_1}(M) = C.$$

Ein Chiffretext C kann dann wieder mit D_{K_2} in einen Klartext M überführt bzw. *entschlüsselt* werden:

$$D_{K_2}(C) = M.$$

Zusammengefaßt gilt für alle $M \in \mathcal{M}$:

$$D_{K_2}(E_{K_1}(M)) = M.$$

An ein Kryptosystem werden mindestens folgende Anforderungen gestellt (s. [Wät99b]):

1. Zur Begrenzung des Zeit- und Speicherplatzbedarfs müssen E_{K_1} und D_{K_2} für alle $K_1, K_2 \in \mathcal{K}$ effizient berechnet werden können.

2. Es muß leicht sein, $K_1, K_2 \in \mathcal{K}$ sowie die zugehörigen E_{K_1} und D_{K_2} zu finden.

3. Um die Sicherheit eines Kryptosystems untersuchen zu können, sollte diese nicht auf der Geheimhaltung von E und/oder D beruhen, sondern auf der Geheimhaltung des jeweils verwendeten K_1 und/oder K_2. Das heißt, es sollte berechnungsmäßig praktisch nicht möglich sein, aus C und der Kenntnis von E und/oder D wieder M zu bestimmen.

Auf dieser Grundlage kann die Menge aller Kryptosysteme in zwei Klassen aufgeteilt werden, die der *symmetrischen Kryptosysteme* und die der *asymmetrischen Kryptosysteme*. Ein Kryptosystem gehört zur Klasse der symmetrischen Kryptosysteme, wenn die Schlüssel K_1 der Chiffriertransformation E_{K_1} und K_2 der Dechiffriertransformation D_{K_2} gleich sind ($K_1 = K_2$) oder sich berechnungsmäßig einfach auseinander bestimmen lassen. Man kann dann auf die Indizes 1 und 2 verzichten. Gilt dagegen $K_1 \neq K_2$ oder genauer gesagt, daß sie sich berechnungsmäßig praktisch nicht auseinander ableiten lassen, so gehört das Kryptosystem zur Klasse der asymmetrischen Kryptosysteme, die auch *Public-Key-Kryptosysteme* genannt werden. „Public-Key“ deshalb, weil entweder E_{K_1} oder D_{K_2} öffentlich bekannt sein darf, ohne daß dies negative Auswirkungen auf die jeweils andere Transformation hat.

Die Grundaufgaben der Kryptographie bestehen nun darin, die *Authentizität* oder die *Geheimhaltung* von Informationen zu sichern (s. a. Kapitel 2). Diese Aufgaben werden über folgende Anforderungen an die Kryptosysteme definiert (s. [Wät99b]):

- *Authentizitätsanforderungen*:
  Es sollte berechnungsmäßig praktisch unmöglich sein,

  1. systematisch E_{K_1} aus einem Chiffretext C zu bestimmen, selbst dann, wenn der Klartext M mit $E_{K_1}(M) = C$ bekannt ist, oder
  2. unerkannt einen Chiffretext C' für C einsetzen zu können, das heißt, einen Chiffretext C' zu finden, so daß $D_{K_2}(C')$ ein gültiger Klartext aus $\mathcal{M}$ ist.

- *Geheimhaltungsanforderungen*:
  Es sollte berechnungsmäßig praktisch unmöglich sein,

  1. systematisch D_{K_2} aus einem Chiffretext C zu bestimmen, selbst dann, wenn der Klartext M mit $E_{K_1}(M) = C$ bekannt ist, und

2. den Klartext M aus C zu bestimmen.

Aus den Authentizitätsanforderungen läßt sich ableiten, daß D_{K_2} öffentlich bekannt sein darf, solange D_{K_2} nicht E_{K_1} verrät, und aus den Geheimhaltungsanforderungen geht hervor, daß E_{K_1} für jederman zugänglich sein darf, wenn sich aus E_{K_1} nicht D_{K_2} ableiten läßt. Daraus folgt, daß mittels Public-Key-Kryptosystemen die Sicherung der Authentizität und der Geheimhaltung von Informationen voneinander getrennt werden kann, während dies bei symmetrischen Kryptosystemen nicht möglich ist.

1.6.2 Hilfsfunktionen

Einen weiteren Bereich stellen die kryptographischen Hilfsfunktionen dar. Dies sind Funktionen, die auch außerhalb der Kryptologie Verwendung finden, an die aber in der Kryptographie teilweise besondere Anforderungen gestellt werden. In der Kryptographie stellen diese für sich allein genommen nicht unbedingt einen besonderen Wert dar, sondern entfalten diesen erst unter Beachtung der Grundaufgaben, wodurch beispielsweise kryptographische Protokolle oder Verkettungen entstehen. Die drei häufigsten/wichtigsten Arten der kryptographischen Hilfsfunktionen sind die Hashfunktionen, die Einwegfunktionen und die Zufallsfolgengeneratoren.

Definition 1.6.2. Eine *Hashfunktion* ist eine surjektive Funktion $h: X^* \longrightarrow X^m$ mit $m \in \mathbb{N}$ und $X^m \subseteq X^*$. □

Anders ausgedrückt bildet h eine beliebig lange Folge von Objekten $x = x_1...x_n$ auf eine Folge von Objekten $y = x_1...x_m$ fester Länge m ab. y wird auch *Hashwert* oder *Fingerabdruck* genannt.

Für gegebene x und h wird zumeist gefordert, daß $y = h(x)$ effizient berechnet werden kann. Dadurch soll zum Beispiel erreicht werden, daß die Berechnung und Weiterverarbeitung von y sehr viel einfacher und schneller ist als wenn durchgehend mit x gearbeitet würde. Da h surjektiv ist, muß dabei beachtet werden, daß der Fingerabdruck y nur mit einer gewissen Wahrscheinlichkeit für ein Original x steht. Diese Wahrscheinlichkeit ist abhängig von h und von der Länge von y. So können theoretisch unendlich viele und unendlich lange Folgen x auf genau eine Folge y abgebildet werden. Für praktische Überlegungen ist allerdings meistens die maximal mögliche Länge von x limitiert, wodurch gleichzeitig x nur aus einer endlichen Teilmenge der unendlich großen Menge X^* stammen kann.

Üblicherweise wird in der Kryptographie für X die Menge der Bits $\{0, 1\}$ verwendet, wobei dann x für einen Klartext aus $\mathcal{M}$ steht. Außerdem werden an h gewisse Mindestanforderungen gestellt, die sich aus den folgenden drei Definitionen (s. [Wät99b]) ergeben:

Definition 1.6.3. h heißt *schwach kollisionsfrei* für x, wenn es berechnungsmäßig praktisch unmöglich ist, $x' \neq x$ mit $h(x) = h(x')$ zu finden. □

Definition 1.6.4. h heißt *stark kollisionsfrei*, wenn es berechnungsmäßig praktisch unmöglich ist, x und x' mit $x' \neq x$ und $h(x') = h(x)$ zu finden. □

Definition 1.6.5. Eine Hashfunktion h_e heißt *Einweg-Hashfunktion*, wenn es für einen gegebenen Fingerabdruck y berechnungsmäßig praktisch unmöglich ist, ein Original x mit $h_e(x) = y$ zu finden. □

Die Einweg-Hashfunktionen bilden den Übergang zu den Einwegfunktionen. Allgemein ist unter einer Einwegfunktion folgendes zu verstehen:

Definition 1.6.6. Eine injektive Funktion $f_e : X \longrightarrow Y$ heißt *Einwegfunktion* (*one-way function*), wenn

1. $f_e(x)$ für alle $x \in X$ effizient berechnet werden kann und
2. es berechnungsmäßig praktisch unmöglich ist, für alle $y \in \mathrm{Im}(f_e) \subseteq Y$ die $x \in X$ mit $f_e(x) = y$ zu finden. □

Der Unterschied zu den Einweg-Hashfunktionen besteht vor allem darin, daß Einwegfunktionen injektiv sind und nicht surjektiv. Desweiteren läßt sich die Definition der Einwegfunktionen folgendermaßen erweitern:

Definition 1.6.7. Eine Einwegfunktion $f_h : X \longrightarrow Y$ heißt *Einwegfunktion mit Hintertür* (*trapdoor one-way function*), wenn $f_h^{-1} : \mathrm{Im}(f_h) \longrightarrow X$ mittels einer geheim zu haltenden Zusatzinformation (*Hintertür*, *trapdoor information*) effizient berechnet werden kann. □

Das heißt, daß es mittels der Zusatzinformation berechnungsmäßig einfach wird, für alle $y \in \mathrm{Im}(f_h) \subseteq Y$ die $x \in X$ mit $f_h(x) = y$ zu finden.

Die Definitionen für Einwegfunktionen und solche mit Hintertüren stellen Ideale dar, deren Existenz in der Realität nicht ohne weiteres bewiesen werden kann. Genauso verhält es sich mit den abschließenden Zufallsfolgengeneratoren. Ausführlich diskutiert wird dieses Problem beispielsweise in [MOV97] und in [Schn96].

Definition 1.6.8. Ein *Zufallsfolgengenerator* $RND : \mathbb{N} \longrightarrow X^m$ liefert nicht reproduzierbar für ein $m \in \mathbb{N}$ eine zufällige Folge von Objekten $x = x_1...x_m$ mit $x_i \in X,\ i = 1, ..., m$. □

Die Formulierung „nicht reproduzierbar“ bedeutet, daß es auch für die wiederholte Eingabe von m völlig ungewiß ist, welche Folge von Objekten $x \in X^m$ zurückgeliefert wird.

1.6.3 Techniken und Protokolle

Die Unterscheidung der Begriffe in kryptographische Techniken und kryptographische Protokolle orientiert sich an [Schn96].

Der Bereich der kryptographischen Techniken kann dabei als Sammellager für all jene Techniken angesehen werden, die zur Realisierung der Grundaufgaben der Kryptographie benötigt werden oder aus denen sich bestimmte Protokolle ableiten lassen. In diesem Zusammenhang bilden die kryptographischen Verkettungen einen eigenständigen Teilbereich der kryptographischen Techniken. Obwohl kryptographische Verkettungen in [Schn96] nicht als allgemeine

graphentheoretische Probleme aufgefaßt werden, stellen die im dortigen Kapitel 9 vorgestellten Blockchiffriermodi bestimmte kryptographische Verkettungen dar. Diese werden aber im Rahmen dieser Arbeit nicht weiter analysiert.

Nach [Schn96] dient ein *Protokoll* der Durchführung einer bestimmten Aufgabe und besteht aus einer Folge von Aktionen, an denen zwei oder mehr Parteien beteiligt sind. Im Vergleich dazu ist nach [Schn96] ein *kryptographisches Protokoll* ein Protokoll, das kryptographische Mittel einsetzt. In [Schn96] und [Wät99b] werden eine Vielzahl von kryptographischen Protokollen vorgestellt, die verschiedensten Zwecken dienlich sind.

Eine Unterscheidung der Begriffe in kryptographische Verkettungen und kryptographische Protokolle ist erforderlich, weil nicht alle, sondern nur bestimmte kryptographische Protokolle, wie beispielsweise für Zeitstempel und für Log-Dateien, von kryptographischen Verkettungen Gebrauch machen. Desweiteren können die Eigenschaften von kryptographischen Verkettungen untersucht werden, ohne daß sie ein Protokoll darstellen müssen.

Kapitel 2

Sicherheit

Der Begriff *Sicherheit* hat in verschiedenen Zusammenhängen verschiedene Bedeutungen. Daher soll in diesem Kapitel die Frage geklärt werden:

Was bedeutet Sicherheit im Bereich der Informationen?

Dazu wird zunächst in Abschnitt 2.1 auf die allgemeine Problematik bei der Formalisierung des Begriffs Sicherheit eingegangen, wobei sich herausstellen wird, daß der Begriff Sicherheit eng mit dem Begriff *Gefahr* zusammenhängt. Danach wird in Abschnitt 2.2 aus den resultierenden Schlußfolgerungen ein System einer Innen- und Außenwelt erstellt. Mit Hilfe dieses Systems können verschiedene Maßnahmen zur Wahrung der Authentizität der Einträge von Log-Dateien in einen größeren Zusammenhang eingeordnet werden.

2.1 Problematik

Der für diese Arbeit relevante Bereich des Begriffs Sicherheit ergibt sich aus der in der Einleitung gestellten Frage (s. S. 1) nach der Authentizität der Einträge einer Log-Datei. Diese Einträge stellen gewisse Informationen dar, die, wie in Abbildung 2.1 dargestellt, Gefahren unterliegen und daher Sicherheit benötigen. Je größer dabei die Gefahren sind, desto mehr Aufwand muß für die Sicherheit betrieben werden und umgekehrt. Daraus folgt, daß die Frage nach der Sicherheit von Informationen (Abb. 2.3) nicht für sich allein behandelt werden kann, sondern gleichzeitig unter dem Aspekt der tatsächlichen Gefahren für die Informationen (Abb. 2.2) betrachtet werden muß. Ansonsten könnten die Maßnahmen für die Sicherheit nicht alle Gefahren abdecken, was Unsicherheit bedeuten würde, oder es könnten mehr Sicherungsmaßnahmen ergriffen werden als erforderlich, was zwar Sicherheit bedeuten würde, aber mit unnötigen Kosten verbunden wäre. Außerdem ist zu beachten, daß sich die Gefahren sowohl auf den Informationsträger als auch nur auf die eigentlichen Informationen auf dem Träger beziehen können.

Gefahren für Informationen Abbildung 2.2 stellt die verschiedenen Gesichtspunkte der Gefahren für Informationen dar. Die erste Zeile dieser Grafik ist so zu verstehen, daß die Gefahr eine Wahrscheinlichkeit $p(Ereignis)$ für das Eintreten eines Ereignis ist, das zu einem Schaden führt. Ein Ereignis kann

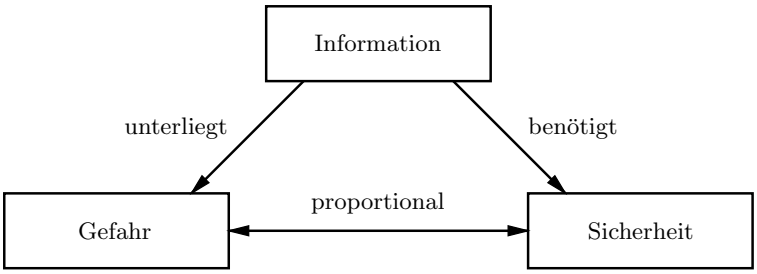

Abbildung 2.1: Zusammenhänge zwischen Information, Gefahr und Sicherheit

aber ohne Eintritts-Wahrscheinlichkeit (die Gefahr) nicht eintreten und folglich zu keinem Schaden führen. Ebensowenig kann eine Eintritts-Wahrscheinlichkeit ohne ein Ereignis schaden. Daher werden hier die Begriffe Gefahr und Ereignis nur noch zusammenfassend unter dem Oberbegriff „Gefahr“ verwendet (s. [Schr96]). Ein möglicher *Schaden* an Informationen bezieht sich im wesentlichen auf deren Verfügbarkeit, Authentizität oder Geheimhaltung. Er kann zum Beispiel dadurch entstehen, das Informationen vernichtet, verzögert, wiederholt, verfälscht, verleugnet oder bekannt gemacht werden.

Die zweite Zeile stellt die Unterscheidung in zufällige und bewußt herbeigeführte Gefahren, den Angriffen, dar. *Zufällige Gefahren* für Informationen sind zum Beispiel (s. a. [Schr96]):

- Höhere Gewalt: Feuer, Wasser, Krieg, ...
- Menschliches Versagen: Unwissenheit, Fahrlässigkeit, ...
- Technisches Versagen: Abnutzung, Fehlfunktion, ...

Sie können zu jeder Art von Schaden an Informationen führen.

Ein *Angriff* auf Informationen muß, wie in Zeile drei, in aktiv und passiv unterschieden werden. Bei einem aktiven Angriff geht es vor allem darum, Informationen zu vernichten, zu verzögern, zu wiederholen, zu fälschen oder zu leugnen, was auf die Verfügbarkeit und Authentizität der Informationen zielt. Unter einem passiven Angriff ist dagegen das (unerwünschte) Erlangen von Informationen zu verstehen, was gegen die Geheimhaltung der Informationen gerichtet ist.

Der wesentliche Unterschied zwischen zufälligen Gefahren und Angriffen besteht in der Motivation des potentiellen Verursachers. Diese ist bei einer zufälligen Gefahr völlig ungerichtet bzw. nicht vorhanden, während sie bei einem Angriff darauf abzielt, für den Angreifer einen Vorteil zu erlangen bzw. dem Geschädigten einen Nachteil zuzufügen. Desweiteren kann ein Angriff durch eine zufällige Gefahr begünstigt werden, was umgekehrt nicht unbedingt gelten muß. So kann zum Beispiel eine Naturkatastrophe einem Angreifer den Zugang zu Informationen ermöglichen, die eigentlich geheim bleiben sollten. Der Umkehrschluß ist hier aber nicht möglich, da der Zugang zu Informationen keine Naturkatastrophe auslösen kann, sondern höchstens eine durch Menschenhand

erzeugte Katastrophe. Für weitere Details kann die Zuordnung der Gefahren zum Begriff Sicherheit bei der nachfolgenden Erläuterung von Abbildung 2.3 herangezogen werden.

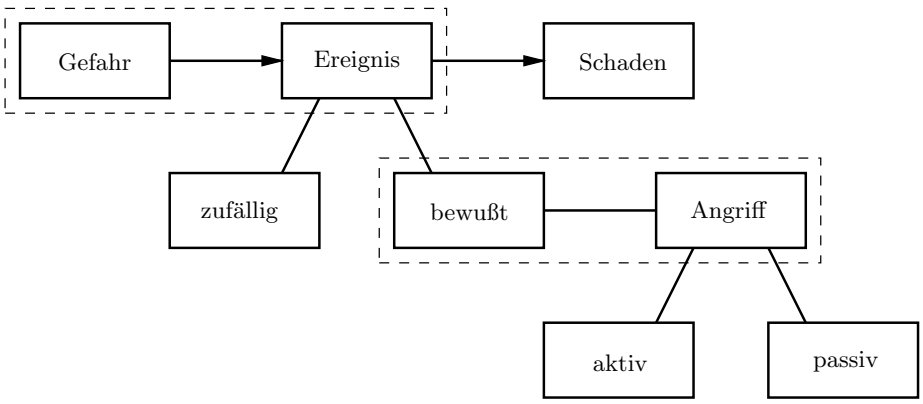

Abbildung 2.2: Gefahren für Informationen [Schr96]

Anforderungen an die Sicherheit von Informationen Abbildung 2.3 stellt die Anforderungen an die Sicherheit von Informationen dar. Informationen müssen demnach verfügbar, prüfbar und auswertbar sein. Die *Verfügbarkeit* ist dabei Voraussetzung für die Prüfbarkeit und nur geprüfte Informationen sollten ausgewertet werden. Zu beachten ist, daß die Verfügbarkeit durch jede Art der zufälligen Gefahren und durch aktive Angriffe, die auf die Vernichtung oder zumindest die Verzögerung von Informationen zielen, beeinträchtigt werden kann.

Die *Prüfbarkeit* ist so zu verstehen, daß nicht die Bedeutung oder der Wahrheitsgehalt der Informationen überprüft wird, sondern daß überprüft wird, ob die ursprünglichen Informationen vorliegen und welchen Ursprung sie haben. Die ursprünglichen Informationen können zum Beispiel durch einen zufälligen Fehler verlorengehen oder dadurch, daß sie bewußt verfälscht werden. Der Unterschied zu vollständig gefälschten Informationen besteht darin, daß diese nicht wirklich falsch sind, sondern einen anderen Ursprung haben als angenommen. In diesem Sinne kann auch angenommen werden, daß bei einem Fehler oder einer Verfälschung die alten Informationen vernichtet werden und an dieser Stelle der Ursprung von neuen Informationen liegt, die mit den alten mehr oder weniger übereinstimmen.

Die *Auswertbarkeit* bezieht sich vor allem auf passive Angriffe und ist umgekehrt proportional zur Wichtigkeit der Informationen. Das heißt, daß unwichtige Informationen für alle auswertbar (nicht geheim) sein können und wichtige Informationen für Unbefugte nicht auswertbar (geheim) sein müssen. Bei der Definition der Wichtigkeit einer Information kann es durchaus vorkommen, daß eine Information für eine Gruppe unwichtig ist, sie aber indirekt dadurch wichtig wird, daß eine andere Gruppe keine Kenntnis von ihr erlangen darf. Diese zunächst unwichtige Information ist daher letztendlich eine wichtige Information.

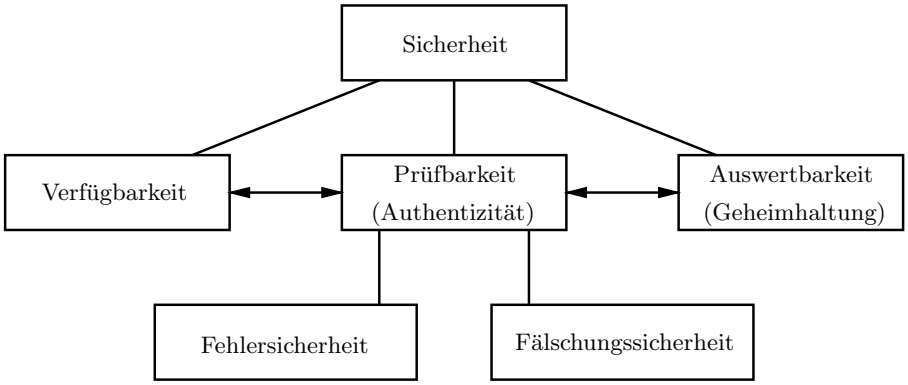

Abbildung 2.3: Anforderungen an die Sicherheit von Informationen

Zusammenhänge Vergleichbar zur Gefahr kann man sagen, daß die Sicherheit eine Wahrscheinlichkeit $p(\neg Ereignis)$ für das Nichteintreten eines Ereignis ist, das zu einem Schaden führt. Die Wahrscheinlichkeiten für Gefahr und Sicherheit stehen dabei mittels $p(Ereignis) + p(\neg Ereignis) = 1$ in einem direktem Zusammenhang, denn ein Ereignis kann nur entweder eintreten oder nicht eintreten. So liegt das eigentliche Problem darin begründet, daß man sich letztendlich[1] nie wirklich sicher sein kann, alle Ereignisse zu kennen, die zu einem Schaden führen können. Die Abwägung der Maßnahmen für die Sicherheit auf Seite 18 muß daher dahingehend konkretisiert werden, daß die Maßnahmen immer nur gegen bekannte Gefahren gerichtet sind. Eine Aussage darüber, ob und zu welchem Grad die Maßnahmen gegen unbekannte Gefahren schützen, ist nicht möglich.

Abbildung 2.4 ordnet noch einmal überblicksartig die verschiedenen Arten von Gefahren den verschiedenen Anforderungen an die Sicherheit zu. Zu beachten ist hierbei, daß verschiedene Arten von Gefahren, zwar auf die gleiche Anforderung an die Sicherheit zeigen, aber trotzdem getrennt gelöst werden können und müssen. So steht für die Prüfbarkeit bei zufälligen Gefahren die Fehlersicherheit im Vordergrund und bei aktiven Angriffen die Fälschungssicherheit.

Weitergehende Einführungen in die allgemeine Problematik der Sicherheit von Informationen sind beispielsweise in [BSI, Gri94, Pom91, Schr96] zu finden.

2.2 Lösungsansätze

In Abschnitt 2.1 wurde beschrieben, wie verschiedene Arten von Gefahren für Informationen auf verschiedene Anforderungen an die Sicherheit von Informationen zeigen. Allgemeines Ziel des folgenden Abschnitts ist es Maßnahmen aufzuzeigen, die die *Eintritts-Wahrscheinlichkeiten der Gefahren reduzieren* und somit die *Wahrscheinlichkeiten für die Sicherheit erhöhen*. Das heißt, daß durch

[1] Man kann sich nie wirklich sicher sein, daß die Welt so ist, wie man sie wahrnimmt (s. z. B. [Rus12, Vol88]).

