

Grundlagen der Computersicherheit

Stefan Röhrich

Rainer Steinwandt

6. August 2007

Inhaltsverzeichnis

1	Einleitung	4
1.1	Erste grundlegende Frage	4
1.2	Vertraulichkeit	4
1.3	Integrität	5
1.4	Verfügbarkeit	5
1.5	Bedrohungen	5
1.6	Policy and Mechanism	5
2	Access Control	8
2.1	Zugriffsmodell nach Lampson	8
2.2	Harrison-Ruzzo-Ullman-Modell	9
2.3	Bell-LaPadula-Modell	10
2.4	Chinese Wall Model	12
2.5	Access Control in realen Systemen	13
2.5.1	Unix/Linux	13
2.5.2	Windows NT / Windows 2000	14
3	Authentifikation	17
3.1	Typische Anwendung: Kennworte	17
3.1.1	Kennwort im System explizit abspeichern	18
3.1.2	Problem der Wahl guter Kennworte	18
3.1.3	Verhinderung von Online-Angriffen	18
3.2	Alternative Authentifikationsmechanismen	19
3.2.1	Challenge-Response-Verfahren	19
3.2.2	SPEKE	19
3.2.3	Nutzung von Einwegkennworten	21
3.3	Hardware-Unterstützung bei Authentifikationsverfahren	22
3.3.1	Token	22
3.3.2	Zeitbasiert	22
3.3.3	Positionsbasiert	22
3.3.4	Biometrie	22
3.3.5	Gegenseitige Authentifikation	22
3.3.6	Authentifikation mit geringen Hardwaremöglichkeiten	22
4	Schwachstellen in Programmen	23
4.1	Fehler in Webanwendungen	23

5	Netzwerksicherheit	25
5.1	Schichtenmodell	25
5.2	ISO/OSI Security Architecture	25
5.3	TCP/IP	26
5.4	Bedrohungen für Rechner in Netzwerken	26
5.5	Schutzmaßnahmen	27
5.5.1	Firewalls	27
5.5.2	E-Mail-Sicherheit	29
5.6	Key Management	29
5.7	Key Exchange/Key Establishment	29
5.7.1	Key Generation	30
5.8	Public Key Infrastructure (PKI)	30
5.8.1	Merkle Tree-Authentication-Schema	30
5.8.2	SPKI – Simple Public Key Infrastructure	31
5.8.3	PKIX/X.509	32
5.8.4	OpenPGP	33
5.9	Key Escrow	34
6	Sichere Wahlen	36
6.1	Papierbasierte Verfahren	36
6.1.1	ThreeBallot	36
6.1.2	Punchscan	37
6.2	Verfahren mit Wahlmaschinen	38
6.2.1	Vefahren nach Moran und Naor	38
6.2.2	Bingo Voting	40
7	Sicherheitsbewertung	44
7.1	Orange Book	45
7.2	ITSEC	45

Vorwort des Editors

In dieser Fassung besteht das Skript im Wesentlichen aus den Tafelanschriften der Vorlesung „Grundlagen der Computersicherheit“ im Sommersemester 2005 von Rainer Steinwandt und Stefan Röhrich, der von Christian Henrich mit L^AT_EX gesetzt wurde. Weitere Ergänzungen und Korrekturen erfuhr das Skript nach den Vorlesungen im Sommersemester 2006 und 2007 von Stefan Röhrich.

1 Einleitung

Zur Einleitung siehe auch die Folien unter <http://iaks-www.ira.uka.de/home/roehrich/lehre/ss2007/computersicherheit/folien-computersicherheit-ss2007-2007-04-18.pdf>.

1.1 Erste grundlegende Frage

Die erste grundlegende Frage, die sich bei der Betrachtung der Sicherheit eines Systemes stellt, ist, in welcher Art und Weise Anlagen kompromittiert werden können. Eine übliche Einteilung für die Eigenschaften, die gefährdet werden können, ist:

- Vertraulichkeit (*confidentiality*)
- Integrität (*integrity*)
- Verfügbarkeit (*availability*)

Die genauen Definitionen dieser Begriffe variieren und man kann sich über die Vollständigkeit der Liste streiten. Gelegentlich werden Begriffe wie *accountability* (Verantwortlichkeit) oder Authentizität separat aufgeführt.

Das Grunddilemma in der Computersicherheit besteht darin, dass der zentrale Begriff „Sicherheit“ sehr unterschiedlich definiert wird. Es wurde (und wird) viel Zeit darauf verwendet, Sicherheit eindeutig zu definieren, leider ohne einheitliches Ergebnis. Es ist daher in der Regel nicht anzunehmen, dass identische Begriffe in unterschiedlichen Dokumenten identische Bedeutung haben.