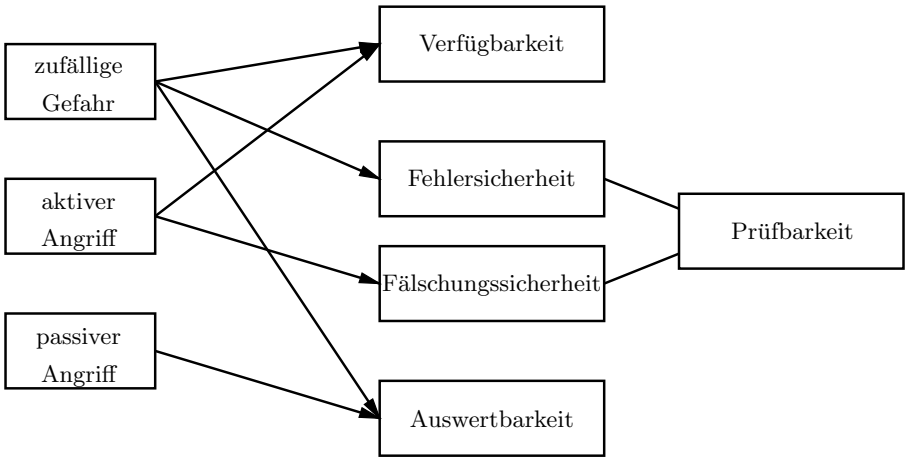

Abbildung 2.4: Zuordnung der Gefahren zur Sicherheit

die Maßnahmen $p(Ereignis)$ gegen 0 und $p(\neg Ereignis)$ gegen 1 gehen soll. Ein weiteres Ziel ist die genauere Einordnung der kryptographischen Verkettung, bevor diese in Kapitel 3 dargelegt wird.

Maßnahmen Bestimmte Informationen unterliegen in der Regel mehreren voneinander unabhängigen Gefahren gleichzeitig. Daher müssen für diese Informationen auch mehrere Maßnahmen gleichzeitig ergriffen werden, um die Wahrscheinlichkeiten für die Sicherheit zu erhöhen. Die Menge dieser Maßnahmen stellt ein sogenanntes *Sicherheitssystem* dar, daß die Welt in eine „sichere“ Innenwelt und eine „unsichere“ Außenwelt aufspaltet. Die einzelnen Maßnahmen dieses Sicherheitssystems können wiederum in drei Teilmengen aufgeteilt werden, die der *physischen, organisatorischen und logischen Sicherheit der Informationen* dienen (s. a. [Pom91, Schr96]). Abbildung 2.5 soll diese Zusammenhänge verdeutlichen. Weiter dienen *passive Maßnahmen* dem Ausgrenzen von Gefahren und *aktive Maßnahmen* dem Eingrenzen von Gefahren.

Die physische Sicherheit von Informationen bezieht sich in erster Linie auf den Informationsträger, zum Beispiel die Festplatte in einem Computer oder ein Stück Papier, und daher nur indirekt auf die eigentlichen Informationen. Bauliche Maßnahmen, wie beispielsweise ein Bunker oder ein Tresor, sollen dabei passiv mögliche Gefahren ausgrenzen, wogegen zum Beispiel eine Feuerlöscheinrichtung aktiv mögliche Gefahren eingrenzen soll.

Die organisatorische Sicherheit ist vor allem auf den Menschen und seinen Umgang mit den Informationen gerichtet. Zum einen wird durch Benutzerberechtigungen die Menschheit in vertrauenswürdige und nicht vertrauenswürdige Menschen gespalten, die mit bestimmten Informationen umgehen dürfen bzw. nicht umgehen dürfen. Zum anderen soll die Verwendung möglichst sicherer Arbeitsabläufe Gefahren ausgrenzen bzw. sollen Notfallpläne Gefahren eingrenzen.

Die logische Sicherheit bezieht sich auf die Informationen selbst, wogegen die physische und die organisatorische Sicherheit nur indirekt auf die Informationen gerichtet sind. Die Maßnahmen für die logische Sicherheit sollen direkt die in Abschnitt 2.1 beschriebenen Anforderungen an die Sicherheit von Infor-

mationen erfüllen. Die Verfügbarkeit kann dabei durch redundante Informationen (Backups), die sich physisch an möglichst verschiedenen Orten befinden müssen, erhöht werden. Bei der Prüfbarkeit können zufällige Fehler dadurch erkennbar werden, daß die Informationen in einer bestimmten Art und Weise codiert werden (s. Codierungstheorie). Bewußte Fälschungen können durch das Signieren der Informationen erschwert werden (s. Kryptologie). Die Auswertbarkeit kann für Unbefugte dadurch behindert werden, daß die Informationen auf eine geheime Art und Weise verschlüsselt werden (s. Kryptologie).

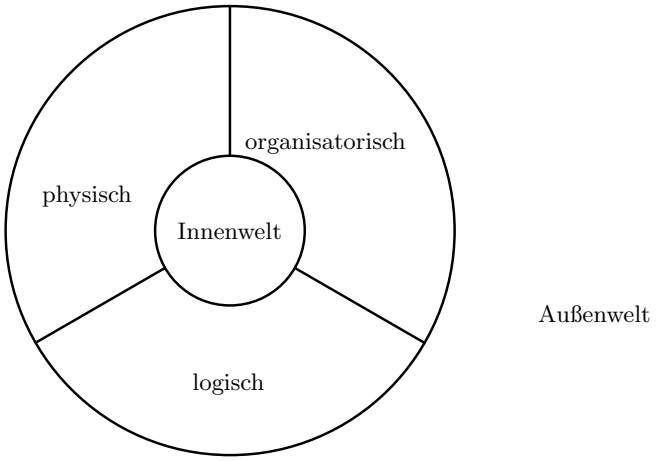

Abbildung 2.5: Sicherheitssystem für Informationen (s. a. [Schr96])

Zusammenhänge Die Prüfbarkeit und die Auswertbarkeit hängen bei möglichen Maßnahmen in mehreren Stufen miteinander zusammen. So kann das Verschlüsseln der Informationen auch dazu dienen, ihre Fälschung zu erschweren, und das Signieren der Informationen kann gleichfalls dazu genutzt werden, Fehler zu erkennen. Allerdings könnte bei letzterem nicht entschieden werden, ob es sich nur um einen Fehler oder um eine Fälschung handelt.

Maßnahmenhierarchie Die Zusammenhänge zwischen den auf Seite 20 beschriebenen Anforderungen an die Sicherheit von Informationen und den Maßnahmen für die Erfüllung der Anforderungen lassen eine gewisse hierarchische Ordnung erkennen. Daher sollten die Maßnahmen für die logische Sicherheit in einem Sicherheitssystem ebenfalls hierarchisch geordnet werden, wobei die Maßnahmen der nächsten Stufe alle Maßnahmen der vorherigen Stufen möglichst sinnvoll einkapseln.

Ohne die Details zu vertiefen, stellt Abbildung 2.6 einen Vorschlag für die hierarchische Anordnung von Maßnahmen dar, die der logischen Sicherheit von Informationen dienen. Dabei wurde neben den Maßnahmen für die Sicherheit auch die Komprimierung von Informationen aufgenommen, weil sie sehr häufig angewendet wird und ihr Einsparpotential maßgeblich von ihrer Positionierung innerhalb der Hierarchie abhängig ist (s. dazu [Schn96]). Weiterhin enthalten die ersten vier Stufen mit der Codierung jeweils eine Maßnahme zur Fehlererkennung. Dadurch kann entschieden werden, ob die Umkehrung der Maßnahme

der nächsten Stufe ohne Fehler funktioniert hat oder nicht. Die Codierung selbst muß dabei so gewählt werden, daß sie auf die Maßnahme der nächsten Stufe möglichst wenig negative Auswirkungen hat.

In Beispiel 1.5.1 stellt die Modulo-2-Addition der Nachrichtenbits eine Hashfunktion dar. Allgemein kann daher zur Codierung der Informationen $x = x_1...x_n$ zum Beispiel mittels einer Hashfunktion h der Hashwert $y = h(x)$ berechnet werden, der zusammen mit den Informationen komprimiert wird. Nach einer Dekomprimierung von x und y müßte dann ein neuerlich berechneter Hashwert $y' = h(x)$ mit y übereinstimmen, um mit einer gewissen Wahrscheinlichkeit feststellen zu können, daß die ursprünglichen Informationen wiederhergestellt wurden. Entsprechendes gilt für die übrigen Stufen der Maßnahmenhierarchie. Die Positionen der Signierung und der Verschlüsselung können eventuell auch miteinander vertauscht werden, während letztendlich all diese Maßnahmen vor einer Vervielfältigung stattgefunden haben sollten, damit sie nicht für jede einzelne Kopie wiederholt werden müssen.

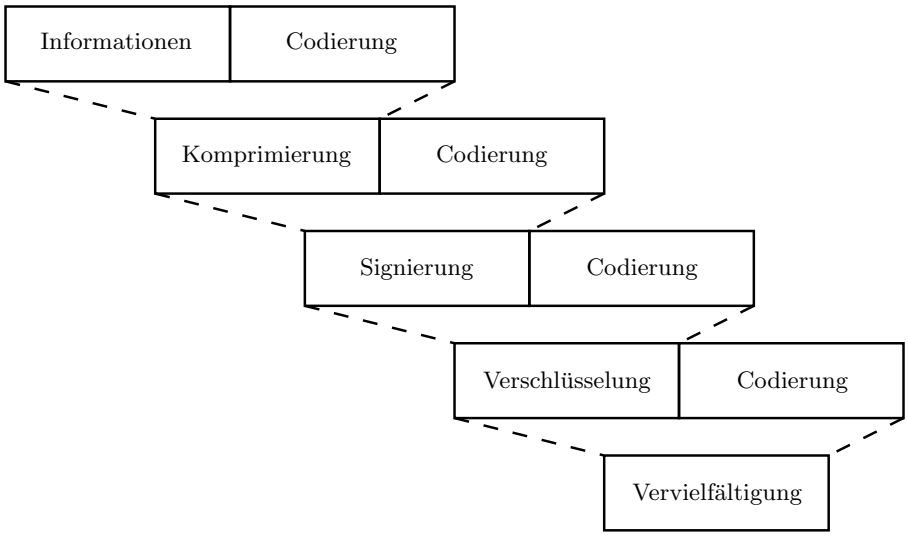

Abbildung 2.6: Maßnahmen für die logische Sicherheit von Informationen

Kryptographische Verkettung Die kryptographische Verkettung dient der logischen Sicherheit von Informationen und soll als Maßnahme zur Sicherung der Authentizität in erster Linie bewußte Fälschungen verhindern. Unter bestimmten Voraussetzungen kann dadurch gleichzeitig die Auswertbarkeit der Informationen eingeschränkt werden.

Kapitel 3

Kryptographische Verkettung

In diesem Kapitel beschreibe ich die grundlegenden Prinzipien von kryptographischen Verkettungen. Anders als beispielsweise in den Arbeiten von Haber und Stornetta [HS91, BHS93, HS97] oder den Arbeiten von Schneier und Kelsey [KS96, KS99, SK97a, SK97b, SK98, SK99], in denen bestimmte kryptographische Verkettungen zur Realisierung konkreter Ziele (Zeitstempel, Log-Dateien) eingesetzt wurden, fasse ich hier erstmals die kryptographische Verkettung allgemeiner als ein graphentheoretisches Problem auf.

Dazu legt der erste Abschnitt 3.1 die eigentliche Motivation und die in der Einleitung bereits angeschnittenen Ziele der kryptographischen Verkettung eines Graphen dar und listet die wichtigsten Nebenbedingungen auf. Der darauf folgende Abschnitt 3.2 stellt eine spezielle Repräsentationsform für Graphen vor. Basierend auf dieser Repräsentation wird in Abschnitt 3.3 erläutert, wie die Ziele der kryptographischen Verkettung prinzipiell erreicht werden können, während abschließend in Abschnitt 3.4 die Nebenbedingungen thematisiert werden. Dadurch bildet dieses Kapitel die theoretische Grundlage für das nächste Kapitel 4, in dem verschiedene praktische Lösungen von kryptographischen Verkettungen für sichere Log-Dateien vorgestellt werden.

3.1 Motivation und Ziele

Ein Graph stellt die Beziehungen zwischen ansonsten eigenständigen Objekten dar. Die Ecken des Graphen stehen dabei für die Objekte und die Kanten des Graphen für die Beziehungen. Die Objekte und Beziehungen sollen nun mittels kryptographischer Verfahren gesichert werden, wobei die Objekte ihre Eigenständigkeit behalten sollen.

Log-Datei Wie in der Einleitung auf Seite 2 beschrieben, kann beispielsweise eine Log-Datei $\mathcal{L}$ durch einen gerichteten Graphen $\vec{G} = (E, K_{\vec{G}})$ dargestellt werden. Dabei wird jedem eigenständigen Eintrag $e_i \in \mathcal{L}$ mit $i = 1, ..., n,\ n = |\mathcal{L}|,\ n \in I\!N_0$, bijektiv eine Ecke $v_i \in E$ zugeordnet und jede der Beziehungen $t_{e_i} < t_{e_{i+1}}$ zwischen den Einträgen e_i und e_{i+1} durch die gerichteten Kanten $(v_i, v_{i+1}) \in K_{\vec{G}}$ beschrieben:

$$E = \{v_i \mid v_i = e_i,\ e_i \in \mathcal{L},\ i = 1, ..., n,\ n = |\mathcal{L}|\}$$

und

$$K_{\vec{G}} = \{(v_i, v_{i+1}) \mid v_i, v_{i+1} \in E,\ i = 1, ..., n-1,\ n = |E|\}.$$

Für $\mathcal{L}$ wurde nun die Frage gestellt, wie die Authentizität, Reihenfolge und Vollständigkeit der Einträge e_i sichergestellt werden kann. Dabei bezieht sich

1. die Authentizität der Einträge e_i auf deren Inhalt t_{e_i} und d_{e_i},
2. die Reihenfolge auf die gerichteten Kanten $(v_i, v_{i+1}) \in K_{\vec{G}}$, und
3. die Vollständigkeit auf den Zusammenhang von $\vec{G}$.

Es scheint zunächst, als könnte die Frage dadurch beantwortet werden, daß $\mathcal{L}$ als Ganzes signiert und/oder verschlüsselt wird. Das Besondere bei der Beantwortung dieser Frage ist allerdings, daß $\mathcal{L}$ nicht als ein monolithisches Gebilde angesehen werden kann. Vielmehr muß berücksichtigt werden, daß ständig neue Einträge e_{n+1} an das Ende n von $\mathcal{L}$ angehängt werden, der Graph $\vec{G}$ also beständig wächst:

$$E \longmapsto E \cup \{v_{n+1}\} \quad \text{und} \quad K_{\vec{G}} \longmapsto K_{\vec{G}} \cup \{(v_n, v_{n+1})\}.$$

Als Lösung bietet sich daher an, jeden neuen Eintrag e_{n+1} für sich zu signieren und/oder zu verschlüsseln und ihn mit dem Eintrag e_n entsprechend der gerichteten Kante (v_n, v_{n+1}) in geeigneter Weise zu verketten, die es im weiteren Verlauf zu beschreiben gilt. Offenbar handelt es sich hierbei um zwei Teillösungen, die auf der einen Seite Punkt 1 berücksichtigen und auf der anderen Seite die Punkte 2 und 3 zu einem Punkt zusammenfassen. Die erste Teillösung bezieht sich dabei auf die Sicherung der Einträge $e_i \in \mathcal{L}$ und die zweite Teillösung auf die Sicherung der Kanten $(v_i, v_{i+1}) \in K_{\vec{G}}$ von $\vec{G}$. Die Eigenständigkeit der Einträge e_i bleibt dabei erhalten.

Allgemein Allgemeiner betrachtet führt diese Lösung zur sogenannten kryptographischen Verkettung. Wie schon in der Einleitung auf Seite 1 erwähnt, ist unter einer kryptographischen Verkettung zu verstehen, daß die Ecken eines Graphen, entsprechend den zwischen ihnen existierenden Kanten, mittels kryptographischer Verfahren miteinander verkettet werden. Die *Ziele der kryptographischen Verkettung eines Graphen* werden dabei folgendermaßen formuliert:

1. Kein Unberechtigter soll Ecken unbemerkt manipulieren können.
2. Kein Unberechtigter soll Ecken oder Kanten unbemerkt aus dem Graphen entfernen können.
3. Ein Berechtigter soll die Einhaltung der Bedingungen 1 und 2 überprüfen können.

Entsprechend dem oben genannten Beispiel wird unter 1. angenommen, daß Ecken einen Inhalt haben, den es zu sichern gilt. Dieser besteht hier aus t_{e_i} und d_{e_i} eines Eintrags $e_i \in \mathcal{L}$. Die Sicherung muß dabei nicht notwendigerweise durch die kryptographische Verkettung erfolgen. Die Sicherung der Kanten wird unter 2. genauer formuliert. Bezogen auf das Beispiel sind das die gerichteten Kanten $(v_i, v_{i+1}) \in K_{\vec{G}}$. Wichtige *Nebenbedingungen der kryptographischen Verkettung eines Graphen* sind:

1. Soll die Prüfbarkeit einer Ecke von der vorhergehenden Prüfung einer anderen Ecke abhängig sein?
2. Soll der Graph nur prüfbar oder auch auswertbar sein?
3. Soll der Graph abgeschlossen oder änderbar sein?
4. Sollen Berechtigungen nur für einen Teil des Graphen oder für den gesamten Graphen gelten?
5. Sollen sich die Bezeichnungen „Unberechtigter“ und „Berechtigter“ jeweils auf einen einzelnen, eine Gruppe oder alle Individuen beziehen?

Berechtigter und Unberechtigter Um mit den Begriffen Berechtigter und Unberechtigter besser umgehen zu können, werden die beiden etwas intuitiven nachfolgenden Definitionen angegeben. Die erste Definition teilt die Menge aller Individuen $\mathcal{I}$ in zwei Teilmengen auf:

Definition 3.1.1. $\mathcal{B} \subseteq \mathcal{I}$ sei die *Menge aller Berechtigten* und $\mathcal{U} \subseteq \mathcal{I}$ sei die *Menge aller Unberechtigten*, wobei $\mathcal{B} = \mathcal{I} \setminus \mathcal{U}$ bzw. $\mathcal{U} = \mathcal{I} \setminus \mathcal{B}$ gilt. □

Das heißt, $\mathcal{B}$ und $\mathcal{U}$ sind paarweise disjunkt. Bezugnehmend auf die Ziele der kryptographischen Verkettung wird definiert:

Definition 3.1.2. $\mathcal{B}_m$, $\mathcal{B}_e$ und $\mathcal{B}_p$ seien die Mengen aller Berechtigten, die jeweils

- eine Ecke unbemerkt manipulieren dürfen,
- eine Ecke oder Kante unbemerkt aus dem Graphen entfernen dürfen bzw.
- die Einhaltung der Bedingungen überprüfen dürfen.

Entsprechend sind $\mathcal{U}_m = \mathcal{I} \setminus \mathcal{B}_m$, $\mathcal{U}_e = \mathcal{I} \setminus \mathcal{B}_e$ und $\mathcal{U}_p = \mathcal{I} \setminus \mathcal{B}_p$ die Mengen aller Unberechtigten, die das jeweils nicht dürfen. □

Der Begriff *unbemerkt* soll in diesem Zusammenhang bedeuten, daß ein Individuum aus $\mathcal{I}$, das den vorhergehenden Zustand nicht kennt, nicht erkennen kann, daß sich etwas geändert hat.

3.2 Repräsentation der Graphen

Als nächstes wird für die weitere Behandlung der kryptographischen Verkettung eine spezielle Repräsentation der Graphen gewählt. Dadurch wird die Erläuterung der Prinzipien der kryptographischen Verkettung in Abschnitt 3.3 erleichtert. Ein Graph kann beispielsweise folgendermaßen repräsentiert werden:

1. Bildlich,
2. durch Angabe der Menge der Ecken und der Menge der Kanten,
3. durch Adjazenz- und/oder Inzidenzmatrizen oder

4. durch eine Menge von Ecken mit Zeigern, die auf diejenigen Ecken zeigen, die eine gemeinsame Kante mit der jeweiligen Ecke haben.

Diese Repräsentationen werden in den folgenden Absätzen ein wenig genauer erläutert, wobei dann die 4. Repräsentation gewählt wird.

Repräsentation 1 In Abschnitt 1.4 über die Graphentheorie wurde bereits beschrieben, wie sich ein Graph bildlich darstellen läßt. Auf den ersten Blick erscheint es relativ trivial, das entsprechende Bild zu sichern, da es nur in einer Datei abgelegt werden muß, die entsprechend signiert und/oder verschlüsselt werden kann. Die Umsetzung wird allerdings schon aufwendiger, da das Bild entweder eingescannt oder im Rechner gezeichnet werden muß. Dabei tritt das Problem auf, wie komplexe Graphen möglichst übersichtlich gezeichnet werden können. Die Frage nach der Erweiterbarkeit wird dadurch beeinträchtigt, daß dafür entweder die Signatur erneuert oder das Bild entschlüsselt und wieder verschlüsselt werden muß und ohne weitere Maßnahmen nichts darauf hindeutet, was sich im Vergleich zu vorher geändert hat. Außerdem kann die Frage, wie ein Graph repräsentiert wird, der automatisch im Rechner gezeichnet werden soll, einen logischen Zirkel oder unendlichen Regreß bewirken, wenn „bildlich“ als Antwort genannt wird. Sollte als Antwort eine der übrigen Repräsentationen genannt werden, könnte diese auch gleich als grundlegende Repräsentation gewählt werden.

Repräsentation 2 und 3 Die Repräsentationen 2 und 3 eignen sich gut, die Ecken und Kanten getrennt voneinander zu sichern, was auf der anderen Seite ungünstig sein kann, wenn sie gemeinsam gesichert werden sollen. Die 2. Repräsentation wurde bereits in Abschnitt 1.4 für die Definitionen zur Graphentheorie eingesetzt.

Unter 3. werden zwei Möglichkeiten zur Repräsentation eines Graphen eingeordnet, die hier nur für gerichtete Graphen $\vec{G} = (E, K_{\vec{G}})$ mit $E = \{v_1, ..., v_n\}$ und $K_{\vec{G}} = \{k_1, ..., k_m\}$ definiert werden. Für diese Arbeit sind sie nicht weiter von Bedeutung und werden nur des besseren Überblicks wegen erwähnt, weil sie an vielen anderen Stellen in der Graphentheorie eingesetzt werden (s. [Aig93, Jun94, SS89]). Eine *Adjazenzmatrix* ist eine $n \times n$-Matrix (a_{ij}) mit

$$a_{ij} = \begin{cases} 1 & \text{falls} \quad v_i v_j \in K_{\vec{G}}, \\ 0 & \text{sonst.} \end{cases}$$

Eine *Inzidenzmatrix* ist eine $n \times m$-Matrix (b_{ij}) mit

$$b_{ij} = \begin{cases} -1 & \text{falls} \quad v_i \text{ der Startpunkt von } k_j \text{ ist}, \\ 1 & \text{falls} \quad v_i \text{ der Endpunkt von } k_j \text{ ist}, \\ 0 & \text{sonst.} \end{cases}$$

Werden bei der 2. Repräsentation die Mengen als verkettete Listen realisiert, würde dies wiederum der 4. nahe kommen. Deshalb wird die 4. Repräsentation für die Behandlung der kryptographischen Verkettung gewählt und als nächstes genauer definiert.

Repräsentation 4 Die 4. Repräsentation kann sowohl (ungerichtete) Graphen $G = (E, K_G)$ als auch gerichtete Graphen $\vec{G} = (E, K_{\vec{G}})$ mittels einer *Menge von Gliedern* $\mathcal{G}$ repräsentieren, die nachfolgend spezifiziert wird.

Definition 3.2.1. $\mathcal{Z}$ ist eine *Menge von Kennungen* und die injektive (evtl. bijektive) Funktion

$$E_{\mathcal{Z}} : \; E \longrightarrow \mathcal{Z}$$

weist jeder Ecke $v \in E$ genau eine *Kennung* $k \in \mathcal{Z}$ zu. □

Die Menge von Kennungen $\mathcal{Z}$ kann zum Beispiel gleich $\mathbb{N}_0$ oder gleich der Menge von Schlüsseln $\mathcal{K}$ eines Kryptosystems $(\mathcal{M}, \mathcal{C}, \mathcal{K}, E_{K_1}, D_{K_2})$ sein.

Definition 3.2.2. Ein *Glied* $g \in \mathcal{G}$ ist ein Tripel (k_g, Z_g, v), bestehend aus

- einer Kennung $k_g \in \mathcal{Z}$,
- einer Menge $Z_g \subseteq \mathcal{Z}$ von Kennungen (auch *Zeiger* genannt) und
- einer Ecke $v \in E$

mit $k_g = E_{\mathcal{Z}}(v)$. □

Wegen der Injektivität von $E_{\mathcal{Z}}$ und wegen $k_g = E_{\mathcal{Z}}(v)$ gilt für die Kennungen k_{g_i} und k_{g_j} der Glieder $g_i, g_j \in \mathcal{G}$ mit $i \neq j$, daß sie paarweise verschieden sind. Daraus folgt, daß jedes Glied $g \in \mathcal{G}$ anhand seiner Kennung k_g innerhalb dieser Repräsentation eines Graphen eindeutig identifiziert werden kann. Jeder Ecke $v \in E$ wird demnach bijektiv ein Glied $g \in \mathcal{G}$ zugeordnet.

Für (ungerichtete) Graphen $G = (E, K_G)$ enthält die Menge Z_g eines Gliedes $g \in \mathcal{G}$ die Kennungen k_{g_i} derjenigen Glieder $g_i \in \mathcal{G}$, deren Ecken $v_i \in E$ eine gemeinsame Kante $\{v, v_i\} \in K_G$ mit der Ecke $v \in E$ des Gliedes g haben und jedes Glied g_i enthält in Z_{g_i} die Kennung k_g des Gliedes g. Kurz: $k_{g_i} \in Z_g$ und $k_g \in Z_{g_i}$ für $\{v, v_i\}$. Daraus folgt:

$$\begin{aligned} \mathcal{G} = \{g \mid \; & g = (k_g, Z_g, v), \\ & k_g = E_{\mathcal{Z}}(v), \\ & Z_g = \{k_{g_i} \mid k_{g_i} = E_{\mathcal{Z}}(v_i) \Longleftrightarrow \{v, v_i\} \in K_G\}, \\ & v \in E\}. \end{aligned}$$

Für gerichtete Graphen $\vec{G} = (E, K_{\vec{G}})$ gilt dagegen die Einschränkung, daß für eine gerichtete Kante $(v, v_i) \in K_{\vec{G}}$ nur in der Menge Z_g des Gliedes g, das zum Startpunkt v gehört, die Kennung k_{g_i} des Gliedes g_i, das für den Endpunkt v_i steht, eingetragen wird. Kurz: $k_{g_i} \in Z_g$ für (v, v_i). Daraus folgt:

$$\begin{aligned} \mathcal{G} = \{g \mid \; & g = (k_g, Z_g, v), \\ & k_g = E_{\mathcal{Z}}(v), \\ & Z_g = \{k_{g_i} \mid k_{g_i} = E_{\mathcal{Z}}(v_i) \Longleftrightarrow (v, v_i) \in K_{\vec{G}}\}, \\ & v \in E\}. \end{aligned}$$

Aus diesen Überlegungen ergeben sich die beiden folgenden Definitionen, für die g_i und g_j Glieder seien:

Definition 3.2.3. (g_i, g_j) ist eine *einfache (gerichtete) Verkettung* des Gliedes g_j mit dem Glied g_i, wenn $k_{g_j} \in Z_{g_i}$ und $k_{g_i} \notin Z_{g_j}$ gilt. □

Definition 3.2.4. $\{g_i, g_j\}$ ist eine *doppelte (ungerichtete) Verkettung* der Glieder g_i und g_j, wenn $k_{g_j} \in Z_{g_i}$ und $k_{g_i} \in Z_{g_j}$ gilt. □

Man kann erkennen, daß durch diese Repräsentation, analog zu der Beschreibung auf Seite 10, ein (ungerichteter) Graph $G = (E, K_G)$ in einen gerichteten Graphen $\vec{G} = (E, K_{\vec{G}})$ überführt wird. Daher gehen alle weiteren Überlegungen nur von $\mathcal{G}$ bzw. gerichteten Graphen $\vec{G} = (E, K_{\vec{G}})$ aus. Abschließend soll ein kleines Beispiel den Zusammenhang zwischen einem gerichteten Graphen $\vec{G} = (E, K_{\vec{G}})$ und seiner Repräsentation $\mathcal{G}$ demonstrieren:

Beispiel 3.2.1. $\vec{G} = (E, K_{\vec{G}})$ sei ein gerichteter Graph mit $E = \{v_0, v_1, v_2\}$ und $K_{\vec{G}} = \{v_0v_1, v_1v_0, v_1v_2\}$, wie er in Abbildung 3.1 dargestellt wird.

Wird jeder Ecke $v_i \in E$ mit $i = 0, 1, 2$ bijektiv ein Glied $g_i \in \mathcal{G}$ zugeordnet, ergibt sich $\mathcal{G} = \{g_0, g_1, g_2\}$. Anhand der Kanten aus $K_{\vec{G}}$ folgt für die Mengen der Kennungen Z_{g_i} der Glieder $g_i \in \mathcal{G}$:

- $Z_{g_0} = \{k_{g_1}\}$,
- $Z_{g_1} = \{k_{g_0}, k_{g_2}\}$ und
- $Z_{g_2} = \{\} = \emptyset$. □

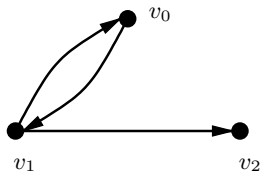

Abbildung 3.1: Gerichteter Graph $\vec{G}$ aus Beispiel 3.2.1

3.3 Prinzipien

Im vorhergehenden Abschnitt 3.2 wurde für ungerichtete und gerichtete Graphen die Repräsentation durch eine Menge von Gliedern $\mathcal{G}$ eingeführt. Auf Grundlage von $\mathcal{G}$ werden hier in den Abschnitten 3.3.2 und 3.3.3 zwei fundamentale Prinzipien zur kryptographischen Verkettung der Glieder $g \in \mathcal{G}$ und somit des dazugehörigen gerichteten Graphen $\vec{G} = (E, K_{\vec{G}})$ vorgestellt. Beide Prinzipien haben gemeinsam, daß sie die auf Seite 26 formulierten Ziele erreichen. Ihre Unterschiede werden dagegen bei den im nächsten Abschnitt 3.4 thematisierten Nebenbedingungen deutlich. Um Schwierigkeiten zu vermeiden wird daher vorausgesetzt, daß die kryptographische Verkettung für alle $g \in \mathcal{G}$ jeweils auf einem einheitlichen Prinzip beruht.

Authentizität und Geheimhaltung In Abschnitt 1.6 wurde dargelegt, daß die Grundaufgaben der Kryptographie in der Sicherung der Authentizität und der Geheimhaltung von Informationen bestehen. Die Sicherung der Authentizität zielt dabei darauf ab, den Ursprung einer Nachricht $M \in \mathcal{M}$ zu

beweisen, während die Sicherung der Geheimhaltung darauf abzielt, eine Nachricht $M \in \mathcal{M}$ geheimzuhalten. Bezogen auf das Erreichen der Ziele der kryptographischen Verkettung ist es notwendig, die Authentizität der Ecken und Kanten zu sichern. Die Geheimhaltung von Ecken oder Kanten ist lediglich von den Nebenbedingungen abhängig.

Vereinbarungen Für die weiteren Überlegungen sei

$$KS = (\mathcal{M}, \mathcal{C}, \mathcal{K}, E_{K_1}, D_{K_2})$$

ein Kryptosystem, daß die Authentizitätsanforderungen von Seite 14 erfüllt. Es soll zunächst offen bleiben, ob es sich dabei um ein Public-Key-Kryptosystem, mit E_{K_1} privat und D_{K_2} öffentlich, oder ein symmetrisches Kryptosystem, mit E_{K_1} und D_{K_2} privat, handelt.

Es wird davon ausgegangen, daß ein Berechtigter aus $\mathcal{B}_m$ oder $\mathcal{B}_e$ die Chiffriertransformation E_{K_1} kennt und ein Berechtigter aus $\mathcal{B}_p$ die Dechiffriertransformation D_{K_2}. Ob E_{K_1} ein Unberechtigter aus $\mathcal{U}_m$ oder $\mathcal{U}_e$ kennen darf und D_{K_2} ein Unberechtigter aus $\mathcal{U}_p$, hängt von den zu erfüllenden Nebenbedingungen ab. Im allgemeinen wird davon ausgegangen, daß sie die jeweilige Transformation nicht kennen.

3.3.1 Einfache kryptographische Verkettung

Die folgende Definition bildet die Grundlage für die kryptographische Verkettung der Glieder $g \in \mathcal{G}$.

Definition 3.3.1. $g, g_i \in \mathcal{G} \subseteq \mathcal{M}$ seien Glieder mit $i = 0, ..., n,\ n \in \mathbb{N}_0$.

$$E_{K_1}(g) = C_g \in \mathcal{C}$$

ist eine *einfache kryptographische Verkettung* des Gliedes g_i mit dem Glied g, wenn $k_{g_i} \in Z_g$ gilt. □

Je nachdem, ob KS ein Public-Key-Kryptosystem oder ein symmetrisches Kryptosystem ist, handelt es sich hierbei nur um die Signierung oder Verschlüsselung von g. Für den Fall, daß KS ein Public-Key-Kryptosystem ist und D_{K_2} im Gegensatz zur oben getroffenen Vereinbarung privat ist, kann KS auch zur Verschlüsselung von g eingesetzt werden.

Für die Ziele der kryptographischen Verkettung von $\vec{G}$ bedeutet dies, daß kein Unberechtigter aus $\mathcal{U}_m$ das Glied g und somit die Ecke $v \in E$ unbemerkt manipulieren kann (1. Ziel). Außerdem kann kein Unberechtigter aus $\mathcal{U}_e$ unbemerkt die Ecke v aus E entfernen ($E \longmapsto E \setminus \{v\}$) oder eine gerichtete Kante (v, v_i) aus $K_{\vec{G}}$ entfernen ($K_{\vec{G}} \longmapsto K_{\vec{G}} \setminus \{(v, v_i)\}$), solange C_g verfügbar ist (2. Ziel). In bezug auf $\vec{G}$ wird also die Ecke v gleichzeitig mit den gerichteten Kanten (v, v_i) gesichert. Allerdings bleiben die Glieder aus $\mathcal{G} \setminus \{g\}$ und die entsprechenden Ecken und gerichteten Kanten ungesichert. Schließlich kann ein Berechtigter aus $\mathcal{B}_p$ mittels D_{K_2} die Einhaltung der Bedingungen 1 und 2 überprüfen (3. Ziel).

Anwendung auf alle Glieder Es stellt sich nun die Frage, ob die Ziele der kryptographischen Verkettung erreicht werden, wenn diese einfache kryptographische Verkettung auf alle $g \in \mathcal{G}$ angewendet wird. Dafür sei

$$C_{\mathcal{G}} = \{C_g \mid C_g = E_{K_1}(g),\ g \in \mathcal{G}\} \subseteq \mathcal{C}.$$

Unter der Voraussetzung, daß $C_{\mathcal{G}}$ vollständig verfügbar ist, kann ein Berechtigter aus $\mathcal{B}_p$ für alle $C_g \in C_{\mathcal{G}}$ überprüfen, ob $D_{K_2}(C_g)$ mit dem entsprechendem $g \in \mathcal{G}$ übereinstimmt. Wenn er C_g als authentisch ansieht, kann er auf diese Weise feststellen, ob g manipuliert wurde.

Allerdings muß jetzt beachtet werden, daß die Authentizität aller $C_g \in C_{\mathcal{G}}$ und die Vollständigkeit von $C_{\mathcal{G}}$ nicht vorauszusetzen, sondern zu beweisen sind. Dafür wird auf die Authentizität in den nächsten Absätzen näher eingegangen und auf die Vollständigkeit in den Abschnitten 3.3.2 und 3.3.3.

Authentizität Speziell zum Feststellen der Authentizität aller $C_g \in C_{\mathcal{G}}$ muß unterschieden werden, ob E_{K_1} einem Unberechtigtem aus $\mathcal{U}_m$ oder $\mathcal{U}_e$

1. nicht bekannt oder
2. bekannt

ist.

Im ersten Fall kann ein Berechtigter aus $\mathcal{B}_p$ soweit gehen zu sagen, daß

$$g = D_{K_2}(C_g)$$

ist, wenn $D_{K_2}(C_g)$ einen „plausiblen“ Klartext ergibt. Dadurch kann auf die Speicherung von $\mathcal{G}$ und $\vec{G}$ verzichtet werden, weil sie sich aus $C_{\mathcal{G}}$ rekonstruieren lassen. Somit wird das 1. Ziel erreicht und entsprechend das 3. Ziel zur ersten Hälfte.

Für den zweiten Fall wird angenommen, daß ein Unberechtigter aus $\mathcal{U}_m$ oder $\mathcal{U}_e$ ein plausibles Glied g' erzeugen und daher ein $C_{g'} = E_{K_1}(g')$ für ein $C_g \in C_{\mathcal{G}}$ einsetzen kann. Das bedeutet, daß die Überprüfung der Plausibilität von $D_{K_2}(C_g)$ nicht ausreicht, um die Authentizität von C_g feststellen zu können. Zum Feststellen der Authentizität von C_g ist daher die Überprüfung einer weiteren Zusatzinformation nötig, die kein Unberechtigter aus $\mathcal{U}_m$ oder $\mathcal{U}_e$ abändern kann.

Zusatzinformation Eine unabänderbare Zusatzinformation kann vom Typ her entweder

1. öffentlich oder
2. privat

sein. Zum einem muß sie einem Berechtigtem aus $\mathcal{B}_p$ bekannt sein und zum anderem muß sie sich für ihn berechnungsmäßig einfach über ein festgelegtes Verfahren aus einem zu überprüfendem $C_g \in C_{\mathcal{G}}$ ableiten lassen. Dadurch kann er im oben geschilderten zweiten Fall ein $C_g \in C_{\mathcal{G}}$ als authentisch ansehen, wenn $D_{K_2}(C_g)$ plausibel ist und sich die ihm bekannte unabänderbare Zusatzinformation auch aus C_g ableiten läßt.

Unter einer öffentlichen Zusatzinformation ist zu verstehen, daß sie prinzipiell allen Individuen aus $\mathcal{I}$ bekannt ist. Zum Beispiel kann das der Hashwert $h_e(g)$ des ursprünglichen Gliedes $g \in \mathcal{G}$ oder der Hashwert $h_e(C_g)$ des ursprünglichen Elements $C_g \in C_\mathcal{G}$ sein, wobei h_e eine Einweg-Hashfunktion ist. Im übertragenen Sinne wird von diesem Prinzip beispielsweise in [HS91] für Zeitstempel Gebrauch gemacht.

Eine private Zusatzinformation darf einem Unberechtigtem aus $\mathcal{U}_m$ oder $\mathcal{U}_e$ weder bekannt noch für ihn berechnungsmäßig einfach zu bestimmen sein. Allgemein gesehen wird dafür beispielsweise in [SK98] für Log-Dateien ebenfalls eine Einweg-Hashfunktion eingesetzt. Als Alternative dazu kann zum Beispiel davon ausgegangen werden, daß D_{K_2} keinem Unberechtigtem aus $\mathcal{U}_m$ oder $\mathcal{U}_e$ bekannt ist und die Kennungen k_g der Glieder $g \in \mathcal{G}$ jeweils die private Zusatzinformation darstellen. Für einen Unberechtigten aus $\mathcal{U}_m$ oder $\mathcal{U}_e$ ist es dann nicht möglich, ein plausibles Glied g' mit der geforderten Kennung k_g zu erzeugen, um unerkannt $C_{g'} = E_{K_1}(g')$ für $C_g \in C_\mathcal{G}$ einsetzen zu können.

Plausibilität Zur Prüfung der Plausibilität kann g zum Beispiel vor der Transformation durch E_{K_1} mit einem Code zur Fehlererkennung codiert werden (s. Abschnitt 1.5 und Abbildung 2.6). Nach der Transformation durch D_{K_2} wäre dann $D_{K_2}(C_g)$ plausibel, wenn bei der Prüfung der Codierung kein Fehler auftritt. Eine andere Methode wäre, daß k_g als Bedingung den Fingerabdruck der Einweg-Hashfunktion $h_e((Z_g, v))$ darstellt. $D_{K_2}(C_g)$ wäre dann plausibel, wenn die Bedingung erfüllt ist.

3.3.2 Unabhängige kryptographische Verkettung

Zunächst einmal soll die Formulierung „unabhängig“ ausdrücken, daß die Dechiffriertransformation D_{K_2} im Gegensatz zum nächsten Abschnitt 3.3.3 nicht von einem $C_g \in C_\mathcal{G}$ abhängig ist.

Desweiteren wird im Gegensatz zu oben nicht vorausgesetzt, daß $C_\mathcal{G}$ vollständig verfügbar ist. Vielmehr lautet die Frage nun:

Wie kann die Vollständigkeit von $C_\mathcal{G}$ festgestellt werden?

Offenbar wird hier das Problem des Feststellens der Vollständigkeit, daß durch das 2. Ziel beschrieben wird, von $\vec{G}$ nach $C_\mathcal{G}$ verschoben. Das Problem kann auf mindestens zwei verschiedene Arten gelöst werden:

1. Mittels eines externen Zählers oder
2. mittels bestimmter Anforderungen an $\vec{G}$.

Zähler Der einfachste Weg, die Vollständigkeit festzustellen und somit das 2. Ziel und die zweite Hälfte des 3. Zieles zu erreichen, ist sicherlich der, die Anzahl

$$a_\mathcal{G} = |\mathcal{G}| \quad \text{mit} \quad a_\mathcal{G} \in \mathbb{N}_0$$

der Glieder $g \in \mathcal{G}$ extern zu speichern. $C_\mathcal{G}$ wäre dann vollständig, wenn

1. alle $D_{K_2}(C_g)$ plausibel sind,

2. alle $C_g \in C_\mathcal{G}$ paarweise verschieden sind und

3. $|C_\mathcal{G}|$ gleich $a_\mathcal{G}$ ist.

Der Nachteil dieses Weges ist, daß durch die Einführung von $a_\mathcal{G}$ eine mögliche Fehlerquelle und ein Angriffsziel hinzukommen. Vor Angriffen müßte $a_\mathcal{G}$ also zum Beispiel mittels $C_{a_\mathcal{G}} = E_{K_1}(a_\mathcal{G})$ genauso gesichert werden wie jedes einzelne $g \in \mathcal{G}$, wobei dann eine Prüfung von $C_\mathcal{G}$ einen Fehler melden würde, wenn $C_{a_\mathcal{G}}$ fehlt. Bevor dieser Weg weiter unten im Hinblick auf die Nebenbedingungen der kryptographischen Verkettung diskutiert wird, stellen die nächsten Absätze den zweiten möglichen Weg vor.

Anforderungen Die Vollständigkeit von $C_\mathcal{G}$ kann dadurch festgestellt werden, daß im Vorfeld an $\mathcal{G}$ bzw. $\vec{G}$ bestimmte zu erfüllende Anforderungen gestellt werden, wovon eine nachfolgend in zwei Teilen beschrieben wird. Der Einfachheit in den Formulierungen halber wird angenommen, daß alle vorhandenen $C_g \in C_\mathcal{G}$ authentisch sind, was ja wegen der Geheimhaltung von E_{K_1} anhand der Plausibilität überprüft werden kann. Desweiteren sei daran erinnert, daß die gerichteten Kanten $(v, v_i) \in K_{\vec{G}}$ gemeinsam mit der Ecke $v \in E$ im Glied g durch $C_g = E_{K_1}(g)$ gesichert werden. Aufgrund dieser Tatsache ist es nicht möglich, unbemerkt eine einzelne gerichtete Kante (v, v_j) mit $v_j \in E$ dem gerichteten Graphen $\vec{G}$ in $K_{\vec{G}}$ hinzuzufügen ($K_{\vec{G}} \longmapsto K_{\vec{G}} \cup \{(v, v_j)\}$) oder eine einzelne gerichtete Kante $(v, v_j) \in K_{\vec{G}}$ zu entfernen ($K_{\vec{G}} \longmapsto K_{\vec{G}} \setminus \{(v, v_j)\}$), ohne daß auch $v \in E$ und alle anderen $(v, v_i) \in K_{\vec{G}}$ entfernt würden. Letzteres wird durch das folgende erste Beispiel 3.3.1 veranschaulicht:

Beispiel 3.3.1. Es sei $\mathcal{G} = \{g_0, g_1, g_2\}$ eine Menge von Gliedern, und es gilt $k_{g_1} \in Z_{g_0}$ und $k_{g_2} \in Z_{g_1}$. Das heißt, daß die entsprechenden Ecken $v_0, v_1, v_2 \in E$ des gerichteten Graphen $\vec{G} = (E, K_{\vec{G}})$ mit $K_{\vec{G}} = \{v_0v_1, v_1v_2\}$ über den gerichteten einfachen Weg (v_0v_1, v_1v_2) zusammenhängen (s. a. Abb. 3.2). Weiter sei statt der vollständigen Menge $C_\mathcal{G}$ nur die (unvollständige) Menge $C'_{\mathcal{G}'} = \{C_{g_1}\} = C_\mathcal{G} \setminus \{C_{g_0}, C_{g_2}\}$ vorhanden, die es zu prüfen gilt.

Die Prüfung beginnt damit, daß durch D_{K_2} die (unvollständige) Menge der Glieder $\mathcal{G}' = \{g_1\} = \mathcal{G} \setminus \{g_0, g_2\}$ erstellt wird. Danach läßt sich aus $\mathcal{G}'$ der (unvollständige) gerichtete Graph $\vec{G}' = (E', K'_{\vec{G}'})$ mit $E' = \{v_1\} = E \setminus \{v_0, v_2\}$ und $K'_{\vec{G}'} = \{v_1v_2\} = K_{\vec{G}} \setminus \{v_0v_1\}$ ableiten (s. a. Abb. 3.2).

Bei $\vec{G}'$ fällt nun auf, daß $v_1v_2 \in K'_{\vec{G}'}$ auf eine Ecke v_2 zeigt, die nicht Element von E' ist. Wegen der Authentizität von C_{g_1} folgt daraus, daß $C'_{\mathcal{G}'}$ gegenüber $C_\mathcal{G}$ unvollständig ist. □

Am Ergebnis von Beispiel 3.3.1 sind zweierlei Punkte hervorzuheben, wovon vor allem der zweite für die weitere Diskussion von Bedeutung ist:

1. Die Ursache der Unvollständigkeit ist nicht feststellbar, da sie entweder auf einem Fehler bei der Erstellung von $C'_{\mathcal{G}'}$ beruhen kann oder auf dem nachträglichen Entfernen von C_{g_0} und C_{g_2} aus $C_\mathcal{G}$, so daß nur noch $C'_{\mathcal{G}'}$ übrig blieb.

2. Zwar kann anhand der gerichteten Kante v_1v_2 festgestellt werden, daß die Ecke v_2 fehlt, es ist aber nicht erkennbar, daß die Ecke v_0 und die gerichtete Kante v_0v_1 existiert haben.

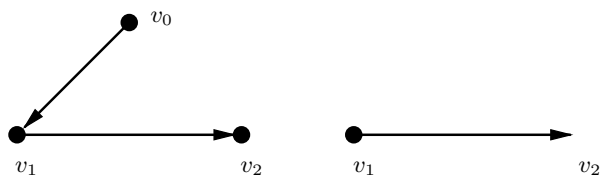

Abbildung 3.2: $\vec{G}$ und $\vec{G'}$ aus Beispiel 3.3.1

Der erste Punkt ist nicht weiter von Bedeutung, da für das 2. Ziel der kryptographischen Verkettung ja nur die Unvollständigkeit feststellbar sein soll und nicht deren Ursache. Daher wird dieser Punkt nicht weiter ausgeführt.