1.2 Vertraulichkeit

Das Verheimlichen von Informationen oder Vorenthalten von Ressourcen. Zur Realisierung werden oft Zugriffskontrollmechanismen eingesetzt (*access control*).

Beispiel.

- Verschlüsselung mit geeigneten kryptographischen Verfahren
- Beschränkung der Zugriffsmöglichkeiten auf Dateien/Prozesse/ Rechner/Netze/... für bestimmte Benutzer/Benutzergruppen/Prozesse.

Auch steganographische Verfahren oder das Geheimhalten von Systemkonfigurationen wird unter *confidentiality* gefasst.

1.3 Integrität

Bezieht sich auf die Vertrauenswürdigkeit von Daten und wird meist über unzulässige oder nicht autorisierte Datenveränderung definiert.

Clark & Wilson: *Integrity—No user of the system, even if authorised, may be permitted to modify data items in such a way that assets or accounting records of the company are lost or damaged.*

Orange Book: *Data Integrity—The state that exists when computerised data is the same as that in the source document and has not been exposed to accidental or malicious alteration or destruction.*

Auch Authentizität kann als „Integrität der Quelle“ hierunter gefasst werden, nicht nur „Unversehrtheit der Daten“. Zur Realisierung werden Mechanismen zur Vermeidung (*prevention*) und Entdeckung (*detection*) (von Problemen) angesetzt. Vermeidung heißt sowohl unzulässige (Daten)veränderungen als auch unbefugte (Daten)veränderungen zu vermeiden. Die dazu benötigten Mechanismen sind unterschiedlich.

Integrität zu garantieren benötigt oftmals Annahmen über die Vertrauenswürdigkeit der Herkunft der Daten.

1.4 Verfügbarkeit

Bezieht sich auf die Möglichkeit, Informationen oder Ressourcen in der benötigten Form zu nutzen. Hier können insbesondere Recovery-Mechanismen (z. B. Backup) zum Einsatz kommen, falls ein Angriff erfolgreich war. Dabei sind *Denial of Service*-Angriffe eventuell schwierig von „normalen Abweichungen“ zu unterscheiden. Beispielsweise kann das Blockieren eines Servers dazu führen, dass ein (schlechter geschütztes) Sekundärsystem zum Einsatz kommt.

1.5 Bedrohungen

Unterteilung nach Shirey:

disclosure unbefugter Zugriff auf Informationen

deception Akzeptieren falscher Daten (*repudiation of origin* oder *receipt*)

disruption Unterbrechen oder Verhindern der korrekten Operation

usurpation nichtautorisierte Kontrolle über einen Teil des Systems

1.6 Policy and Mechanism

Die Trennung zwischen Policy und Mechanismus ist essentiell.

Definition. Eine *security policy* ist eine Aussage darüber, was erlaubt ist und was nicht.

Definition. Ein *security mechanism* ist eine Methode, ein Werkzeug oder eine Vorgehensweise zur Durchsetzung (*enforcement*) einer *security policy*.

Beispiel für Mechanismus: Kennwort, Ausweis

Beispiel für Policy: Die (Praktikums-)Policy verbietet das Kopieren von Dateien.

Security policies liegen häufig nur als Fließtext vor, es existieren aber auch formale Ansätze. In formalen Modellen verwendet man gerne „unsichere“ und „sichere“ Zustände mit klar modellierten Übergangsmöglichkeiten.

Ziele der Mechanismen, die eine security policy implementieren:

- Verhinderung (Passwort)
- Entdeckung ($3 \times$ falsches Passwort \rightarrow Warnung)
- Wiederaufsetzen (Backup/juristische Mittel), Zurückfallen in „abgesicherten Modus“

Eine security policy basiert üblicherweise auf Annahmen, die als realisierbar angesehen werden.

Eine security policy geht

1. von einer klaren Trennung in sichere und unsichere Zustände aus, und
2. die Security Mechanismen sollen einen Übergang von einem sicheren in einen unsicheren Zustand ausschließen.

Etwas formaler werden security policies gerne so gefasst:

Definition. Sei P die Menge der betrachteten Systemzustände, Q die Menge der (von der security policy) als sicher spezifizierten Zustände, R diejenigen Zustände, die durch Anwendung eines Sicherheitsmechanismus erreicht werden können. ($Q \subseteq P$, $R \subseteq P$)

$R = Q \rightarrow$ Sicherheitsmechanismus ist exakt.

$R \subseteq Q \rightarrow$ Sicherheitsmechanismus ist sicher.

$R \not\subseteq Q \rightarrow$ Sicherheitsmechanismus ist breit.

Eine security policy charakterisiert die Menge derjenigen Schutzzustände (*protection states*) eines Systems, die als sicher gelten. Schutzzustände beschreiben dabei diejenigen (System-)Zustände (Speicherbelegung, Registerbelegung, ...), die im Kontext des Systemschutzes relevant sind. Die Implementierung und der Betrieb von Sicherheitsmechanismen fasst man gerne im sogenannten *security life cycle* (siehe Abb. 1.1) zusammen.

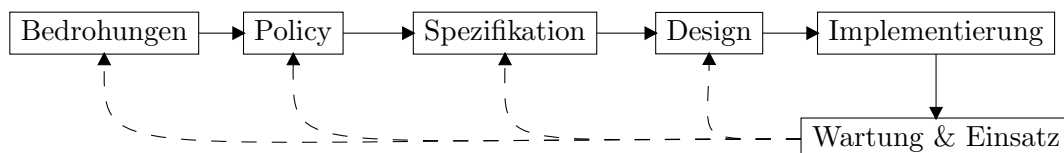


Abbildung 1.1: Security life cycle

Wartung und Einsatz

- Kosten ↔ Nutzen
- Risikoanalyse (Quantifizierung)
- gesetzliche Regelungen, ethische/soziale Probleme
- Eingliederung in organisatorische Abläufe, Alltag → zusätzlicher Verwaltungsaufwand
- social engineering
- Training („will ich, dass“)

2 Access Control

Bei Fragen der Zugriffskontrolle trennt man üblicherweise in Subjekte \mathcal{S} (aktiv), die Zugriffe auf Objekte \mathcal{O} (passiv) haben. Die Art des Zugriffs wird durch eine *access operation* spezifiziert. Üblicherweise betrachtet man \mathcal{S} als Teilmenge von \mathcal{O} , da sich beispielsweise Prozesse gegenseitig aufrufen oder Signale senden können.

2.1 Zugriffsmodell nach Lampson



Abbildung 2.1: Klassisches Zugriffsmodell nach Lampson (1974)

Abhängig vom Anwendungskontext wird für jedes Subjekt spezifiziert, „worauf es wie“ zugreifen kann (*capabilities*), oder es wird spezifiziert, „wer was“ mit einem Objekt machen kann (*access control list*, ACL). Häufig führt man Zwischenschichten ein, um Subjekte oder Objekte zu gruppieren oder Zugriffen nur über definierte Prozeduren zuzulassen. Typisches Beispiel ist rollenbasierter Zugriff.

In theoretischen Modellen unterscheidet man in der Regel zwischen Zugriffen im Modus *alter* („verändern“) oder im Modus *observe* („beobachten“). Das Bell-LaPadula-Modell unterscheidet etwa *execute*, *read*, *write* und *append* (blind write).

Achtung: Bei Bell-LaPadula (siehe Kapitel 2.3) implizieren *write*-Rechte *read*-Rechte.

Ein Beispiel für *execute* ohne *read*: Objekte können genutzt werden, ohne dass sie vom zugreifenden Objekt gelesen werden → Verschlüsselung mit einem kryptographischen Schlüssel in einem „tamper-resistant device“.

Eine Sonderrolle nimmt in der Regel das *own*-Recht ein, welches einem Subjekt gestattet, sich selbst (und anderen) Rechte zu geben oder zu entziehen. Man spricht in diesem Fall von *discretionary access control*. Das Gegenstück hierzu (systemweit festgelegte Rechte) nennt man üblicherweise *mandatory access control*.

Achtung: Gelegentlich (z. B. orange book) werden diese Begriffe auch zur Beschreibung spezifischer security policies genutzt.

Principle of Attenuation of Privilege: Ein Subjekt darf nur solche Rechte vergeben, die es auch selbst besitzt (oder sich selbst geben kann).

	Datei 1	Datei 2	Prozess 1	Prozess 2
Prozess 1	r,w,own	r	r,w,x,own	w
Prozess 2	append	r,own	r	r,w,x,o

	Rechner 1	Rechner 2	Rechner 3
Rechner 1	own	ftp	ftp
Rechner 2		own,ftp,nfs,mail	ftp,nfs,mail
Rechner 3		ftp,mail	own,nfs,mail,ftp

Tabelle 2.1: Beispiele für Access-Control-Matrix

Bemerkung. Im Unterschied zu Privacy-Fragestellungen wird im Access Control normalerweise nicht betrachtet, zu welchem Zweck oder unter welchen Verpflichtungen auf ein Objekt zugegriffen werden darf.