Aus dem zweiten Punkt folgt anhand der Ecke v_2, daß es in einem gerichteten Graphen $\vec{G} = (E, K_{\vec{G}})$ nicht möglich ist, eine Ecke $v \in E$ unerkannt aus E zu entfernen ($E \longmapsto E \setminus \{v\}$), solange ein *Zeuge der Existenz von* v in Form einer gerichteten Kante $(u, v) \in K_{\vec{G}}$ mit $u \in E$ existiert bzw. $P_v \neq \emptyset$ ist. Für einen gerichteten Kantenzug $(v_0v_1, ..., v_{n-1}v_n)$ mit $v_0, ..., v_n \in E$ und $n \in \mathbb{N}$ bedeutet dies, daß die gerichtete Kante $v_{i-1}v_i \in K_{\vec{G}}$ für $i = 1, ..., n$ Zeuge der Existenz der Ecke v_i ist, weshalb v_i nicht unerkannt aus E entfernt werden kann. Das gleiche gilt für einen geschlossenen gerichteten Kantenzug mit der Besonderheit, daß $v_0 = v_n$ ist, wodurch $v_{n-1}v_n \in K_{\vec{G}}$ Zeuge der Existenz von v_0 ist und somit auch v_0 nicht unerkannt aus E entfernt werden kann.

Wie an der Ecke v_0 aus Beispiel 3.3.1 zu sehen ist, kann andersherum in einem gerichteten Graphen $\vec{G} = (E, K_{\vec{G}})$ eine Ecke $v \in E$ unerkannt aus E entfernt werden ($E \longmapsto E \setminus \{v\}$), wenn kein Zeuge der Existenz von v existiert. Soll in diesem Sinne der Zeuge der Existenz von v eine gerichtete Kante (u, v) sein, so muß $(u, v) \notin K_{\vec{G}}$ für alle $u \in E$ gelten, um v unerkannt entfernen zu können.

Vor der Zusammenfassung der in den letzten beiden Abschnitten aufgeführten Folgerungen muß noch das zweite Beispiel 3.3.2 geklärt werden:

Beispiel 3.3.2. $\vec{G} = (E, K_{\vec{G}})$ sei ein gerichteter Graph mit $E = \{v_0, v_1, v_2, v_3\}$ und $K_{\vec{G}} = \{v_0v_1, v_1v_0, v_2v_3, v_3v_2\}$ (s. a. Abb. 3.3), der durch eine entsprechende Menge $\mathcal{G}$ repräsentiert wird, aus der $C_{\mathcal{G}}$ folgt. Desweiteren ist $C'_{\mathcal{G}'} = \{C_{g_0}, C_{g_1}\} = C_{\mathcal{G}} \setminus \{C_{g_2}, C_{g_3}\}$ woraus sich $\vec{G'} = (E', K'_{\vec{G}'})$ mit $E' = E \setminus \{v_2, v_3\}$ und $K'_{\vec{G}'} = K_{\vec{G}} \setminus \{v_2v_3, v_3v_2\}$ ableiten läßt (s. a. Abb. 3.3).

Entgegen Beispiel 3.3.1 zeigt diesmal keine Kante aus $K'_{\vec{G}'}$ auf eine Ecke aus E, die nicht in E' vorhanden ist. Daher scheint die Menge $C'_{\mathcal{G}'}$ vollständig zu sein, also gleich $C_{\mathcal{G}}$ zu sein, was sie aber tatsächlich nicht ist. □

Im obigen Beispiel 3.3.2 enthält $\vec{G}$ zwei gerichtete geschlossene Kantenzüge $W_1 = (v_0v_1, v_1v_0)$ und $W_2 = (v_2v_3, v_3v_2)$ und ist offenbar nicht zusammenhängend. Außerdem ist v_0v_1 Zeuge der Existenz von v_1, v_1v_0 von v_0, v_2v_3 von v_3 und v_3v_2 von v_2. Das heißt, es ist keine Kante des einen Kantenzuges Zeuge der Existenz einer Ecke des anderen Kantenzuges. Unter der Annahme, daß nur $\vec{G'}$ vorhanden ist, also von W_2 alle Ecken und Kanten fehlen, kann nicht festgestellt werden, daß v_2 und v_3 fehlen ($v_2, v_3 \notin E'$), da für sie kein Zeuge der

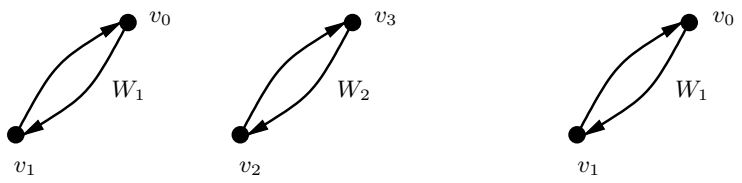

Abbildung 3.3: $\vec{G}$ und $\vec{G'}$ aus Beispiel 3.3.2

Existenz in $K'_{\vec{G'}}$ vorhanden ist ($v_2v_3, v_3v_2 \notin K'_{\vec{G'}}$), obwohl in $K_{\vec{G}}$ für alle Ecken aus E ein Zeuge der Existenz vorhanden ist. Diesem Umstand kann dadurch Abhilfe geleistet werden, daß für eine der Ecken v_2 oder v_3 von W_2 ein Zeuge der Existenz in Form einer gerichteten Kante $k_1 \in \{v_0v_2, v_0v_3, v_1v_2, v_1v_3\}$ in $K_{\vec{G}}$ eingefügt wird ($K_{\vec{G}} \longmapsto K_{\vec{G}} \cup \{k_1\}$). Das heißt, es wird eine Kante k_1 eingefügt, die von W_1 nach W_2 zeigt. Entsprechend muß eine Kante k_2 eingefügt werden, die von W_2 nach W_1 zeigt, um ein Fehlen von W_1 erkennbar zu machen. Für den resultierenden gerichteten Graphen $(E, K_{\vec{G}} \cup \{k_1, k_2\})$ ergibt sich, daß er stark zusammenhängend ist.

Folgerung Zusammenfassend läßt sich daraus schließen, daß das 2. Ziel der kryptographischen Verkettung und die zweite Hälfte vom 3. Ziel erreicht werden, wenn für alle Glieder $g \in \mathcal{G}$ eine einfache kryptographische Verkettung vorgenommen wird ($\mathcal{G} \longrightarrow C_{\mathcal{G}}$) und der dazugehörige gerichtete Graph $\vec{G} = (E, K_{\vec{G}})$ stark zusammenhängend ist. Es muß also nicht nur für alle $v \in E$ einen Zeugen der Existenz in Form einer gerichteten Kante $(u, v) \in K_{\vec{G}}$ mit $u \in E$ geben, sondern auch für alle gerichteten geschlossenen Kantenzüge aus $K_{\vec{G}}$, wenn ihre Anzahl größer als 1 ist. Unter Berücksichtigung des 1. Ziels folgt daraus:

Definition 3.3.2. $(\mathcal{M}, \mathcal{C}, \mathcal{K}, E_{K_1}, D_{K_2})$ sei ein Kryptosystem, und ein gerichteter Graph $\vec{G} = (E, K_{\vec{G}})$ sei durch eine Menge von Gliedern $\mathcal{G} \subseteq \mathcal{M}$ repräsentiert.

$$C_{\mathcal{G}} = \{C_g \mid C_g = E_{K_1}(g),\ g \in \mathcal{G}\} \subseteq \mathcal{C}$$

ist eine *geschlossene unabhängige kryptographische Verkettung* (*gukV*) der Glieder $g \in \mathcal{G}$, wenn für alle $v_0 \neq v_n$ ein gerichteter Kantenzug $(v_0v_1, ..., v_{n-1}v_n)$ mit $v_0, ..., v_n \in E$ und $v_{i-1}v_i \in K_{\vec{G}}$ für $i = 1, ..., n$ existiert. □

Der Begriff „geschlossen“ soll symbolisieren, daß $\vec{G}$ stark zusammenhängend ist und deshalb „in sich geschlossen“ ist (vgl. Def. 3.3.3). Sollte es für eine Ecke v oder einen gerichteten geschlossenen Kantenzug W keinen Zeugen der Existenz in Form einer gerichteten Kante (u, v) geben, was für v bedeutet, daß $P_v = \emptyset$ ist, dann muß für v bzw. W ein geeigneter Zeuge „außerhalb“ von $\vec{G}$ existieren, um ein Fehlen von v bzw. W erkennen zu können. In diesem Sinne ist der Zähler $a_{\mathcal{G}}$ ein solcher Zeuge.

Wurzeln, Quellen und Senken Die Nichtexistenz eines Zeugen der Existenz für v in Form einer gerichteten Kante (u, v) bedeutet zugleich, daß v wegen $P_v = \emptyset$ eine Quelle ist. Wenn nun für v ein Zeuge der Existenz außerhalb

von $\vec{G}$ existiert, reicht es aus, daß für alle $w \in E \setminus \{v\}$ ein gerichteter Kantenzug $(v_0v_1, ..., v_{n-1}v_n)$ mit Startpunkt $v_0 = v$, Endpunkt $v_n = w$, $v_0, ..., v_n \in E$ und $v_{i-1}v_i \in K_{\vec{G}}$ für $i = 1, ..., n$ existiert, um die Vollständigkeit von $C_{\mathcal{G}}$ feststellen zu können (s. „Zeuge der Existenz“ auf Seite 35). Im Gegensatz zu Definition 3.3.2 braucht $\vec{G'} = (E', K'_{\vec{G'}})$ mit $E' = E \setminus \{v\}$ und $K'_{\vec{G'}} = K_{\vec{G}} \setminus (\{k \mid k \in K_{\vec{G}},\ k = (u, v),\ u \in P_v\} \cup \{k \mid k \in K_{\vec{G}},\ k = (v, w),\ w \in S_v\})$ dabei nicht stark zusammenhängend zu sein. Allgemeiner betrachtet sollte v eine Wurzel sein, womit sich folgende Definition ergibt:

Definition 3.3.3. Die Voraussetzungen seien wie in Definition 3.3.2 gegeben.

$$(C_{\mathcal{G}}, k_{g_r}) \quad \text{mit} \quad C_{g_r} \in C_{\mathcal{G}}$$

ist eine *unabhängige kryptographische Verkettung mit Wurzel* $r \in E$ (*ukVmW*), wenn für alle $w \in E \setminus \{r\}$ ein gerichteter Kantenzug $(v_0v_1, ..., v_{n-1}v_n)$ mit $v_0 = r$, $v_n = w$, $v_0, ..., v_n \in E$ und $v_{i-1}v_i \in K_{\vec{G}}$ für $i = 1, ..., n$ existiert. □

Die Kennung k_{g_r} ist dabei außerhalb von $\vec{G}$ Zeuge der Existenz des Gliedes $D_{K_2}(C_{g_r}) = g_r \in \mathcal{G}$ und somit der Wurzel r. Außerdem kann eine gukV für alle Ecken $r \in E$ als eine ukVmW r angesehen werden, weil bei einer gukV von jeder Ecke r zu allen anderen Ecken $w \in E \setminus \{r\}$ ein gerichteter Kantenzug $v_0v_1, ..., v_{n-1}v_n$ mit $v_0 = r$ und $v_n = w$ existiert.

Für eine Senke $t \in E$ bleibt nur anzumerken, daß die von der Menge ihrer Vorgänger P_t ausgehenden gerichteten Kanten $(u, t) \in K_{\vec{G}}$ mit $u \in P_t$ Zeugen der Existenz von t sind. Von t geht aber, weil die Menge ihrer Nachfolger $S_t = \emptyset$ ist, keine Kante (t, w) als Zeuge der Existenz einer Ecke $w \in E$ aus.

Sonderfall Einen Sonderfall stellt die Situation $C_{\mathcal{G}} = \emptyset$ dar, weil sie zum einen dadurch hervorgerufen werden kann, daß alle $C_g \in C_{\mathcal{G}}$ aus $C_{\mathcal{G}}$ entfernt werden ($C_{\mathcal{G}} \longmapsto \emptyset$), oder weil sie zum anderen für den leeren gerichteten Graphen $\vec{G} = (E, K_{\vec{G}})$ mit $E = \emptyset$ und somit $K_{\vec{G}} = \emptyset$ steht ($\vec{G} = (\emptyset, \emptyset) \longrightarrow \mathcal{G} = \emptyset \longrightarrow C_{\mathcal{G}} = \emptyset$). Daraus folgt, daß ein geeigneter Zeuge außerhalb von $\vec{G}$ existieren muß, der bezeugt, ob E gleich $\emptyset$ ist oder nicht.

3.3.3 Abhängige kryptographische Verkettung

Im Gegensatz zum vorhergenden Abschnitt 3.3.2 soll nun die Dechiffriertransformation D_{K_2} für alle $C_{g_v} \in C_{\mathcal{G}}$ jeweils von C_{g_v} „abhängig“ sein. Dafür ist der Schlüssel $K_2 \in \mathcal{K}$ gleich der Kennung k_{g_v} des zu C_{g_v} gehörigen Gliedes $g_v \in \mathcal{G}$, weshalb wiederum $\mathcal{Z} = \mathcal{K}$ für $k_{g_v} = E_{\mathcal{Z}}(v)$ gilt (s. a. Def. 3.2.1 und Def. 3.2.2). Aus der Injektivität von $E_{\mathcal{Z}}$ folgt dabei, daß für alle $C_{g_v} \in C_{\mathcal{G}}$ jeweils eine eigene Dechiffriertransformation $D_{K_2=k_{g_v}}$ benötigt wird.

Allgemein Die Überprüfung von $C_{\mathcal{G}}$ durch einen Berechtigten aus $\mathcal{B}_p$ soll mit einem $C_{g_v} \in C_{\mathcal{G}}$ beginnen. Dafür wird angenommen, daß auf die zu C_{g_v} gehörige Ecke $v \in E$ mindestens eine gerichtete Kante $(u, v) \in K_{\vec{G}}$ zeigt, also $P_v \neq \emptyset$ ist. Daraus folgt wiederum, daß $k_{g_v} \in Z_{g_u}$ in den zu $C_{g_u} \in C_{\mathcal{G}}$ gehörigen Gliedern $g_u \in \mathcal{G}$ der Ecken $u \in P_v$ ist. Es muß nun unterschieden werden, ob der Berechtigte die Kennung k_{g_v} zum Entschlüsseln von C_{g_v} mittels $D_{K_2=k_{g_v}}$ kennt oder nicht:

- Kennt er sie, kann er C_{g_v} entschlüsseln.
- Kennt er sie nicht, kann er C_{g_v} trotzdem entschlüsseln, wenn er zunächst ein C_{g_u} mittels $D_{K_2=k_{g_u}}$ entschlüsselt, um $k_{g_v} \in Z_{g_u}$ zu erhalten. Das bedeutet wiederum, daß nun für jede Kennung k_{g_u} rekursiv die gleiche Unterscheidung gemacht werden muß, wie zuvor für k_{g_v} (s. dazu *breadth first search* in [Aig93, Jun94]).

Daraus läßt sich allgemein ableiten, daß ein Berechtigter aus $\mathcal{B}_p$ ein $C_{g_v} \in C_\mathcal{G}$

- *direkt* entschlüsseln kann, wenn er die Kennung k_{g_v} kennt, oder
- *indirekt* über ein $C_{g_u} \in C_\mathcal{G}$ entschlüsseln kann, wenn er die Kennung k_{g_u} kennt und ein gerichteter Kantenzug $(v_0v_1, ..., v_{n-1}v_n)$ mit $v_0 = u$, $v_n = v$, $v_0, ..., v_n \in E$ und $v_{i-1}v_i \in K_{\vec{G}}$ für $i = 1, ..., n$ existiert.

Gerichteter einfacher Weg und gerichteter einfacher Kreis Für einen gerichteten einfachen Weg $(v_0v_1, ..., v_{n-1}v_n)$ mit $g_i = (k_{g_i}, Z_{g_i}, v_i) \in \mathcal{G}$ für $i = 0, ..., n$ sowie $k_{g_j} \in Z_{g_{j-1}}$ für $j = 1, ..., n$ und der entsprechenden Menge $C_\mathcal{G}$ bedeutet dies, daß mittels der Kennung k_{g_i} für $i = 0, ..., n$ das Element $C_{g_i} \in C_\mathcal{G}$ direkt entschlüsselt werden kann und alle $C_{g_j} \in C_\mathcal{G}$ mit $j > i$ bzw. $j = i + 1, ..., n$ indirekt entschlüsselt werden können:

$$g_j = D_{K_2}(C_{g_j}) \quad \text{mit} \quad K_2 = k_{g_j} \in Z_{g_{j-1}}.$$

Alle $C_{g_j} \in C_\mathcal{G}$ mit $j < i$ bzw. $j = 0, ..., i - 1$ können nicht auf diese Weise entschlüsselt werden. Das bedeutet, daß ein Berechtigter aus $\mathcal{B}_p$ das Element C_{g_j} nicht entschlüsseln kann, wenn er nur k_{g_i} kennt und kein gerichteter Kantenzug mit Startpunkt v_i und Endpunkt v_j existiert.

Unter dieser Voraussetzung kann mittels eines gerichteten einfachen Weges zum Beispiel eine *Hierarchie von Berechtigungen* realisiert werden. Dafür erhält ein Berechtigter aus $\mathcal{B}_p$, der die Elemente $C_{g_i}, ..., C_{g_n} \in C_\mathcal{G}$ für $i = 0, ..., n$ der letzten $n - i + 1$ Ecken entschlüsseln darf, die Kennung k_{g_i}.

Für einen gerichteten einfachen Kreis gilt die Besonderheit $v_0 = v_n$, weshalb von jeder Ecke v_i zu jeder anderen Ecke v_j für $i \neq j$, $i, j = 0, ..., n$ ein gerichteter Kantenzug mit Startpunkt v_i und Endpunkt v_j existiert. Daher kann hier anhand jeder Kennung k_{g_i} für $i = 0, ..., n$ jedes $C_{g_j} \in C_\mathcal{G}$ mit $j = 0, ..., i - 1, i + 1, ..., n$ indirekt entschlüsselt werden (s. a. „Zeuge der Existenz“ auf Seite 35).

Folgerung Die Überlegungen zur Vollständigkeit von $C_\mathcal{G}$ aus dem vorhergehenden Abschnitt 3.3.2 gelten hier weiterhin. Daher ergibt sich aus Definition 3.3.2 und den obigen Überlegungen zur Abhängigkeit die folgende Definition:

Definition 3.3.4. $(\mathcal{M}, \mathcal{C}, \mathcal{K}, E_{K_1}, D_{K_2})$ sei ein Kryptosystem, und ein gerichteter Graph $\vec{G} = (E, K_{\vec{G}})$ sei durch eine Menge von Gliedern $\mathcal{G} \subseteq \mathcal{M}$ repräsentiert.

$$C_\mathcal{G} = \{C_g \mid C_g = E_{K_1}(g),\ k_g = K_2 \in \mathcal{K},\ g \in \mathcal{G}\} \subseteq \mathcal{C}$$

ist eine *geschlossene abhängige kryptographische Verkettung* (*gakV*) der Glieder $g \in \mathcal{G}$, wenn für alle $v_0 \neq v_n$ ein gerichteter Kantenzug $(v_0v_1, ..., v_{n-1}v_n)$ mit $v_0, ..., v_n \in E$ und $v_{i-1}v_i \in K_{\vec{G}}$ für $i = 1, ..., n$ existiert. □

Wurzeln, Quellen und Senken Da auch die Überlegungen zu Wurzeln, Quellen und Senken aus dem vorhergehenden Abschnitt 3.3.2 hier weiterhin gelten, muß Definition 3.3.3 nur um $k_g = K_2 \in \mathcal{K}$ erweitert werden, woraus folgt:

Definition 3.3.5. Die Voraussetzungen seien wie in Definition 3.3.4 gegeben.

$$(C_{\mathcal{G}}, k_{g_r}) \quad \text{mit} \quad C_{g_r} \in C_{\mathcal{G}}$$

ist eine *abhängige kryptographische Verkettung mit Wurzel* $r \in E$ ($akVmW$), wenn für alle $w \in E \setminus \{r\}$ ein gerichteter Kantenzug $(v_0v_1, ..., v_{n-1}v_n)$ mit $v_0 = r$, $v_n = w$, $v_0, ..., v_n \in E$ und $v_{i-1}v_i \in K_{\vec{G}}$ für $i = 1, ..., n$ existiert. □

Mehrfach abhängig Es ist zum Beispiel auch denkbar, daß die Dechiffriertransformation D_{K_2} für alle $C_{g_v} \in C_{\mathcal{G}}$ nicht jeweils von C_{g_v} abhängig ist, sondern jeweils von allen $C_{g_u} \in C_{\mathcal{G}}$ mit $u \in P_v$. Diese Möglichkeit wird hier allerdings nicht weiter ausgeführt und wurde nur der Vollständigkeit halber erwähnt.

3.3.4 Vergleich

Von den Zielen der kryptographischen Verkettung eines Graphen (s. Seite 26) kann eine einfache kryptographische Verkettung (s. Abschnitt 3.3.1)

$$C_{\mathcal{G}} = \{C_g \mid C_g = E_{K_1}(g),\ g \in \mathcal{G}\} \subseteq \mathcal{C}$$

aller Glieder $g \in \mathcal{G} \subseteq \mathcal{M}$ des gerichteten Graphen $\vec{G} = (E, K_{\vec{G}})$ zwar das 1. Ziel ganz und das 3. Ziel zur ersten Hälfte erreichen, nicht aber das 2. Ziel oder das 3. Ziel zur zweiten Hälfte. Zum Erreichen aller Ziele kann zum Beispiel wie bei den Definitionen zur unabhängigen und zur abhängigen kryptographischen Verkettung zusätzlich gefordert werden, daß $\vec{G}$ stark zusammenhängend ist (s. Def. 3.3.2 und Def. 3.3.4).

Die unabhängige und die abhängige kryptographische Verkettung unterscheiden sich in der Zeitkomplexität beim Entschlüsseln eines $C_{g_v} \in C_{\mathcal{G}}$, wenn nur eine Dechiffriertransformation D_{K_2} zum Entschlüsseln aller $C_g \in C_{\mathcal{G}}$ bekannt ist. Die Zeitkomplexität beträgt

- bei einer unabhängigen kryptographischen Verkettung $O(1)$, da alle $C_g \in C_{\mathcal{G}}$ direkt entschlüsselt werden können, und
- bei einer abhängigen kryptographischen Verkettung
  - $O(1)$ für $k_{g_v} = K_2$, wenn C_{g_v} direkt entschlüsselt werden kann, und
  - $O(n+1)$ für $k_{g_v} \neq K_2$, wenn C_{g_v} nur indirekt über $C_{g_u} \in C_{\mathcal{G}}$ mit $k_{g_u} = K_2$ entschlüsselt werden kann und der gerichtete Kantenzug von u nach v die Länge n hat.

Für den gerichteten Kantenzug kann dabei gefordert werden, daß er möglichst kurz bzw. n möglichst klein sein muß. Die Aufgabe besteht dann darin, einen kürzesten gerichteten einfachen Weg von u nach v zu finden. Dieses Suchproblem

kann mit dem Algorithmus von Dijkstra (s. z. B. [Aig93, Jun94]) gelöst werden, wenn $\vec{G}$ bekannt ist und soll hier ansonsten nicht weiter ausgeführt werden.

Im Vorgriff auf den nachfolgenden Abschnitt 3.4 bleibt noch zu erwähnen, daß sich bei einer abhängigen kryptographischen Verkettung mittels einer geschickten Plazierung von Quellen und Senken oder einer entsprechenden Handhabung der Schlüssel einige Nebenbedingungen realisieren lassen, für die bei einer unabhängigen kryptographischen Verkettung zusätzliche Maßnahmen notwendig sind.

Außerdem kann es durchaus vorkommen, daß zum Verschlüsseln eines Gliedes $g \in \mathcal{G}$ selbst wieder eine bestimmte kryptographische Verkettung eingesetzt werden muß, wenn E_{K_1} auf einer Blockchiffrierung beruht und für g mehrere Blöcke benötigt werden (s. a. Kapitel 9 in [Schn96]).

3.4 Nebenbedingungen

Es soll nun untersucht werden, welche Einflüsse die auf Seite 26 aufgeführten Nebenbedingungen auf eine kryptographische Verkettung $C_{\mathcal{G}}$ des gerichteten Graphen $\vec{G}$ haben, der durch $\mathcal{G}$ repräsentiert wird. Dafür werden die Nebenbedingungen jeweils einleitend genauer spezifiziert.

3.4.1 Prüfbarkeit

Die erste zu klärende Nebenbedingung bezieht sich darauf, ob die Prüfbarkeit einer Ecke $v \in E$ von der vorhergehenden Prüfung einer anderen Ecke $u \in E \setminus \{v\}$ abhängig sein soll oder nicht. Diese Nebenbedingung wird hier von $\vec{G}$ auf $C_{\mathcal{G}}$ übertragen. Es ist daher die Frage, ob die Prüfbarkeit des zu v gehörigen Elements $C_{g_v} \in C_{\mathcal{G}}$ von der vorhergehenden Prüfung des zu u gehörigen Elements $C_{g_u} \in C_{\mathcal{G}} \setminus \{C_{g_v}\}$ abhängig sein soll oder nicht. Dadurch betrifft diese Nebenbedingung nicht nur v, sondern auch die gerichteten Kanten $(v, v_i) \in K_{\vec{G}}$ mit $v_i \in E$, die im zu v gehörigen Glied $g_v \in \mathcal{G}$ durch Z_{g_v} repräsentiert werden.

Prüfbarkeit In bezug auf die Möglichkeiten zur Prüfung ab Seite 32 sind für die Prüfbarkeit von C_{g_v} zwei Fälle zu unterscheiden:

1. Unter der Prüfbarkeit von C_{g_v} ist zu verstehen, daß ein Berechtigter aus $\mathcal{B}_p$ die Dechiffriertransformation D_{K_2}, das Element C_{g_v} und das Glied g_v kennt, wodurch er $D_{K_2}(C_{g_v})$ mit g_v vergleichen kann.

2. Die Prüfbarkeit von C_{g_v} besteht darin, daß ein Berechtigter aus $\mathcal{B}_p$ die Dechiffriertransformation D_{K_2}, das Element C_{g_v} und eine Funktion zur Prüfung der Authentizität kennt, wodurch er $D_{K_2}(C_{g_v})$ als g_v ansehen kann, wenn $D_{K_2}(C_{g_v})$ authentisch ist.

Kennt der Berechtigte aus $\mathcal{B}_p$ eine der aufgeführten Voraussetzungen nicht, kann er C_{g_v} nicht prüfen.

Einfach abhängig Allgemein ist unter der einfachen Abhängigkeit der Prüfbarkeit zu verstehen, daß die Prüfung von C_{g_v} nur mittels eines einzelnen Ergebnisses der Prüfung von C_{g_u} erfolgen kann. Zur Vereinfachung wird vereinbart, daß eine einfache Abhängigkeit der Prüfbarkeit entsprechend einer

gerichteten Kante $(u, v) \in K_{\vec{G}}$ beschrieben wird. Das heißt, die Prüfbarkeit von v bzw. C_{g_v} ist von der vorhergehenden Prüfung von u bzw. C_{g_u} abhängig, wenn eine gerichtete Kante $(u, v) \in K_{\vec{G}}$ existiert. Entsprechend den beiden oben genannten Fällen zur Prüfbarkeit gibt es für die einfache Abhängigkeit der Prüfbarkeit zwei übergeordnete Möglichkeiten:

1. D_{K_2} ist nicht wie bei allen $C_g \in C_{\mathcal{G}}$ gleich, sondern D_{K_2} für C_{g_v} geht aus g_u hervor, und

2. g_u enthält das Gegenstück der Bedingung zur Prüfung der Authentizität von $D_{K_2}(C_{g_v})$.

Die erste Möglichkeit beruht auf abhängigen kryptographischen Verkettungen, deren Prinzipien in Abschnitt 3.3.3 dargelegt wurden. Daher wird hier nicht weiter auf diese Möglichkeit eingegangen.

Die zweite Möglichkeit kann dagegen sowohl auf einer abhängigen als auch auf einer unabhängigen kryptographischen Verkettung aufbauen. Dafür wird angenommen, daß das Gegenstück der Bedingung aus der Kennung k_{g_v} besteht, welches wegen der gerichteten Kante (u, v) in Z_{g_u} und somit in g_u enthalten ist. $D_{K_2}(C_{g_v})$ wäre dann authentisch, wenn $D_{K_2}(C_{g_v})$ plausibel ist und eine Kennung aus Z_{g_u} mit der Kennung aus $D_{K_2}(C_{g_v})$ übereinstimmt. Unter bestimmten Umständen kann dabei k_{g_v} auch als die auf Seite 32 beschriebene Zusatzinformation eingesetzt werden.

Für die zweite Möglichkeit ist bei einer abhängigen kryptographischen Verkettung die Kennung k_{g_v} gleich dem Schlüssel K_2, der zum Entschlüsseln von C_{g_v} mittels D_{K_2} benötigt wird. Dies ist wiederum auch für die erste Möglichkeit der Fall. Der Unterschied zwischen beiden Möglichkeiten besteht darin, daß bei der zweiten Möglichkeit die Kennung $k_{g_v} \in Z_{g_u}$ mit der Kennung aus $D_{K_2=k_{g_v}}(C_{g_v})$ verglichen wird und nicht $D_{K_2=k_{g_v}}(C_{g_v})$ mit g_v.

Mehrfach abhängig Analog zur einfachen Abhängigkeit der Prüfbarkeit kann unter einer mehrfachen Abhängigkeit der Prüfbarkeit verstanden werden, daß die Prüfung von C_{g_v} von den Ergebnissen der vorhergehenden Prüfung aller C_{g_u} mit $u \in P_v$ abhängig ist.

Unabhängig Unter der Unabhängigkeit der Prüfbarkeit ist zu verstehen, daß die Prüfbarkeit von C_{g_v} nicht von der vorhergehenden Prüfung eines anderen Elements C_{g_u} abhängig ist. Im Vergleich zur oben beschriebenen Abhängigkeit setzt die Unabhängigkeit voraus, daß $C_{\mathcal{G}}$ auf einer unabhängigen kryptographischen Verkettung beruht. Das heißt, C_{g_v} kann beliebig aus $C_{\mathcal{G}}$ gewählt werden und danach mittels $D_{K_2}(C_{g_v})$ mit g_v verglichen oder auf Authentizität überprüft werden, ohne daß zuvor ein anderes Element C_{g_u} hätte überprüft werden müssen. Erst für die Prüfung der Vollständigkeit müßten alle $C_g \in C_{\mathcal{G}}$ überprüft werden.

Vergleich Die Abhängigkeit bzw. Unabhängigkeit der Prüfbarkeit bestimmt in erster Linie die Möglichkeiten zum Zugriff auf $C_{\mathcal{G}}$. Eine Abhängigkeit der Prüfbarkeit ist nicht für wahlfreien Zugriff auf $C_{\mathcal{G}}$ geeignet. Vielmehr kann hier die Prüfung und somit der Zugriff nur sequentiell erfolgen. Dagegen erlaubt die Unabhängigkeit der Prüfbarkeit sowohl wahlfreien als auch sequentiellen Zugriff auf $C_{\mathcal{G}}$.

Schließlich kann eine Abhängigkeit der Prüfbarkeit sowohl mittels einer abhängigen als auch mit einer unabhängigen kryptographischen Verkettung erreicht werden, während eine Unabhängigkeit der Prüfbarkeit auf einer unabhängigen kryptographischen Verkettung beruhen muß.

3.4.2 Auswertbarkeit

Die nächste Nebenbedingung geht der Frage nach, ob $C_{\mathcal{G}}$ nur prüfbar oder auch auswertbar sein soll.

Nur prüfbar Unter der Formulierung „nur prüfbar“ ist in diesem Zusammenhang zu verstehen, daß ein Berechtigter aus $\mathcal{B}_p$ zwar die Einhaltung der Bedingungen 1 und 2 überprüfen darf, er aber nicht in der Lage sein darf, aus $C_{\mathcal{G}}$ wieder $\vec{G}$ abzuleiten. Dadurch soll es ihm verwehrt sein zu erkennen, welchen Inhalt die Ecken aus E haben und/oder wie sie über welche Kanten aus $K_{\vec{G}}$ zusammenhängen.

Es wurde allerdings vereinbart (s. Seite 31), daß ein Berechtigter aus $\mathcal{B}_p$ die Dechiffriertransformation D_{K_2} kennt, wodurch er $\mathcal{G}$ mittels

$$\mathcal{G} = \{g \mid g = D_{K_2}(C_g),\ C_g \in C_{\mathcal{G}}\} \subseteq \mathcal{M}$$

wiederherstellen kann (s. „Anwendung auf alle Glieder“ auf Seite 32). Da sich aus $\mathcal{G}$ problemlos wieder $\vec{G}$ ableiten läßt, folgt daraus, daß die Geheimhaltung des Inhalts der Ecken aus E und/oder der Kanten aus $K_{\vec{G}}$ bei dieser Nebenbedingung nicht von der kryptographischen Verkettung abhängen darf (s. Erläuterung zu den Zielen der kryptographischen Verkettung auf Seite 26). Vielmehr muß eine weitere Ebene zur Geheimhaltung eingefügt werden.

Die zusätzliche Ebene zur Geheimhaltung wird hier beispielsweise zwischen $\mathcal{G}$ und $C_{\mathcal{G}}$ eingfügt. Dafür seien

$$KS_v = (\mathcal{M}_v, \mathcal{C}_v, \mathcal{K}_v, E_{v_{K_1}}, D_{v_{K_2}})$$

und

$$KS_Z = (\mathcal{M}_Z, \mathcal{C}_Z, \mathcal{K}_Z, E_{Z_{K_1}}, D_{Z_{K_2}})$$

mit

$$\mathcal{M}_v = \mathcal{C}_v = \mathcal{M}_Z = \mathcal{C}_Z = \mathcal{M}$$

zwei Kryptosysteme, die die Geheimhaltungsanforderungen von Seite 14 erfüllen. Aus der Menge von Gliedern $\mathcal{G}$ wird nun zunächst eine Menge von Gliedern

$$\mathcal{G}_{vZ} = \{g_{vZ} \mid g_{vZ} = (k_g, E_{Z_{K_1}}(Z_g), E_{v_{K_1}}(v)),\ (k_g, Z_g, v) = g \in \mathcal{G}\} \subseteq \mathcal{M}$$

abgeleitet und dann daraus

$$C_{\mathcal{G}_{vZ}} = \{C_{g_{vZ}} \mid C_{g_{vZ}} = E_{K_1}(g_{vZ}),\ g_{vZ} \in \mathcal{G}_{vZ}\} \subseteq \mathcal{C}$$

erstellt. Durch die getrennte Verschlüsselung von Z_g und v kann die Geheimhaltung flexibel gehandhabt werden:

1. Ecken geheim und Kanten nicht geheim $\Longleftrightarrow$ Berechtigte aus $\mathcal{B}_p$ kennen $D_{v_{K_2}}$ nicht, aber sie kennen $D_{Z_{K_2}}$,

2. Ecken nicht geheim und Kanten geheim $\Longleftrightarrow$ Berechtigte aus $\mathcal{B}_p$ kennen $D_{v_{K_2}}$, aber sie kennen $D_{Z_{K_2}}$ nicht,

3. Ecken und Kanten getrennt geheim $\Longleftrightarrow$ Berechtigte aus $\mathcal{B}_p$ kennen weder $D_{v_{K_2}}$ noch $D_{Z_{K_2}}$ mit $D_{v_{K_2}} \neq D_{Z_{K_2}}$ und

4. Ecken und Kanten gemeinsam geheim $\Longleftrightarrow$ Berechtigte aus $\mathcal{B}_p$ kennen weder $D_{v_{K_2}}$ noch $D_{Z_{K_2}}$ mit $D_{v_{K_2}} = D_{Z_{K_2}}$.

Im 1. Fall kann ein Berechtigter aus $\mathcal{B}_p$ demnach erkennen, welche Ecken über welche Kanten miteinander verbunden sind. Er kann aber nicht den Inhalt der Ecken lesen. Im 2. Fall ist es genau anders herum. Ein Berechtigter aus $\mathcal{B}_p$ kann zwar den Inhalt der Ecken lesen, er weiß aber nicht, über welche Kanten die Ecken miteinander verbunden sind.

Die Prüfung von $C_{\mathcal{G}_{vZ}}$ muß nun allerdings auch die Ebene zur Geheimhaltung berücksichtigen. Dafür wird als erstes $C_{\mathcal{G}_{vZ}}$ mittels

$$\mathcal{G}_{vZ} = \{g_{vZ} \mid g_{vZ} = D_{K_2}(C_{g_{vZ}}),\ C_{g_{vZ}} \in C_{\mathcal{G}_{vZ}}\} \subseteq \mathcal{M}$$

wieder in $\mathcal{G}_{vZ}$ überführt. Die Prüfung der Plausibilität aller $g_{vZ} \in \mathcal{G}_{vZ}$ kann genauso erfolgen wie bei den $g \in \mathcal{G}$ auf Seite 33. Die Prüfung der Vollständigkeit anhand der Anforderung, daß $\vec{G}$ stark zusammenhängend ist (s. Def. 3.3.2), ist in den oben genannten Fällen 2, 3 und 4 nicht möglich, weil die Kanten aus $K_{\vec{G}}$ gegenüber den Berechtigten aus $\mathcal{B}_p$ geheim sind. Dafür würde sich zum Beispiel ein Zähler anbieten (s. Seite 33). Für den 1. Fall könnte allerdings die Definition 3.3.2 herangezogen werden, weil die Prüfung, ob $\vec{G}$ stark zusammenhängend ist oder nicht, nicht vom Inhalt der Ecken aus E abhängig ist.

Auch auswertbar Ist $C_{\mathcal{G}}$ dagegen „auch auswertbar“, darf ein Berechtigter aus $\mathcal{B}_p$ zur Prüfung der Einhaltung der Bedingungen 1 und 2 aus $C_{\mathcal{G}}$ wieder $\vec{G}$ ableiten. Das heißt, daß die Auswertbarkeit des Inhalts der Ecken aus E und der Kanten aus $K_{\vec{G}}$ von der kryptographischen Verkettung abhängig ist.

3.4.3 Abgeschlossenheit und Änderbarkeit

Eine weitere Nebenbedingung besteht darin, ob $C_{\mathcal{G}}$ abgeschlossen oder änderbar sein soll.

Abgeschlossenheit Die Abgeschlossenheit von $C_{\mathcal{G}}$ soll bedeuten, daß es für alle Individuen aus $\mathcal{I}$ berechnungsmäßig praktisch unmöglich ist, unbemerkt (s. Seite 27) eine Änderung an $C_{\mathcal{G}}$ vorzunehmen. Das heißt, in diesem Fall sind $\mathcal{B}_m = \emptyset$ und $\mathcal{B}_e = \emptyset$ und entsprechend $\mathcal{U}_m = \mathcal{I}$ und $\mathcal{U}_e = \mathcal{I}$. Die Einhaltung dieser Abgeschlossenheitsbedingung kann zum Beispiel auf

1. der Geheimhaltung von E_{K_1} oder

2. Zeugen der Abgeschlossenheit

beruhen.

Aus der Annahme, daß die Einhaltung der Abgeschlossenheitsbedingung von $C_{\mathcal{G}}$ auf der Geheimhaltung von E_{K_1} beruhen soll, folgt, daß kein Unberechtigter aus $\mathcal{U}_m$ oder $\mathcal{U}_e$ die Chiffriertransformation E_{K_1} kennen darf. Da $\mathcal{U}_m = \mathcal{I}$ und $\mathcal{U}_e = \mathcal{I}$ gelten, folgt weiter, daß kein Individuum aus $\mathcal{I}$ die Chiffriertransformation E_{K_1} kennen darf, obwohl sie für die Erstellung von $C_{\mathcal{G}}$ benötigt wird. Dieses Problem kann zum Beispiel dadurch gelöst werden, daß ein Rechner

1. zufällig den Schlüssel K_1 bestimmt,
2. $C_{\mathcal{G}}$ erstellt und
3. K_1 unwiederbringlich löscht,

ohne daß ein Individuum aus $\mathcal{I}$ während dieses Vorgangs von K_1 Kenntnis erlangt. Die Anforderungen „zufällig“, „unwiderbringlich“ und „ohne Kenntnis zu erlangen“ sind allerdings recht intuitiv. Die Einhaltung der Abgeschlossenheitsbedingung auf der Geheimhaltung von E_{K_1} beruhen zu lassen, scheint daher weniger für die Praxis geeignet zu sein, als die nächste Vorgehensweise.

Die Einhaltung der Abgeschlossenheitsbedingung von $C_{\mathcal{G}}$ kann zum Beispiel auch auf Zeugen der Abgeschlossenheit beruhen, wobei es nicht erforderlich sein muß, daß E_{K_1} geheim ist. Dafür kann als erstes unterschieden werden, ob $C_{\mathcal{G}}$

1. von vornherein abgeschlossen sein soll oder
2. erst nach beliebig vielen Erweiterungen ($C_{\mathcal{G}} \longmapsto C_{\mathcal{G}} \cup \{C_g\}$) abgeschlossen wird.

Hierbei ist wiederum bemerkenswert, daß ein Verfahren, das in der Lage ist, $C_{\mathcal{G}}$ erst nach beliebig vielen Erweiterungen abzuschließen, auch dafür genutzt werden kann, $C_{\mathcal{G}}$ von vornherein abzuschließen. Daher wird an dieser Stelle kein Beispiel für ein Verfahren angegeben, das $C_{\mathcal{G}}$ von vornherein abschließt. Bei einem Verfahren, das $C_{\mathcal{G}}$ nach beliebig vielen Erweiterungen abschließt, steht allerdings vor allem die Erweiterbarkeit von $C_{\mathcal{G}}$ im Vordergrund, weshalb in den nächsten Abschnitten unter anderem auf diese eingegangen wird. Ein Zeuge der Abgeschlossenheit könnte dann der Hashwert $h_e(C_{\mathcal{G}})$ einer Einweg-Hashfunktion h_e sein, der an alle Individuen aus $\mathcal{I}$ übermittelt wird oder zum Beispiel zumindest in einer Zeitung veröffentlicht wird. Eine andere Möglichkeit bestünde darin, $h_e(C_{\mathcal{G}})$ mit einem Zeitstempel zu versehen, der selbst wieder auf einer kryptographischen Verkettung beruht (s. a. [HS91]).

Änderbarkeit Die Änderbarkeit von $C_{\mathcal{G}}$ hat vielerlei Bedeutungen. Dabei gibt es zwei Hauptrichtungen zu verfolgen:

1. Zum einen $C_g \in C_{\mathcal{G}}$ zu ändern und
2. zum anderen neue $C_g \in \mathcal{C}$ an $C_{\mathcal{G}} \setminus \{C_g\}$ anzufügen ($C_{\mathcal{G}} \longmapsto C_{\mathcal{G}} \cup \{C_g\}$) oder $C_g \in C_{\mathcal{G}}$ aus $C_{\mathcal{G}}$ zu entfernen ($C_{\mathcal{G}} \longmapsto C_{\mathcal{G}} \setminus \{C_g\}$).

Ein $C_g \in C_{\mathcal{G}}$ zu ändern, kann dadurch motiviert werden, daß im zugehörigen gerichteten Graphen $\vec{G}$

1. eine gerichtete Kante (u, v) mit $u, v \in E$ und $(u, v) \notin K_{\vec{G}}$ in $K_{\vec{G}}$ eingefügt werden soll ($K_{\vec{G}} \longmapsto K_{\vec{G}} \cup \{(u, v)\}$),

2. eine gerichtete Kante $(u, v) \in K_{\vec{G}}$ aus $K_{\vec{G}}$ entfernt werden soll ($K_{\vec{G}} \longmapsto K_{\vec{G}} \setminus \{(u, v)\}$) oder

3. eine Ecke $v \in E$ geändert werden soll.

Das Einfügen oder Entfernen von (u, v) würde dabei jeweils nur Z_{g_u} des Gliedes $g_u \in \mathcal{G}$ der Ecke u und somit nur $C_{g_u} \in C_{\mathcal{G}}$ betreffen. Beim Entfernen von (u, v) muß berücksichtigt werden, daß dadurch eventuell der einzige Zeuge der Existenz von v entfernt wird, wenn $P_v = \{u\}$ ist. Das Ändern von v würde auf das Glied $g_v \in \mathcal{G}$ und somit auf $C_{g_v} \in C_{\mathcal{G}}$ beschränkt sein.

Durch das Anfügen eines C_{g_v} an $C_{\mathcal{G}}$ kann eine neue Ecke $v \notin E$ in E eingefügt werden ($E \longmapsto E \cup \{v\}$). Dementsprechend kann durch das Entfernen eines C_{g_v} aus $C_{\mathcal{G}}$ eine Ecke $v \in E$ aus E entfernt werden ($E \longmapsto E \setminus \{v\}$). In beiden Fällen müssen die Zeugen der Existenz von v entsprechend berücksichtigt werden. Für den Zähler $a_{\mathcal{G}}$ (s. Seite 33) bedeutet das, daß er um eins erhöht werden muß ($a_{\mathcal{G}} = a_{\mathcal{G}} + 1$) oder um eins reduziert werden muß ($a_{\mathcal{G}} = a_{\mathcal{G}} - 1$). Komplexer erscheint die Situation, wenn die Zeugen der Existenz von v auf gerichteten Kanten (u_i, v) mit $u_i \in P_v$ beruhen:

1. Anfügen von C_{g_v} an $C_{\mathcal{G}}$:
   Unter der Annahme $(u_i, v) \notin K_{\vec{G}}$ muß in allen $Z_{g_{u_i}}$ die Kennung k_{g_v} eingetragen werden ($Z_{g_{u_i}} \longmapsto Z_{g_{u_i}} \cup \{k_{g_v}\}$), wodurch $C_{g_{u_i}}$ geändert werden muß.

   Die Änderung von $C_{g_{u_i}}$ kann aber vermieden werden, wenn (u_i, v) bereits bei der Erstellung von g_{u_i} in $K_{\vec{G}}$ vorhanden ist ($(u_i, v) \in K_{\vec{G}}$). In diesem Fall wird nicht angenommen, daß (u_i, v) fehlerhaft ist, weil (u_i, v) auf eine nicht vorhandene Ecke v zeigt, sondern es wird angenommen, daß (u_i, v) auf eine Ecke v zeigt, die in der Zukunft in E vorhanden sein wird. Dementsprechend braucht bei der Erstellung von g_{u_i} für v zunächst nur die Kennung k_{g_v} zu existieren, weil ja k_{g_v} das einzige des Gliedes g_v der Ecke v ist, das in $Z_{g_{u_i}}$ eingetragen wird. g_v und v brauchen deshalb zu diesem Zeitpunkt noch nicht zu existieren.

   Zu einem späteren Zeitpunkt kann dann eine Ecke v in E eingefügt werden, indem ihr die Kennung k_{g_v} zugeordnet wird und g_v sowie C_{g_v} erstellt werden. Dabei gilt laut Definition 3.3.1, daß g_v bereits bei der Erstellung mit g_{u_i} einfach kryptographisch verkettet ist.

   In bezug auf Definition 3.3.2 und Definition 3.3.4 muß in beiden Fällen beachtet werden, daß $\vec{G}$ auch nach dem Einfügen von v stark zusammenhängend ist. Dies kann zum Beispiel durch die gerichteten Kanten (v, u_i) erreicht werden.

2. Entfernen von C_{g_v} aus $C_{\mathcal{G}}$:
   Dadurch wird nicht nur v entfernt, sondern auch alle gerichteten Kanten $(v, w_j) \in K_{\vec{G}}$ mit $w_j \in S_v$, da sie in g_v durch Z_{g_v} repräsentiert werden. Für alle $(u_i, v) \in K_{\vec{G}}$ gilt dann, daß sie auf eine Ecke $v \notin E$ zeigen.

Unter der Annahme, daß auch ohne $(u_i, v) \in K_{\vec{G}}$ und ohne $(v, w_j) \in K_{\vec{G}}$ von allen u_i jeweils ein gerichteter Kantenzug zu allen w_j existiert, reicht es aus, wenn k_{g_v} aus allen $Z_{g_{u_i}}$ entfernt wird ($Z_{g_{u_i}} \longmapsto Z_{g_{u_i}} \setminus \{k_{g_v}\}$), wodurch entsprechend die $C_{g_{u_i}} \in C_{\mathcal{G}}$ geändert werden müssen.

Gilt die Annahme nicht, können beispielsweise alle $k_{g_{w_j}} \in Z_{g_v} \setminus \{k_{g_{u_i}}\}$ jeweils in allen $Z_{g_{u_i}}$ eingetragen werden, falls $k_{g_{w_j}} \notin Z_{g_{u_i}}$ gilt:

$$Z_{g_{u_i}} \longmapsto Z_{g_{u_i}} \cup \{k_{g_{w_j}} \mid k_{g_{w_j}} \in Z_{g_v} \setminus \{k_{g_{u_i}}\} \wedge k_{g_{w_j}} \notin Z_{g_{u_i}}\}.$$

3.4.4 Reichweite von Berechtigungen

Auf die Reichweite von Berechtigungen wird bereits in Abschnitt 3.3.3 eingegangen. Allgemein kann die Nebenbedingung, ob Berechtigungen nur für einen Teil des Graphen oder für den gesamten Graphen gelten sollen, als Erreichbarkeitsproblem aufgefaßt werden. Das heißt, daß sich Berechtigungen auf all die Teile des Graphen erstrecken, die von einem Teil des Graphen aus erreichbar sind, für den eine bestimmte Berechtigung gilt.

3.4.5 Berechtigte und Unberechtigte

Die letzte aufgeführte Nebenbedingung bezieht sich auf die Frage, ob die Bezeichnungen „Unberechtigter“ und „Berechtigter“ jeweils für einen einzelnen, eine Gruppe oder alle Individuen gelten sollen. Diese Nebenbedingung wird im Prinzip durch die Definitionen 3.1.1 und 3.1.2 geklärt, indem immer von einer bestimmten Menge von Individuen ausgegangen wird, die eine bestimmte Mächtigkeit hat.

Kapitel 4

Sichere Log-Datei

Auf Grundlage der Theorie aus dem vorhergehenden Kapitel 3 stelle ich hier ein spezielles Schema zur kryptographischen Verkettung eines gerichteten Hamiltonschen Weges vor, das in bezug auf die kryptographische Verkettung gegenüber [SK98] weiter verallgemeinert ist. Aus diesem Schema leite ich dann verschiedene Möglichkeiten zur Lösung der in der Einleitung auf Seite 2 beschriebene Aufgabenstellung für eine sichere Log-Datei ab. Die einzelnen Möglichkeiten werden entsprechend den Darstellungen über Kryptographie in Abschnitt 1.6 nach

- symmetrischen Kryptosystemen und
- Public-Key-Kryptosystemen

und der Kombination mit

- einer Einweg-Hashfunktion,
- einer Einwegfunktion und
- einem Zufallsfolgengenerator

geordnet, wobei für Public-Key-Kryptosysteme zusätzlich ein einhüllendes Prinzip vorgestellt wird.

Als erstes beschreibt Abschnitt 4.1 noch einmal genauer das Szenario, das zu der Aufgabenstellung geführt hat. Danach wird unter Berücksichtigung des Szenarios in Abschnitt 4.2 das Schema zur Lösung der Aufgabenstellung im Hinblick auf eine Implementierung vorgestellt, während in den Abschnitten 4.3 und 4.4 die Realisierung des Schemas mittels symmetrischen Kryptosystemen und Public-Key-Kryptosystemen behandelt wird. Abschließend listet Abschnitt 4.5 Alternativen zur Sicherung der Einträge von Log-Dateien auf, die nicht auf kryptographischen Verkettungen beruhen.

4.1 Szenario

Die folgenden Ausführungen beziehen sich auf das in der Einleitung auf Seite 1 beschriebene Szenario für eine Log-Datei $\mathcal{L}$. Dafür sei $\mathcal{R}$ der Rechner, in dem $\mathcal{L}$

erstellt wird und mittels einer kryptographischen Verkettung gesichert werden soll.

Weiter sei t_E der Zeitpunkt eines Einbruchs in $\mathcal{R}$ durch einen Eindringling aus $\mathcal{U}$. Dadurch kann unterschieden werden, daß $\mathcal{R}$

1. für $t \leq t_E$ vertrauenswürdig sein soll und
2. für $t > t_E$ nicht vertrauenswürdig sein kann.

Das heißt, daß es nur an Zeitpunkten $t \leq t_E$ überhaupt möglich ist, einen authentischen Eintrag e_{n+1} an $\mathcal{L}$ anzuhängen und mittels einer kryptographischen Verkettung zu sichern. Alle Einträge e_{n+1}, die an Zeitpunkten $t > t_E$ an $\mathcal{L}$ angehängt werden, können trotz einer kryptographischen Verkettung nicht als authentisch angesehen werden (s. a. [SK98]).

Für Einträge e_{n+1}, die an Zeitpunkten $t \leq t_E$ an $\mathcal{L}$ angehängt wurden, gilt daher

$$\mathcal{U}_m = \mathcal{U}_e = \mathcal{U} \quad \text{und} \quad \mathcal{B}_m = \mathcal{B}_e = \mathcal{B}.$$

Außerdem ist bis auf eine Ausnahme in Abschnitt 4.4.3

$$\mathcal{U}_p = \mathcal{U} \quad \text{und} \quad \mathcal{B}_p = \mathcal{B}.$$

4.2 Schema

Wie schon in der Einleitung auf Seite 2 angedeutet, hat der für $\mathcal{L}$ auf Seite 25 definierte gerichtete Graph $\vec{G} = (E, K_{\vec{G}})$ einen gerichteten Hamiltonschen Weg

$$W = (v_1 v_2, ..., v_{n-1} v_n),$$

der mit $v_1 = e_1$ beginnt. Das heißt,

$$E = \bigcup_{i=1}^{n} v_i \quad \text{mit} \quad v_i \neq v_j \quad \text{für} \quad i \neq j,\ i, j = 1, ..., n$$

und

$$K_{\vec{G}} = \bigcup_{i=1}^{n-1} v_i v_{i+1} \quad \text{mit} \quad v_i v_{i+1} \neq v_j v_{j+1} \quad \text{für} \quad i \neq j,\ i, j = 1, ..., n-1.$$

Die hier vorgestellten Möglichkeiten zur kryptographischen Verkettung von $\vec{G}$ beruhen auf dem Prinzip, daß die gerichtete Kante $v_{i-1} v_i$ mit $i = 1, ..., n$ Zeuge der Existenz der Ecke v_i ist. Speziell $v_0 v_1 \notin K_{\vec{G}}$ ist dabei außerhalb von $\vec{G}$ ein Zeuge der Existenz von v_1 und wird zur Überprüfung der kryptographischen Verkettung von $\vec{G}$ benötigt (s. a. „Wurzeln, Quellen und Senken“ auf Seite 36 und Abschnitt 3.4.1).

Menge von Gliedern Desweiteren wird $\vec{G}$ durch eine Menge von Gliedern

$$\mathcal{G} = \bigcup_{i=1}^{n} g_i \quad \text{mit} \quad g_i = (k_{g_i}, Z_{g_i}, v_i) \quad \text{und} \quad Z_{g_i} = \{k_{g_{i+1}}\}$$

repräsentiert. Das heißt, daß in Z_{g_n} die Kennung $k_{g_{n+1}}$ eines Gliedes g_{n+1} steht, das nicht Element von $\mathcal{G}$ ist. Dadurch läßt sich die kryptographische Verkettung von $\vec{G}$ erweitern (s. a. Abschnitt 3.4.3), was durch die Nebenbedingung motiviert wird (s. Seite 2), daß jeweils eine neue Ecke v_{n+1} mittels der gerichteten Kante $v_n v_{n+1}$ an W angehängt werden kann.

Zeitleiste Die Zustände von $\vec{G}$ lassen sich auf einer diskreten Zeitleiste auftragen, wobei $\vec{G}$

1. zum Zeitpunkt t_0 aus $E = \emptyset$ und $K_{\vec{G}} = \emptyset$ besteht und $v_0 v_1$ außerhalb von $\vec{G}$ als Zeuge der Existenz von v_1 existiert,

2. zum Zeitpunkt t_1 aus $E = \{v_1\}$ und $K_{\vec{G}} = \emptyset$ besteht und W die Länge 0 hat und

3. zu einem Zeitpunkt $t_{n>1}$ wie oben beschrieben ist und W die Länge $n-1$ hat.

Implementierung Die Implementierung der Erzeugung und Prüfung der kryptographischen Verkettung von $\vec{G}$ benötigt bei den im weiteren Verlauf vorgestellten Möglichkeiten drei Funktionen. Diese Funktionen werden nachfolgend beschrieben und in den Abbildungen 4.1, 4.2 und 4.3 schematisch dargestellt. Bei der dadurch erzeugten kryptographischen Verkettung

$$(C_{\mathcal{G}}, k_{g_1})$$

von $\vec{G}$ handelt es sich um eine kryptographische Verkettung mit Wurzel v_1, die sowohl unabhängig als auch abhängig sein kann (s. Definition 3.3.3 und Definition 3.3.5).

Initialisierungsfunktion Vor dem Zeitpunkt t_0 erzeugt eine Initialisierungsfunktion mittels $E_{\mathcal{Z}}$ die Kennung k_{g_1} für die später zu sichernde Wurzel v_1 (s. a. Abb. 4.1). Die Kennung k_{g_1} muß sowohl temporär in $\mathcal{R}$ als auch außerhalb von $\mathcal{R}$ gespeichert werden. In $\mathcal{R}$ selbst wird k_{g_1} für den Zeitpunkt t_1 von der nachfolgend beschriebenen Verkettungsfunktion die Wurzel v_1 zugeordnet und nach der Erstellung von

$$g_1 = (k_{g_1}, \{k_{g_2}\}, v_1)$$

und

$$C_{g_1} = E_{K_1}(g_1)$$

unwiederbringlich von der Kennung k_{g_2} überschrieben. Außerhalb von $\mathcal{R}$ steht k_{g_1} für die gerichtete Kante $v_0 v_1 \notin K_{\vec{G}}$, die der Zeuge der Existenz von v_1 ist, und wird ab dem Zeitpunkt t_1 erst wieder für die weiter unten beschriebene Prüffunktion benötigt.

Außerdem gilt für die kryptographische Verkettung von $\vec{G}$ zum Zeitpunkt t_0 wegen $\vec{G} = (\emptyset, \emptyset)$, daß

$$C_{\mathcal{G}} = \emptyset$$

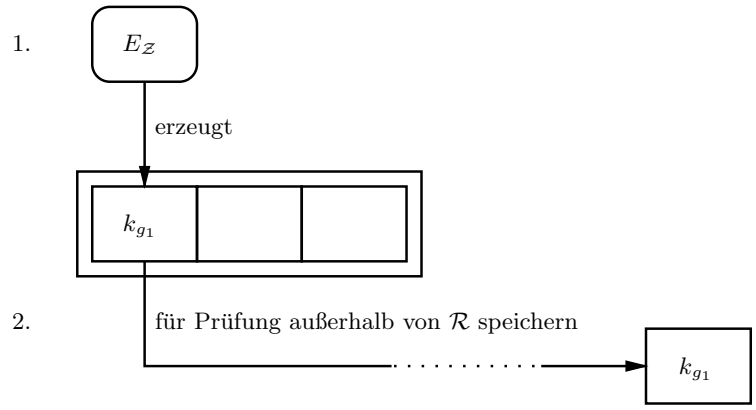

Abbildung 4.1: Initialisierung der kryptographischen Verkettung

ist.

Verkettungsfunktion Nach einem Zeitpunkt $t_{n\geq 0}$ kann für einen Zeitpunkt t_{n+1} mittels einer Verkettungsfunktion eine neue Ecke v_{n+1} anhand der gerichteten Kante $v_n v_{n+1}$ an W angehängt werden (mit dem Sonderfall, daß $v_0 v_1 \notin K_{\vec{G}}$ ist) bzw. $C_{\mathcal{G}}$ um ein neues Element $C_{g_{n+1}}$ erweitert werden (s. a. Abb. 4.2 und Abschnitt 3.4.3):

$$C_{\mathcal{G}} \longmapsto C_{\mathcal{G}} \cup \{C_{g_{n+1}}\}.$$

Dafür erzeugt die Verkettungsfunktion mittels $E_{\mathcal{Z}}$ als erstes die Kennung $k_{g_{n+2}}$ für ein nachfolgendes Glied g_{n+2}. Damit ist

$$g_{n+1} = (k_{g_{n+1}}, \{k_{g_{n+2}}\}, v_{n+1})$$

und

$$C_{g_{n+1}} = E_{K_1}(g_{n+1}),$$

wobei noch nicht genauer spezifiziert werden soll, wie die Chiffriertransformation E_{K_1} zustande kommt. Danach erweitert die Verkettungsfunktion $C_{\mathcal{G}}$ um $C_{g_{n+1}}$ und überschreibt $k_{g_{n+1}}$ unwiederbringlich mit $k_{g_{n+2}}$ für die nächste Verkettung. Um eine Geheimhaltung von v_{n+1} sicherzustellen, muß dafür Sorge getragen werden, daß v_{n+1} nach dem Erstellen von $C_{g_{n+1}}$ unwiederbringlich gelöscht wird.

Prüffunktion Die Überprüfung von $(C_{\mathcal{G}}, k_{g_1})$ beruht auf einer einfachen Abhängigkeit der Prüfbarkeit (s. a. Abb. 4.3 sowie „Wurzeln, Quellen und Senken“ auf Seite 36 und Abschnitt 3.4.1). Genauer formuliert, kann ein Berechtigter aus $\mathcal{B}_p$ die Einhaltung der Bedingungen 1 und 2 ab der Ecke v_i für $i = 1, ..., n$ überprüfen, wenn er die Kennung k_{g_i} (typischerweise k_{g_1}) und die zum Entschlüsseln von $C_{g_i} \in C_{\mathcal{G}}$ benötigte Dechiffriertransformation D_{K_2} kennt. Dafür wird angenommen, daß $D_{K_2}(C_{g_i})$ authentisch ist, wenn $D_{K_2}(C_{g_i})$ plausibel ist und die Kennung k_{g_i} mit der Kennung aus $D_{K_2}(C_{g_i})$ übereinstimmt. Entsprechend ist $D_{K_2}(C_{g_i})$ nicht authentisch, wenn k_{g_i} und die Kennung aus $D_{K_2}(C_{g_i})$ nicht übereinstimmen.

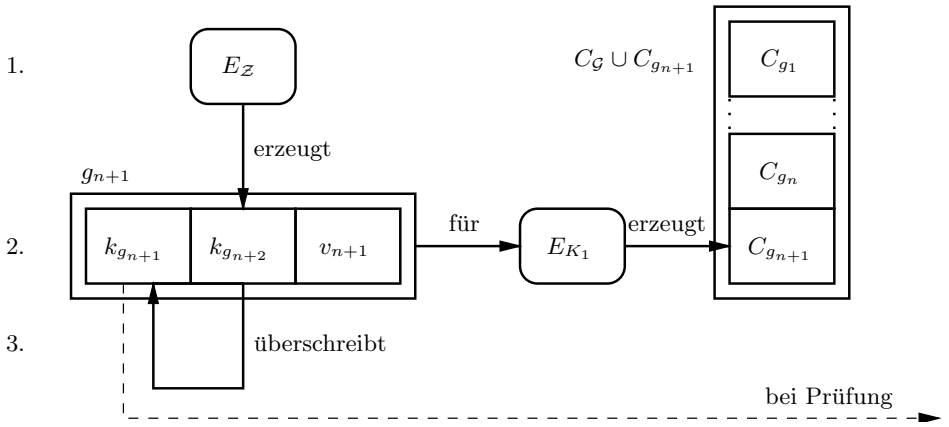

Abbildung 4.2: Erzeugung der kryptographischen Verkettung

Sollte nun $D_{K_2}(C_{g_i})$ authentisch sein, kann rekursiv das nächste Element $C_{g_{i+1}} \in C_{\mathcal{G}}$ bis einschließlich $i+1=n$ mittels der Kennung $k_{g_{i+1}} \in Z_{g_i}$ und der entsprechenden Dechiffriertransformation D_{K_2} überprüft werden. Die Überprüfung entdeckt weder einen Fehler noch einen Einbruch, wenn alle $D_{K_2}(C_{g_j})$ für $j=i,...,n$ authentisch sind und $k_{g_{n+1}} \in Z_{g_n}$ aus $D_{K_2}(C_{g_n})$ mit der Kennung $k_{g_{n+1}}$ für die nächste Verkettung übereinstimmt.

Sollte dagegen $D_{K_2}(C_{g_i})$ nicht authentisch sein, wird die Überprüfung von $C_{\mathcal{G}}$ abgebrochen. Das bedeutet, daß dadurch entweder ein Fehler oder ein Einbruch entdeckt wird. Zur genaueren Eingrenzung der Ursache müssen dann weitergehende Maßnahmen ergriffen werden (s. a. „Grenzen und Einordnung“ auf Seite 2 und Beispiel 3.3.1).

Anders formuliert überprüft die Prüffunktion für jeweils zwei Elemente $C_{g_i}, C_{g_{i+1}} \in C_{\mathcal{G}}$ mit $i=1,...,n-1$ und v_i, v_{i+1} aus $D_{K_2}(C_{g_i})$ sowie $D_{K_2}(C_{g_{i+1}})$, ob $v_i \in P_{v_{i+1}}$ und $v_{i+1} \in S_{v_i}$ ist (s. a. Def. 1.4.13). Allgemein wird daher ein Fehler oder ein Einbruch entdeckt, wenn v_n nicht von v_0 aus erreichbar ist.

Sicherheit Es muß angenommen werden, daß ein Eindringling aus $\mathcal{U}$ die Chiffriertransformation E_{K_1} zum Verschlüsseln von g_{n+1} kennt, da sie für die Verkettungsfunktion in $\mathcal{R}$ vorhanden sein muß. Dadurch kann er an Zeitpunkten $t > t_E > t_n$ mittels der Verkettungsfunktion beliebige Ecken v_{n+1} an W bzw. beliebige Einträge e_{n+1} an $\mathcal{L}$ anhängen (s. a. Abschnitt 4.1 und [SK98]).

Das Überschreiben von k_{g_i} für $i=1,...,n$ ist ein erster Schritt, einem Eindringling aus $\mathcal{U}$ zu verwehren, zu einem Zeitpunkt $t > t_E > t_n$ unbemerkt ein Element $C_{g'_i}$ für $C_{g_i} \in C_{\mathcal{G}}$ einzusetzen (s. a. [SK98]). Denn dafür muß er die Kennungen k_{g_i} und $k_{g_{i+1}}$ kennen, damit bei einer Überprüfung von $C_{g'_i}$ und $C_{g_{i+1}} \in C_{\mathcal{G}}$ die Kennung $k_{g_i} \in Z_{g_{i-1}}$ mit der Kennung aus $D_{K_2}(C_{g'_i})$ übereinstimmt bzw. die Kennung $k_{g'_{i+1}} \in Z_{g'_i}$ aus $D_{K_2}(C_{g'_i})$ mit der Kennung aus $D_{K_2}(C_{g_{i+1}})$. Daher muß gelten, daß

1. er keine $C_{g_i} \in C_{\mathcal{G}}$ entschlüsselen kann und
2. es für ihn auch sonst berechnungsmäßig praktisch unmöglich ist, eine Kennung k_{g_i} zu bestimmen.

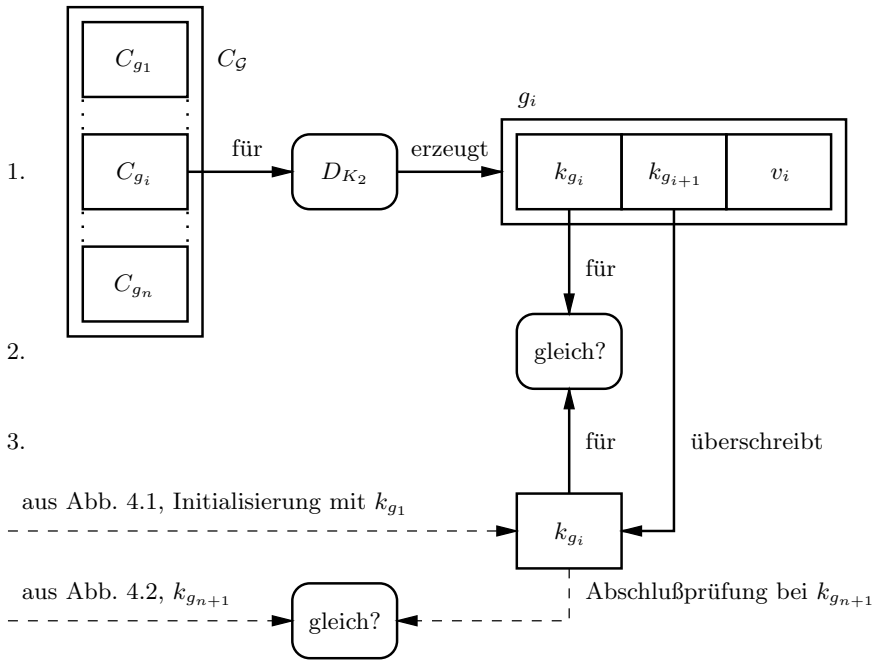

Abbildung 4.3: Prüfung der kryptographischen Verkettung

Das heißt, daß k_{g_i} unter diesen Umständen eine private Zusatzinformation sein muß (s. Seite 32).

Um sicherzustellen, daß die Prüffunktion und der Pfad der Daten von der Eingabe (z. B. Festplatte, Tastatur) zur Ausgabe (z. B. Festplatte, Bildschirm, Drucker) nicht manipuliert sind, sollte die Überprüfung von $(C_{\mathcal{G}}, k_{g_1})$ nur in einem vertrauenswürdigen Rechner stattfinden und nicht in $\mathcal{R}$.

Blockchiffrierung An dieser Stelle wird noch kurz auf den zu wählenden Blockchiffriermodus eingegangen, wenn E_{K_1} auf einer Blockchiffrierung beruht (s. a. Kapitel 9 in [Schn96]).

Dafür wird angenommen, daß E_{K_1} das Glied g_{n+1} in m Blöcke $b_1, ..., b_m$ zerlegt und jeden Block b_i für $i = 1, ..., m$ einzeln verschlüsselt:

$$C_{g_{n+1}} = E_{K_1}(g_{n+1}) = E_{K_1}(b_1...b_m) = C_{b_1}...C_{b_m}.$$

Wird weiter angenommen, daß die private Zusatzinformation $k_{g_{n+1}}$ die Blöcke $b_1, ..., b_j$ mit $1 \leq j < m$ belegt, dann soll b_i für $i = 2, ..., m$, anders als beim *Cipher Block Chaining*[1], vor der Verschlüsselung mit b_{i-1} XOR-verknüpft werden, um ein Austauschen der Blöcke $C_{b_{j+1}}, ..., C_{b_m}$ zu verhindern.

Diese Änderung ist wichtig, weil sonst ein Eindringling aus $\mathcal{U}$ die verschlüsselten Blöcke $C_{b_1}, ..., C_{b_j}$ unangetastet lassen und $C_{b_{j+1}}, ..., C_{b_m}$ ersetzen könnte, da er E_{K_1} und C_{b_j} zum Verschlüsseln von b_{j+1} kennen würde. Dies wird ihm durch die Änderung verwehrt, da $k_{g_{n+1}}$ und somit b_j überschrieben werden, und er C_{b_j} nicht entschlüsseln können darf.

[1] Ein zu verschlüsselnder Block b_i wird vor dessen Verschlüsselung mit dem Ergebnis der Verschlüsselung von b_{i-1} XOR-verknüpft.

Um dann aus $C_{g_{n+1}}$ wieder g_{n+1} zu erhalten, muß das Ergebnis der Entschlüsselung von C_{b_i} für $i = 2, ..., m$ wieder mit dem Ergebnis der Entschlüsselung von $C_{b_{i-1}}$ XOR-verknüpft werden.

4.3 Symmetrische Kryptosysteme

Dieser Abschnitt beschreibt die Realisierung des Schemas aus dem vorhergehenden Abschnitt 4.2 auf Grundlage eines symmetrischen Kryptosystems

$$KS = (\mathcal{M}, \mathcal{C}, \mathcal{K}, E_{K_i}, D_{K_i}).$$

Es wird angenommen, daß die Sicherheit von KS einzig auf der Geheimhaltung von K_i beruht und die Algorithmen E und D einem potentiellen Eindringling aus $\mathcal{U}$ bekannt sind. Daraus folgt wiederum, daß ein Eindringling aus $\mathcal{U}$ bei einem Einbruch in $\mathcal{R}$ die Transformation $E_{K_{n+1}}$ zum Verschlüsseln von g_{n+1} erfährt und daraus $D_{K_{n+1}}$ zum Entschlüsseln von $C_{g_{n+1}}$ ableiten kann. Um die Sicherheit des Schemas gewährleisten zu können, darf es deshalb nicht möglich sein, mit $D_{K_{n+1}}$ ein Element $C_{g_i} \in C_{\mathcal{G}}$ für $i = 1, ..., n$ entschlüsseln zu können. Das heißt, daß bei der Chiffriertransformation E_{K_1} der Verkettungsfunktion auf Seite 50 für jedes g_{n+1} ein anderer Schlüssel K_{n+1} verwendet werden muß. Genauer gesagt muß es berechnungsmäßig praktisch unmöglich sein, für $i = 1, ..., n+1$ aus einem Schlüssel K_i einen Schlüssel K_j für $j = 1, ..., i-1$ zu bestimmen. In den folgenden Ausführungen wird daher das Hauptaugenmerk auf die Auswahl eines neuen Schlüssels K_{n+1} aus $\mathcal{K}$ gerichtet.

4.3.1 Einweg-Hashfunktion

Als Voraussetzung wird angenommen, daß eine Einweg-Hashfunktion

$$h_e : \; X^* \longrightarrow X^m \quad \text{mit} \quad m \in \mathbb{N} \quad \text{und} \quad X^m \subseteq X^* = \mathcal{K}$$

existiert (s. a. Def. 1.6.5). Eine Möglichkeit zur Auswahl von K_{n+1} aus $\mathcal{K}$ besteht dann darin, K_{n+1} mittels

$$K_{n+1} = h_e(K_n)$$

für $n > 0$ aus $K_n \in \mathcal{K}$ abzuleiten, wobei K_1 vorgegeben werden muß.

Der Schlüssel K_{n+1} kann gleichzeitig auch als Kennung $k_{g_{n+1}}$ genutzt werden, wobei $E_{\mathcal{Z}}$ mit $\mathcal{Z} = \mathcal{K}$ für $n > 0$ aus h_e besteht. Dadurch kann bei dieser Realisierung $(C_{\mathcal{G}}, k_{g_1})$ sowohl eine unabhängige kryptographische Verkettung mit Wurzel v_1 sein, wenn bei einer Überprüfung der Berechtigte aus $\mathcal{B}_p$ die Einweg-Hashfunktion h_e kennt, als auch eine abhängige kryptographische Verkettung mit Wurzel v_1, wenn er sie nicht kennt.

In dem Fall, daß alle Berechtigten aus $\mathcal{B}_p$ auch h_e kennen, kann darauf verzichtet werden, $k_{g_{n+1}}$ und $Z_{g_{n+1}}$ zusammen mit v_{n+1} zu verschlüsseln, so daß

$$C_{g_{n+1}} = E_{K_{n+1}}(v_{n+1})$$

ist. Für die Prüffunktion ist dann $D_{K_2}(C_{g_i})$ authentisch, wenn $D_{K_2}(C_{g_i})$ plausibel ist, weshalb hier im Gegensatz zum Schema eine Unabhängigkeit der Prüfbarkeit vorliegt (s. a Abschnitt 3.4.1).

Vom Prinzip her ist diese Art der Realisierung des Schemas mit der in [KS99, SK98, SK99] beschriebenen Vorgehensweise für eine sichere Log-Datei vergleichbar. Ansonsten ist die dort beschriebene Vorgehensweise allerdings wegen einer Vielzahl von Nebenbedingungen sehr viel komplexer.

4.3.2 Einwegfunktion

Unter der Annahme, daß eine Einwegfunktion

$$f_e : \mathcal{K} \longrightarrow \mathcal{K}$$

existiert (s. a. Def. 1.6.6), besteht eine weitere Möglichkeit zur Auswahl von K_{n+1} aus $\mathcal{K}$ zum Beispiel darin, K_{n+1} mittels

$$K_{n+1} = f_e(K_n)$$

aus $K_n \in \mathcal{K}$ abzuleiten.

Für f_e gelten im Prinzip die gleichen Folgerungen wie oben für h_e beschrieben, wobei allerdings die Längen der Schlüssel von K_n und K_{n+1} besonders berücksichtigt werden müssen:

1. Wenn K_{n+1} stets kürzer ist als K_n, dann wird die unberufene Entzifferung der $C_{g_{n+x}}$ für $x \nearrow \infty$ und $x \in \mathbb{N}$ immer leichter.

2. Wenn K_{n+1} stets genauso lang ist wie K_n, dann sollte von f_e her keine Gefahr für die Sicherheit von $C_{g_{n+x}}$ auftreten.

3. Wenn K_{n+1} stets länger ist als K_n, dann wächst die für $E_{K_{n+x}}(g_{n+x})$ und $D_{K_{n+x}}(C_{g_{n+x}})$ benötigte Zeit und der für K_{n+x} benötigte Speicherplatz.

Um diesen Problemen aus dem Weg zu gehen und die Längen der Schlüssel K_{n+x} zu garantieren, sollte daher eher eine Einweg-Hashfunktion eingesetzt werden.

4.3.3 Zufallsfolgengenerator

Für die letzte aufgeführte Möglichkeit wird angenommen, daß ein Zufallsfolgengenerator

$$RND : \mathbb{N} \longrightarrow X^m \quad \text{mit} \quad m \in \mathbb{N} \quad \text{und} \quad X^m \subseteq X^* = \mathcal{K}$$

existiert (s. a. Def. 1.6.8). Dadurch kann K_{n+1}, im Vergleich zu h_e und f_e unabhängig von K_n, mittels

$$K_{n+1} = RND(m)$$

für $n \geq 0$ zufällig aus $\mathcal{K}$ ausgewählt werden, wobei für K_{n+1} jeweils eine bestimmte Länge m vorgegeben wird.

Die eigentliche Idee besteht nun darin, RND als $E_{\mathcal{Z}}$ mit $\mathcal{Z} = \mathcal{K}$ zu nutzen, weshalb $k_{g_{n+1}} = K_{n+1}$ ist. Das bedeutet, daß $(C_{\mathcal{G}}, k_{g_1})$ eine abhängige kryptographische Verkettung mit Wurzel v_1 ist.

Im Vergleich zur Verwendung von h_e oder f_e kann bei der Verwendung von RND keine unabhängige kryptographische Verkettung erreicht werden, wenn einem Berechtigtem aus $\mathcal{B}_p$ nur k_{g_1} bekannt sein soll.

4.4 Public-Key-Kryptosysteme

Im Gegensatz zum vorhergehenden Abschnitt 4.3 wird hier die Realisierung des Schemas aus Abschnitt 4.2 auf Grundlage eines Public-Key-Kryptosystems

$$KS = (\mathcal{M}, \mathcal{C}, \mathcal{K}, E_{K_1}, D_{K_2})$$

beschrieben. Das heißt, daß

$$K_1 \neq K_2$$

ist. Für einen Eindringling aus $\mathcal{U}$ bedeutet das, daß er aus der Kenntnis von E_{K_1} zum Verschlüsseln von g_{n+1} nicht D_{K_2} zum Entschlüsseln von $C_{g_{n+1}}$ ableiten kann. Für die Sicherheit des Schemas muß daher nur gewährleistet werden, daß ein Eindringling aus $\mathcal{U}$ nicht auf einem anderen Wege Kenntnis von D_{K_2} bzw. von K_2 zum Entschlüsseln von $C_{g_i} \in C_{\mathcal{G}}$ für $i = 1, ..., n$ erlangen kann. Dies wäre zum Beispiel der Fall, wenn D_{K_2} bzw. K_2 bei einem Einbruch in $\mathcal{R}$ dort vorhanden sind. Im Vergleich zu Abschnitt 4.3 geht es daher jetzt im wesentlichen um die Auswahl der privaten Zusatzinformation aus $\mathcal{Z}$ (s. a. „Sicherheit“ auf Seite 51).

4.4.1 Einweg-Hashfunktion

In Kombination mit einem Public-Key-Kryptosystem sei

$$h_e : X^* \longrightarrow X^m \quad \text{mit} \quad m \in I\!N \quad \text{und} \quad X^m \subseteq X^* = \mathcal{Z}$$

eine Einweg-Hashfunktion, bei der X^* nicht gleich $\mathcal{K}$ sein muß.

In der Initialisierungsfunktion wird k_{g_1} mit einer Zufallsfolge zum Beispiel von $RND : I\!N \longrightarrow X^m$ vorbelegt. Danach wird für $n > 0$ jede neue Kennung $k_{g_{n+1}}$ mittels $k_{g_{n+1}} = h_e(k_{g_n})$ aus k_{g_n} abgeleitet. Das heißt,

$$k_{g_{n+1}} = E_{\mathcal{Z}}(v_{n+1}) = \begin{cases} RND(m) & \text{falls} \quad n = 0, \\ h_e(k_{g_n}) & \text{falls} \quad n > 0. \end{cases}$$

Bei $(C_{\mathcal{G}}, k_{g_1})$ handelt es sich um eine unabhängige kryptographische Verkettung mit Wurzel v_1. Ähnlich wie in Abschnitt 4.3.1 kann hier auf die Verschlüsselung von $Z_{g_{n+1}}$ verzichtet werden, wenn h_e allen Berechtigten aus $\mathcal{B}_p$ bekannt ist, so daß

$$C_{g_{n+1}} = E_{K_1}((k_{g_{n+1}}, v_{n+1}))$$

ist. Außerdem ist in diesem Fall die Prüfbarkeit unabhängig, da die Prüffunktion k_{g_i} aus k_{g_1} ableiten kann, ohne zuvor $C_{g_{i-1}}$ überprüfen zu müssen (s. a Abschnitt 3.4.1).

Alternativ kann mittels $k_{g_{n+1}} = h_e((k_{g_n}, v_n))$ auch noch v_n mit einbezogen werden. Dadurch kann die Prüffunktion k_{g_i} nur aus k_{g_1} ableiten, wenn sie zuvor $C_{g_{i-1}}$ überprüft hat, weshalb die Prüfbarkeit jetzt abhängig ist.

Sicherheit Abgesehen von der Sicherheit von KS und h_e ist ein besonderes Augenmerk auf die Größe von m bzw. die Länge der k_{g_i} aus $D_{K_2}(C_{g_i})$ mit $C_{g_i} \in C_{\mathcal{G}}$ und $i = 1, ..., n$ zu legen.

Bei einem Eindringling aus $\mathcal{U}$ muß angenommen werden, daß er Z_{g_n} und v_n für C_{g_n} kennt bzw. leicht erraten kann. Diese Annahmen werden dadurch gerechtfertigt, daß $k_{g_{n+1}}$ mit $Z_{g_n} = \{k_{g_{n+1}}\}$ beim Einbruch in $\mathcal{R}$ dort vorhanden ist und v_n aus dem Programm ersichtlich werden sollte, das jeweils v_i für C_{g_i} erstellt hat.

Daraus folgt, daß er k_{g_n} in $|X|^m$ Schritten mittels eines *Brute-Force-Angriffs*, das heißt durch das Ausprobieren aller Möglichkeiten (s. a. [Schn96]), bestimmen kann. Dafür vergleicht er $C_{g_j} = E_{K_1}((k_{g_j}, \{k_{g_{n+1}}\}, v_n))$ für $k_{g_j} \in X^m$ und $j = 1, ..., |X|^m$ mit C_{g_n}, bis er ein $C_{g_j} = C_{g_n}$ gefunden hat, für das dann $k_{g_j} = k_{g_n}$ gilt.

Mit dem gleichen Algorithmus kann er nun $k_{g_{n-1}}$ bis k_{g_1} bestimmen, wodurch er $(C_{\mathcal{G}}, k_{g_1})$ bis auf k_{g_1} beliebig ändern kann. Daher muß m je nach Bedarf ausreichend groß gewählt werden, wobei $m = 512$ nach heutigem Kenntnisstand ausreichend sein sollte (s. a. [Schn96]), wenn einmal von der Sicherheit von KS und h_e abgesehen wird.

4.4.2 Einwegfunktion

Bei Verwendung einer Einwegfunktion

$$f_e : \mathcal{Z} \longrightarrow \mathcal{Z} \quad \text{für} \quad k_{gn+1} = f_e(k_{g_n})$$

sind analog zu Abschnitt 4.3.2 vor allem die Längen der Kennungen k_{g_n} und $k_{g_{n+1}}$ zu beachten.

Im Unterschied zu Abschnitt 4.3.2 wird dabei nicht die Sicherheit von KS schwächer, wenn $k_{g_{n+1}}$ stets kürzer ist als k_{g_n}. Vielmehr kann der oben in Abschnitt 4.4.1 beschriebene Brute-Force-Angriff für $C_{g_{n+x}}$ mit $x \nearrow \infty$ und $x \in \mathbb{N}$ immer schneller durchgeführt werden.

4.4.3 Zufallsfolgengenerator

Ein Zufallsfolgengenerator läßt sich in diesem Zusammenhang auf mindestens zwei verschiedene Arten einsetzen, wobei er entweder

1. eine Zusatzinformation $k_{g_{n+1}}$ oder
2. ein Schlüsselpaar $(K_{1_{n+1}}, K_{2_{n+1}})$

erzeugt.

Zusatzinformation Die erste Einsatzart erinnert an Abschnitt 4.4.1. Dabei besteht ein Unterschied darin, daß diesmal

$$RND: \mathbb{N} \longrightarrow X^m \quad \text{mit} \quad m \in \mathbb{N} \quad \text{und} \quad X^m \subseteq X^* = \mathcal{Z}$$

ständig für $E_{\mathcal{Z}}$ genutzt wird, so daß

$$k_{g_{n+1}} = E_{\mathcal{Z}}(v_{n+1}) = RND(m) \quad \text{für} \quad n \geq 0$$

ist.

Bei $(C_{\mathcal{G}}, k_{g_1})$ handelt es sich auch diesmal um eine unabhängige kryptographische Verkettung mit Wurzel v_1. Ein weiterer Unterschied ist allerdings, daß nicht auf die Verschlüsselung von $Z_{g_{n+1}}$ verzichtet werden kann, da keine Möglichkeit besteht, eine Kennung k_{g_i} aus k_{g_1} abzuleiten. Ansonsten gelten für die Größe von m die gleichen Überlegungen, wie in Abschnitt 4.4.1, weshalb diese hier nicht wiederholt werden.

Schlüsselpaar Aus der folgenden Einsatzart geht meine erste grundlegende Idee für diese Arbeit hervor.

Dafür sei RND ein Zufallsfolgengenerator, der für KS zufällig ein Schlüsselpaar (K_1, K_2) mit $K_1, K_2 \in \mathcal{K}$ liefert, das gewissen Mindestanforderungen genügt. Auf die Mindestanforderungen wird an dieser Stelle nicht weiter eingegangen, sondern unter dem Stichwort „Schlüsselerzeugung“ auf [MOV97, Schn96, Wät99b] verwiesen. In Anlehnung an Abschnitt 4.3.3 kann dann mittels

$$(K_{1_{n+1}}, K_{2_{n+1}}) = RND(Mindestanforderungen)$$

unabhängig von (K_{1_n}, K_{2_n}) zufällig für jedes $C_{g_{n+1}}$ ein neues Schlüsselpaar erzeugt werden, wobei

$$k_{g_{n+1}} = K_{2_{n+1}} = E_{\mathcal{Z}}(v_{n+1}) \quad \text{mit} \quad \mathcal{Z} = \mathcal{K} \quad \text{und} \quad n \geq 0$$

ist.

Unter der Annahme, daß $K_{1_{n+1}}$ nicht bekannt wird und nach der Erstellung von

$$C_{g_{n+1}} = E_{K_{1_{n+1}}}((k_{g_{n+1}}, \{k_{g_{n+2}}\}, v_{n+1})) = E_{K_{1_{n+1}}}((K_{2_{n+1}}, \{K_{2_{n+2}}\}, v_{n+1}))$$

stets von $K_{1_{n+2}}$ überschrieben wird, kann $(C_{\mathcal{G}}, k_{g_1})$ selbst ein Berechtigter aus $\mathcal{B}$ nicht ändern, wenn k_{g_1} allen Individuen aus $\mathcal{I}$ bekannt ist. Besser gesagt ist in diesem Fall $\mathcal{B} = \emptyset$ und $\mathcal{U} = \mathcal{I}$ bzw. $\mathcal{B}_p = \mathcal{I}$ und $\mathcal{U}_p = \emptyset$ (s. a. „Beispiel für Abgeschlossenheit“ auf Seite 44). Außerdem folgt daraus, daß $(C_{\mathcal{G}}, k_{g_1})$ eine abhängige kryptographische Verkettung mit Wurzel v_1 ist.

4.4.4 Einhüllend

Das auf Seite 47 angekündigte einhüllende Prinzip bei Public-Key-Kryptosystemen ist in mancherlei Hinsicht anders, als die bisher vorgestellten Prinzipien. Es kommt bei der Verkettung und Überprüfung ohne weitere Hilfsfunktionen aus, wobei allerdings für die Initialisierung ein Zufallsfolgengenerator $RND: \mathbb{N} \longrightarrow$

X^m mit $m \in \mathbb{N}$ benötigt wird. Desweiteren bleiben E_{K_1} und D_{K_2} unverändert, sind also für alle $C_g \in C_\mathcal{G}$ gleich, wobei D_{K_2} nicht in $\mathcal{R}$ vorhanden sein darf.

Im Unterschied zum Schema kann auf die Zuordnung von $k_{g_{n+1}}$ zu v_{n+1} verzichtet werden, weshalb auch $Z_{g_{n+1}}$ entfällt. Vielmehr ist nun

$$C_{g_{n+1}} = \begin{cases} E_{K_1}(v_1) \quad \text{mit} \quad v_1 = RND(m) \quad \text{falls} \quad n = 0, \\ E_{K_1}((C_{g_n}, v_{n+1})) \quad \text{falls} \quad n > 0, \end{cases}$$

wobei C_{g_n} mit $n > 0$ stets von $C_{g_{n+1}}$ unwiederbringlich überschrieben wird, so daß

$$C_\mathcal{G} = \{C_{g_{n+1}}\}$$

ist. Dabei wird C_{g_1} von der Initialisierungsfunktion erzeugt, die v_1 mit $RND(m)$ vorbelegt und v_1 anstelle von k_{g_1} außerhalb von $\mathcal{R}$ für die Prüfung speichert.

Die Prüfbarkeit von C_{g_i} ist abhängig von der vorhergehenden Prüfung von $C_{g_{i+1}}$ für $i = n-1, ..., 1$, da erst durch das Entschlüsseln von $C_{g_{i+1}}$ ein Zugriff auf C_{g_i} möglich ist. Dabei ist $D_{K_2}(C_{g_{i+1}})$ authentisch, wenn $D_{K_2}(C_{g_{i+1}})$ plausibel ist und $D_{K_2}(C_{g_1})$ mit der außerhalb von $\mathcal{R}$ gespeicherten Ecke v_1 übereinstimmt.

Das bedeutet, daß die Erzeugung und die Prüfung von $(C_\mathcal{G}, v_1)$ anders als bisher in entgegengesetzten Richtungen verlaufen. Dabei ist v_1 keine Wurzel, sondern eine Senke.

Ein gravierender Nachteil dieses Prinzips ist, daß für jede weitere Ecke auch alle vorhergehenden Ecken noch einmal verschlüsselt werden müssen. Zur Verschlüsselung von n Ecken beträgt daher die Zeitkomplexität

$$O(1 + 2 + 3 + ... + n) = O\left(\frac{n}{2} \cdot (n+1)\right) \leq O\left(n^2\right),$$

während sie für die anderen Realisierungen $O(n)$ beträgt. Als Lösung könnte sich anbieten, nur eine Hälfte des Elements C_{g_n} in $C_{g_{n+1}}$ zu verschlüsseln und nur diese Hälfte von C_{g_n} zu überschreiben, wodurch die Zeitkomplexität $O(1.5 \cdot n)$ betragen würde. Diese Lösung soll hier aber nicht weiter verfolgt werden.

4.5 Alternativen

Abschließend werden Alternativen zur Sicherung der Einträge von Log-Dateien aufgelistet, die nicht auf einer kryptographischen Verkettung beruhen.

Vorherige Schlüsselerzeugung Die erste Alternative beruht auf der Idee des *One-Time-Pads*[2]. Sie erinnert noch sehr stark an kryptographische Verkettungen, da bei ihr jede neue Ecke v_{n+1} jeweils mit einem eigenem Schlüssel $K_{1_{n+1}}$ mittels

$$C_{g_{n+1}} = E_{K_{1_{n+1}}}(v_{n+1})$$

signiert oder verschlüsselt wird. Dafür wird der Schlüssel $K_{1_{n+1}}$ allerdings nicht während der Laufzeit neu erzeugt, sondern im voraus in einer Datei abgelegt. Aus dieser wird er dann während der Laufzeit ausgelesen und gelöscht.

[2] S. [Schn96] auf den Seiten 17–20.

Das bedeutet, daß die Datei bei einem symmetrischem Kryptosystem auch außerhalb von $\mathcal{R}$ gespeichert werden muß, während dies bei einem Public-Key-Kryptosystem entsprechend für $K_{2_{n+1}}$ gilt. Als Alternative zur Datei könnte ein mit $\mathcal{R}$ verbundener vertrauenswürdiger zweiter Rechner $K_{1_{n+1}}$ zur Laufzeit erzeugen und $K_{2_{n+1}}$ bei sich lokal speichern.

XOR-Verknüpfung Die zweite Alternative basiert auf der Idee, jede neue Ecke v_{n+1} mit einem One-Time-Pad XOR zu verknüpfen. Dazu muß das One-Time-Pad im voraus erzeugt und sowohl in $\mathcal{R}$ als auch außerhalb von $\mathcal{R}$ gespeichert werden. Bedingung dabei ist, daß die Ecken v_i für $i = 1, ..., n$ nicht erratbar sein dürfen, da ansonsten die XOR-Verknüpfung umgekehrt werden könnte und danach eine manipulierte Ecke v_i' für v_i eingesetzt werden könnte. Zur Prüfung kann dann die XOR-Verknüpfung mit dem außerhalb von $\mathcal{R}$ gespeichertem One-Time-Pad umgekehrt werden, so daß die Ecken v_i sichtbar werden. Dabei ist eine Ecke v_i authentisch, wenn sie plausibel ist.

Hashkette Die dritte Alternative basiert auf einer Kette von Hashwerten, für die h_e eine Einweg-Hashfunktion sei. Dabei wird die Variable y_0 mit $RND(m)$ vorbelegt und sowohl in $\mathcal{R}$ als auch außerhalb von $\mathcal{R}$ gespeichert. Für $n \geq 0$ ist dann

$$y_{n+1} = h_e(y_n, v_{n+1}),$$

wobei y_n unwiederbringlich von y_{n+1} überschrieben wird. Zur Prüfung muß dann der Hashwert y_n mittels der außerhalb von $\mathcal{R}$ gespeicherten Variable y_0 nachgerechnet werden.

Druckausgabe Die vierte Alternative ist die klassische Methode von Clifford Stoll (s. [Sto89]), die jede neue Ecke v_{n+1} auf einem Drucker ausgibt. Als Optimierung könnte es ausreichend sein, nur $h_e(v_{n+1})$ auf dem Drucker auszugeben.

Kapitel 5

Zusammenfassung und Ausblick

Diese Arbeit wurde von der Frage geleitet, wie die Authentizität, Reihenfolge und Vollständigkeit der Einträge einer Log-Datei sichergestellt werden kann, wobei die Log-Datei um weitere Einträge erweiterbar sein muß.

Dazu wurde im ersten Kapitel der Begriff Log-Datei formalisiert und auf die benötigten Grundlagen für kryptographische Verkettungen eingegangen, wobei der Schwerpunkt auf der Graphentheorie und der Kryptographie lag. Danach wurde im zweiten Kapitel die Problematik bei der Formalisierung des Begriffs Sicherheit dargestellt, wobei die kryptographische Verkettung als Maßnahme zur logischen Sicherheit von Informationen eingeordnet wurde.

Als Basis für das vierte Kapitel, wurde im dritten Kapitel eine allgemeine Theorie für kryptographische Verkettungen erarbeitet, bei der die Ecken eines beliebigen Graphen, entsprechend den zwischen ihnen existierenden Kanten, mittels kryptographischer Verfahren miteinander verkettet werden. Dadurch soll kein Unberechtigter Ecken unbemerkt manipulieren oder Ecken bzw. Kanten unbemerkt aus dem Graphen entfernen können. Die Konstruktionsvorschrift für die kryptographische Verkettung des Graphen kann dabei so angelegt werden, daß der Graph und entsprechend dessen kryptographische Verkettung um weitere Ecken und Kanten erweitert werden können.

Anschließend wurde auf Grundlage der Theorie im vierten Kapitel ein spezielles Schema zur kryptographischen Verkettung eines gerichteten Hamiltonschen Weges vorgestellt. Mit Hilfe dieses Schemas kann die Authentizität und Reihenfolge der Einträge einer Log-Datei überprüft und das Fehlen von Einträgen erkannt werden. Da sich das Schema in Kombination mit verschiedenen kryptographischen Hilfsfunktionen sowohl mittels symmetrischer Kryptosysteme als auch mittels Public-Key-Kryptosystemen realisieren ließ, wurden dadurch im Vergleich zu der in [KS99, SK98, SK99] vorgestellten Idee weitere Wege für die kryptographische Verkettung der Einträge von Log-Dateien aufgezeigt.

Andere Arbeiten Die durch die Theorie ermöglichte Einordnung verschiedener anderer Arbeiten in einen größeren Zusammenhang erfolgte nur ansatzweise, da sie kein primäres Ziel dieser Arbeit war.

Offene Fragen Nach dieser Zusammenfassung werden als nächstes einige

Fragen aufgelistet, die es in zukünftigen Arbeiten zu klären gilt:

1. Die Theorie müßte stärker berücksichtigen, daß nicht immer k_g und Z_g eines Gliedes $g \in \mathcal{G}$ benötigt werden, wie dies zum Beispiel in den Abschnitten 4.3.1, 4.4.1 und 4.4.4 gezeigt wurde.

2. Die kryptographische Verkettung müßte mit Hashketten verglichen werden.

3. Der Einfluß der Blockchiffriermodi auf die kryptographische Verkettung müßte untersucht werden (s. a. Seite 40 und Seite 52).

4. Die Einordnung der Arbeiten von Haber und Stornetta [HS91, BHS93, HS97], Anderson [And96], Schneier und Kelsey [KS96, KS99, SK97a, SK97b, SK98, SK99] sowie Merkle [Mer89] erfolgte nur ansatzweise und könnte weiter präzisiert werden, wenn die Fragen 1 bis 3 geklärt sind.

5. Die angedeutete schnellere Version des einhüllenden Prinzips müßte ausführlicher behandelt werden (s. a. Abschnitt 4.4.4).

6. Die Verwendung der kryptographischen Verkettung für andere Einsatzbereiche müßte umfassend untersucht werden.

Warnung Zum Schluß sei davor gewarnt, sich bei der Verwendung von kryptographischen Verkettungen in falscher Sicherheit zu wiegen, da auch die pyhsische und die organisatorische Sicherheit von Informationen beachtet werden müssen und nicht vernachlässigt werden dürfen. Denn was nutzt die kryptographische Verkettung einer Log-Datei, wenn ein Angreifer zunächst die privaten Zusatzinformationen vom Monitor des Rechners ablesen kann, bevor er in den Rechner eindringt? Eine anstandslose Überprüfung der Log-Datei bedeutet daher nicht zwangsläufig, daß kein Einbruch stattgefunden hat.

Anhang A

Beispielprogramm

Die Motivation für dieses Beispielprogramm besteht darin, die Praxistauglichkeit der in Kapitel 4 erarbeiteten Realisierungsmöglichkeiten für sichere Log-Dateien testen und demonstrieren zu können.

Aufgrund des beschränkten Zeitrahmens werden nur die auf Public-Key-Kryptosystemen beruhenden Realisierungen mittels einer Einweg-Hashfunktion, eines Zufallsfolgengenerators und des einhüllenden Prinzips implementiert, die in den Abschnitten 4.4.1, 4.4.3 und 4.4.4 beschrieben wurden. Eine spätere Implementierung der auf symmetrischen Kryptosystemen beruhenden Realisierungsmöglichkeiten wird allerdings vorbereitet.

Die weitere Beschreibung des Beispielprogramms stellt in den Abschnitten A.1 und A.2 dessen Implementierung und Anwendung dar.

A.1 Implementierung

Die Implementierung beruht auf der Programmiersprache C++ (s. a. [Str98]). Dadurch konnte das Programm nach objektorientierten Kriterien aus verschiedenen Klassen aufgebaut werden. Für das Zusammenspiel der Klassen wurden verschiedene Entwurfsmuster aus [GHJV96] eingesetzt, wobei der Kern des Programms aus dem *Abstrakte-Fabrik*-Erzeugungsmuster und dem *Strategie*-Verhaltensmuster aufgebaut wurde (s. a. Abb. A.1).

Fabrik Nach [GHJV96] bietet das Abstrakte-Fabrik-Erzeugungsmuster eine Schnittstelle zum Erzeugen von Familien verwandter Objekte, ohne auf deren konkreten Klassen einzugehen. Bezogen auf das Programm bedeutet das, daß die Klasse `CSecLog` eine Schnittstelle zum Erzeugen von sicheren Log-Dateien darstellt.

Dadurch muß in `main` nur an genau einer Stelle festgelegt werden, welcher Typ von sicheren Log-Dateien im folgenden Programmlauf erzeugt werden soll, wobei jeder Typ für genau eine Realisierungsmöglichkeit steht. Das hat den Vorteil, daß in `main` nur noch über `CSecLog` auf die entsprechende Klasse zugegriffen wird und nicht jedesmal, zum Beipiel beim Anfügen eines neuen Eintrags, unterschieden werden muß, welche Funktion für einen bestimmten Typ aufgerufen werden muß. Hinzukommt, daß dadurch das Programm auf einfache Art und Weise um eine weitere Realisierungsmöglichkeit erweitert werden kann,

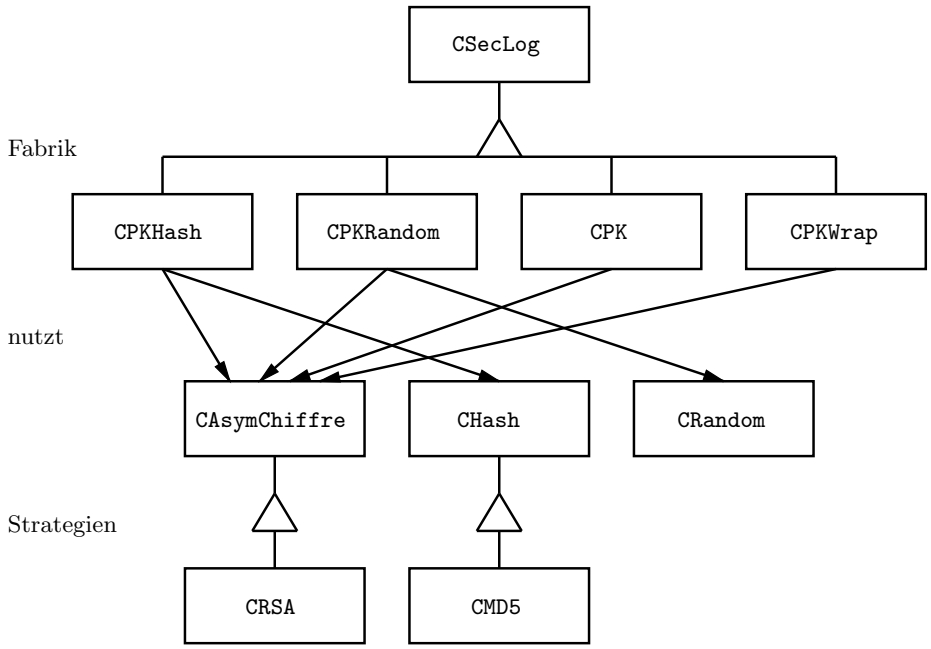

Abbildung A.1: Klassenübersicht

indem für sie zunächst die entsprechende Klasse erstellt wird und danach für diese in `main` an der Stelle zur Festlegung des Typs eine weitere Unterscheidung aufgenommen wird.

In den nachfolgend aufgeführten Klassen, die von `CSecLog` abgeleitet werden, wird jeweils eine bestimmte auf Public-Key-Kryptosystemen beruhende Realisierung implementiert:

- `CPKHash` — Einweg-Hashfunktion aus 4.4.1,
- `CPKRandom` — Zufallsfolgengenerator für Zusatzinformation aus 4.4.3,
- `CPK` — Zufallsfolgengenerator für Schlüsselpaar aus 4.4.3 und
- `CPKWrap` — einhüllendes Prinzip aus 4.4.4.

Dabei werden in jeder abgeleiteten Klasse die folgenden Funktionen implementiert:

- `InitKeys` und `InitLog` zum Initialisieren der Schlüssel und der kryptographischen Verkettung,
- `AddEntry` zum Anhängen eines neuen Eintrags an die kryptographische Verkettung und
- `Check` zum Überprüfen der kryptographischen Verkettung.

Alle weiteren benötigten Funktionen zur Handhabung von sicheren Log-Dateien, die nicht von einer bestimmten Realisierungsmöglichkeit abhängig sind,

werden in `CSecLog` zusammengefaßt und implementiert. Dadurch wird Redundanz vermieden.

Für die Nachrüstung der auf symmetrischen Kryptosystemen beruhenden Realisierungsmöglichkeiten müssen zumindest zwei Schritte vollzogen werden. Als erstes sind die Klassen

- `CSymHash` für Abschnitt 4.3.1 und
- `CSym` für Abschnitt 4.3.3

von `CSecLog` abzuleiten und jeweils die oben aufgeführten Funktionen zu implementieren. Als zweites muß in `main` jeweils eine Option zum Auswählen dieser Realisierungsmöglichkeiten eingebaut werden. Dafür muß außerdem die Klasse `CSymChiffre` für Symmetrische Kryptosysteme analog zu `CAsymChiffre` einmalig implementiert werden.

Strategien Das Strategie-Verhaltensmuster dient nach [GHJV96] dazu, bei einer Familie von Algorithmen jeden einzelnen zu kapseln und alle gegeneinander austauschbar zu machen. Der Unterschied zum Abstrakte-Fabrik-Erzeugungsmuster besteht hier darin, daß jenes Objekte unterschiedlichen Typs zurückliefert, während das Strategie-Verhaltensmuster stets Objekte gleichen Typs zurückliefert, die aber unterschiedliche Werte haben können. Die Implementierung wird wieder so gestaltet, daß eine Basisklasse als Schnittstelle zum Zugriff auf die unterschiedlichen Strategien dient, wobei deren konkrete Klassen von der Basisklasse abgeleitet werden.

Als Schnittstellen sind hierbei hervorzuheben die Basisklassen

- `CAsymChiffre` für Public-Key-Kryptosysteme,
- `CHash` für Hashfunktionen und
- `CRandom` für Zufallsfolgengeneratoren.

Dafür wurden bisher nur das RSA-Public-Key-Kryptosystem in `CRSA` und die MD5-Einweg-Hashfunktion in `CMD5` implementiert, die jeweils in [Schn96] beschrieben werden. Für ernsthafte Anwendungen müßte in `CRSA` noch der auf Seite 52 beschriebene Blockchiffriermodus implementiert und in `CRandom` der Zufallsfolgengenerator überarbeitet werden, da dieser auf eine Systemfunktion zurückgreift.

Ähnlich wie bei der Fabrik ist es möglich, das Programm um weitere Strategien zu erweitern, indem die konkrete Klasse für eine Strategie von der entsprechenden Basisklasse abgeleitet wird und in `main` eine Option zur Auswahl dieser Strategie eingebaut wird.

Verzeichnisbaum Wie in dem Verzeichnisbaum in Abbildung A.2 zu sehen ist, enthält das Hauptverzeichnis die beiden Unterverzeichnisse `doc` und `src`, in denen zum einen diese Ausarbeitung und zum anderen die Quelltexte des Beispielprogramms zu finden sind. Das Unterverzeichnis `src` enthält dabei weitere Unterverzeichnisse für die oben erwähnten Klassen und die Unterverzeichnisse `Misc` und `Test`. In `Misc` befinden sich diverse Hilfsfunktionen, wobei in `convert.{cc|h}` Funktionen zur Wort- und Zahltransformation enthalten

sind, die sich an Abschnit 1.3 orientieren. Die anderen Hilfsfunktionen werden nicht weiter beschrieben, da sie hier nur nebensächliche Details darstellen. Auf `Test` wird im nächsten Abschnitt A.2 eingegangen.

doc	SecLog.ps	
src	SecLog	CSecLog.{cc|h}, CPKHash.{cc|h}, CPKRandom.{cc|h}, CPK.{cc|h}, CPKWrap.{cc|h}
	SymChiffre	
	AsymChiffre	CAsymChiffre.{cc|h}, CRSA.{cc|h}
	Hash	CHash.{cc|h}, CMD5.{cc|h}
	Misc	CBytes.{cc|h}, CDebug.{cc|h}, CObject.{cc|h}, CRandom.{cc|h}, Easy.h, IO.{cc|h}, convert.{cc|h}, math.{cc|h}
	Test	../seclog, su
	Makefile, main.cc, seclog	

Abbildung A.2: Verzeichnisbaum

Compilieren Zum Compilieren des Programms wird für Langzahl-Arithmetik ein Datentyp `Integer` benötigt. Dieser wird zum Beispiel vom `libg++-2.8.1.1a`-Zusatz zur `libstdc++` des GNU-Projekts bereitgestellt. Das Compilieren selbst wird mit Hilfe eines `Makefile`s gesteuert, das für `GNU Make version 3.76.1` angelegt wurde. Der dabei aufgerufene C++-Compiler `g++` des GNU-Projekts sollte mindestens in `version 2.95.1` vorliegen.

Nach dem Wechsel in das Unterverzeichnis `src` kann dort durch Aufruf von

```
make
```

das Programm compiliert werden, das dabei in der ausführbaren Datei `seclog` abgelegt wird. Die Anwendung des Programms wird im nächsten Abschnitt A.2 beschrieben.

A.2 Anwendung

Nach der Beschreibung der Implementierung des Beispielprogramms im vorherigen Abschnitt A.1 folgt nun eine kurze Beschreibung seiner Anwendung.

Die Benutzerschnittstelle des Programms beschränkt sich auf das Notwendigste und sollte vor allem schnell zu implementieren sein. Sie hat dabei folgende Syntax:

```
seclog [option] <pMin> <qMin> <tMax> <logfile> <entries>
```

Bei den weiteren Erläuterungen dieser Syntax muß zwischen

- dem Anfügen von Einträgen an eine Log-Datei und
- dem Überprüfen der Einträge in einer Log-Datei

unterschieden werden.

Anfügen Die Initialisierung einer neuen Log-Datei `<logfile>` erfolgt automatisch, wenn `<logfile>` beim Anfügen der Einträge `<entries>` noch nicht existiert. Außerdem wird in diesem Fall für die Klasse `CRSA` ein neues Schlüsselpaar aus öffentlichem und privatem Schlüssel erzeugt. Dieses wird in `<logfile>.pub` und `<logfile>.pri` getrennt abgelegt. Dabei ist darauf zu achten, daß der private Schlüssel in `<logfile>.pri` vom Rechner zu entfernen ist und für eine spätere Überprüfung von `<logfile>` extern aufgehoben werden muß.

Die Erzeugung von Schlüsseln für das RSA-Public-Key-Kryptosystem orientiert sich an dem in [Wät99b] auf Seite 76 beschriebenen Algorithmus 5.2.3. Dafür muß mittels der Parameter `<pMin>` und `<qMin>` die minimale Anzahl der Dezimalstellen (Länge, Größe) der Primzahlen p und q angegeben werden und mittels des Parameters `<tMax>` die maximale Länge von $ggT(p-1, q-1)$. Außer zum Erzeugen neuer Schlüsselpaare sollten diese Parameter auf 0 gesetzt werden.

Mittels `option` muß eine der implementierten Realisierungsmöglichkeiten für `<logfile>` ausgewählt werden:

- `-h` — Einweg-Hashfunktion aus 4.4.1,
- `-r` — Zufallsfolgengenerator für Zusatzinformation aus 4.4.3,
- `-p` — Zufallsfolgengenerator für Schlüsselpaar aus 4.4.3 und
- `-w` — einhüllendes Prinzip aus 4.4.4.

Diese darf beim Anfügen neuer `<entries>` an `<logfile>` nicht geändert werden, da es sonst zu undefinierten Zuständen kommen kann.

Beim Initialisieren einer neuen Log-Datei, die mittels einer Einweg-Hashfunktion gesichert werden soll (`option: -h`), wird ein zufälliger Initialisierungs-Hashwert erzeugt und ausgegeben. Dieser muß außerhalb des Rechners notiert werden, da er bei einer späteren Überprüfung mit dem ersten Wert in der Liste der ausgegebenen Hashwerte übereinstimmen muß. Analog dazu gilt dies auch für die Werte, die bei der Sicherung mittels einem Zufallsfolgengenerator für Zusatzinformation (`option: -r`) und bei der Sicherung mittels des einhüllenden Prinzips (`option: -w`) ausgegeben werden.

Überprüfung Für die Überprüfung der Einträge einer Log-Datei muß `seclog` diesmal ohne `<entries>` aber sonst wie beim Anfügen aufgerufen werden. Dabei ist darauf zu achten, daß die Überprüfung auf einem vertrauenswürdigen Rechner stattfindet. Desweiteren muß sich `<logfile>.pri` im gleichen Verzeichnis befinden, wie `<logfile>`, da `<logfile>` sonst zum Überprüfen nicht entschlüsselt werden kann.

Die Überprüfung wird mit 0 abgeschlossen, wenn kein Fehler gefunden wurde, und mit 1, wenn ein Fehler erkannt wurde. Es ist dabei darauf zu achten, daß die erste ausgegebene private Zusatzinformation mit der bei der Initialisierung ausgegebenen privaten Zusatzinformation übereinstimmen muß.

Befehle einhüllen Im Unterverzeichnis `Test` zeigt die Batch-Datei `su`, wie ein existierender Befehl eingehüllt werden kann, um dessen Aufruf zu protokollieren, ohne daß der Befehl selbst geändert werden muß. Es muß dabei nur dafür Sorge getragen werden, daß die Batch-Datei nicht umgangen werden kann.

Literaturverzeichnis

[Aig93] Martin Aigner. *Diskrete Mathematik.* Vieweg, Braunschweig 1993.

[And96] Ross J. Anderson. *The Eternity Service.* Pragocrypt 1996.

[AU96] Alfred V. Aho, Jeffrey D. Ullman. *Informatik.* International Thomson Publishing, Bonn 1996.

[Bau93] Friedrich L. Bauer. *Kryptologie.* Springer-Verlag, Berlin 1993.

[BHS93] Dave Bayer, Stuart Haber, W. Scott Stornetta. Improving the Efficiency and Reliability of Digital Time-Stamping. In *Sequences II: Methods in Communication, Security, and Computer Science*, pp. 329–334. Springer-Verlag, Berlin 1993.

[Bos96] Siegfried Bosch. *Algebra.* Springer-Verlag, Berlin, 2. Auflage 1996.

[BSI] *IT-Grundschutzhandbuch.* Bundesamt für Sicherheit in der Informationstechnik, in der neuesten Auflage.

[Bun96] Peter Bundschuh. *Einführung in die Zahlentheorie.* Springer-Verlag, Berlin, 3. Auflage 1996.

[GHJV96] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Entwurfsmuster.* Addison-Wesley, Bonn, 1. Auflage 1996.

[Gri94] Rüdiger Grimm. *Sicherheit für offene Kommunikation.* BI-Verlag, Mannheim 1994.

[HS91] Stuart Haber, W. Scott Stornetta. How to Time-Stamp a Digital Document. In *Advances in Cryptology – Crypto '90*, pp. 437–455. Lecture Notes in Computer Science v. 537, Springer-Verlag, Berlin 1991.

[HS97] Stuart Haber, W. Scott Stornetta. Secure Names for Bit-Strings. In *Proceedings of the 4th ACM Conference on Computer and Communication Security.* 1997.

[Jun94] Dieter Jungnickel. *Graphen, Netzwerke und Algorithmen.* BI-Verlag, Mannheim 1994.

[KS96] John Kelsey, Bruce Schneier. Authenticating Outputs of Computer Software Using a Cryptographic Coprocessor. In *Proceedings 1996 CARDIS*, pp. 11–24. September 1996.

[KS99] John Kelsey, Bruce Schneier. Minimizing Bandwidth for Remote Access to Cryptographically Protected Audit Logs. In *Second International Workshop on the Recent Advances in Intrusion Detection (RAID '99)*, to appear. September 1999.

[Lei89] Hans-Otto Leilich. *Rechnerstrukturen I.* Vorlesungsskript, Braunschweig, Oktober 1989.

[Mer89] Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology – Crypto '89*, pp. 218–238. Lecture Notes in Computer Science v. 435, Springer-Verlag, Berlin 1990.

[MOV97] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. *Handbook of applied crytography.* CRC Press, Boca Raton 1997.

[Pom91] Klaus Pommerening. *Datenschutz und Datensicherheit.* BI-Verlag, Mannheim 1991.

[Roh95] Hermann Rohling. *Einführung in die Informations- und Codierungstheorie.* Teubner Verlag, Stuttgart 1995.

[Rus12] Bertrand Russel. *The Problems of Philosophy.* Oxford University Press, 1912.

[Scha92] Ingrid Schaumüller-Bichl. *Sicherheitsmanagement.* BI-Verlag, Mannheim 1992.

[Schn96] Bruce Schneier. *Angewandte Kryptographie.* Addison-Wesley, Bonn, 1. Auflage 1996/1., korrigierter Nachdruck 1997.

[Schr96] Patrick Schramböck. *Sicherheit in Computernetzwerken.* Diplomarbeit, Linz, August 1996.

[SK97a] Bruce Schneier, John Kelsey. Automatic Event Stream Notarization Using Digital Signatures. In *Security Protocols*, pp. 155–169. International Workshop April 1996 Proceedings, Springer-Verlag, Berlin 1997.

[SK97b] Bruce Schneier, John Kelsey. Remote Auditing of Software Outputs Using a Trusted Coprocessor. In *Journal of Future Generation Computer Systems, v. 13, n. 1, 1997*, pp. 9–18.

[SK98] Bruce Schneier, John Kelsey. Cryptographic Support for Secure Logs on Untrusted Machines. In *The Seventh USENIX Security Symposium Proceedings*, pp. 53–62. USENIX Press, Januar 1998.

[SK99] Bruce Schneier, John Kelsey. Secure Audit Logs to Support Computer Forensics. In *ACM Transactions on Information and System Security, v. 1, n. 3, 1999*, to appear.

[SS89] Gunther Schmidt, Thomas Ströhlein. *Relationen und Graphen.* Springer-Verlag, Berlin 1989.

[Sto89] Clifford Stoll. *The Cuckoo's Egg.* Doubleday, New-York 1989.

[Str98] Bjarne Stroustrup. *Die C++-Programmiersprache.* Addison-Wesley, Bonn, 3., aktualisierte und erweiterte Auflage 1998.

[Vol88] Gerhard Vollmer. *Was können wir wissen? Band 1.* Hirzel Verlag, Stuttgart, 2., durchgesehene Auflage 1988.

[Wät94] Dietmar Wätjen. *Theoretische Informatik.* Oldenbourg Verlag, München 1994.

[Wät99a] Dietmar Wätjen. *Automatentheorie und Formale Sprachen.* Vorlesungsskript, Braunschweig 1999.

[Wät99b] Dietmar Wätjen. *Kryptologie.* Vorlesungsskript, Braunschweig, Oktober 1999.

Index