Für theoretische Belange verwendet man gerne eine Access-Control-Matrix (ACM) $M = (M_{so})_{s \in \mathcal{S}, o \in \mathcal{O}}$ mit $M_{so} \subseteq A$, wobei A die Menge der möglichen Zugriffsrechte beschreibt.

Einträge einer Access Control Matrix können sich in Abhängigkeit vorangehend erfolgter Zugriffe verändern.

Beispiel: statistische Datenbank, aus der Informationen über Gruppen von Einträgen ausgelesen werden können, aber nicht auf individuelle Einträge rückgeschlossen werden können soll.

2.2 Harrison-Ruzzo-Ullman-Modell

Hier werden 6 *Grundoperationen (primitive operations)* zur Manipulation der Subjektmenge \mathcal{S} , der Objektmenge \mathcal{O} und der Zugriffskontrollmatrix $M = M_{so}$ bereitgestellt:

- enter r in M_{so}
- delete r in M_{so}
- create subject s
- delete subject s
- create object o
- delete object o

Beispiel: Precondition: $s \notin \mathcal{S}$

Primitive operation: create subject s

Postcondition: $\mathcal{S}' = \mathcal{S} \cup \{s\}$

$$\mathcal{O}' = \mathcal{O} \cup \{s\}$$

$$\forall y \in \mathcal{O}' : M_{sy} = \emptyset, \forall x \in \mathcal{S}' : M_{xs} = \emptyset$$

$$\forall x \in \mathcal{S}, y \in \mathcal{O} : M'_{xy} = M_{xy}$$

Befehle (*commands*) haben die Form

command $c(x_1, \dots, x_k)$

```

    if  $r_1$  in  $M_{s_1o_1}$  and ... and  $r_k$  in  $M_{s_ko_k}$  then
       $op_1; \dots; op_r$ 
    end

```

wobei $x_i \in \mathcal{O} \cup \mathcal{S}$ und op_i Primitivoperationen.

Beispiele für Kommandos im HRU-Modell:

```

command create_file( $s, f$ )
  create object  $f$ ;
  enter own into  $M_{sf}$ ;
  enter read into  $M_{sf}$ ;
  enter write into  $M_{sf}$ ;
end

command grant_read( $s, p, f$ )
  if own in  $M_{sf}$  then
    enter read into  $M_{pf}$ 
  end
end

```

Definition. Existiert für einen Schutzzustand (eine ACM) ein Kommando c , welches ein Recht r an einer Position der ACM hinzufügt, bei der r zuvor nicht enthalten war, so „entweicht“ (*leaks*) das Recht. Es existiert also $s \in \mathcal{S}, o \in \mathcal{O}$ mit $r \notin M_{so}$ und $M_{so} \xrightarrow{c(x_1, \dots, x_n)} M'_{so}$ mit $r \in M'_{so}$.

Definition. Ein Zustand/eine ACM heißt „sicher“ (*safe*) bzgl. des Rechtes r , falls keine Kommandosequenz M in einen Zustand überführen kann, in dem das Recht r entweicht.

Um im HRU-Modell zu prüfen, ob ein (*authorization*) System mit einer security policy konform ist, sind *safety*-Eigenschaften zu verifizieren.

Satz. Für monooperationale Systeme kann zu gegebenen M und r die *safety* von M bezüglich r algorithmisch entschieden werden. Das *safety*-Problem ist ebenfalls entscheidbar, falls keine create-Befehle zugelassen werden oder es nur endlich viele Subjekte gibt.

Es gibt verschiedene Einschränkungen des Modells, wie monooperationale Systeme, so dass das Problem, ob eine vorgegebene ACM sicher ist, algorithmisch lösbar ist. Aber im Allgemeinen gilt:

Zu gegebener ACM M und Recht r ist es *nicht entscheidbar*, ob M sicher bzgl. r ist. (Beweis durch Reduktion auf das Halteproblem von Turingmaschinen)

Die Aufgabe besteht nun darin, ausdrucksstarke und gleichzeitig „handhabbare“ Modelle zu finden.

Näheres zur (Un-)Entscheidbarkeit des HRU-Modells kann man in den Folien zur entsprechenden Vorlesung unter <http://iaks-www.ira.uka.de/home/roehrich/lehre/ss2007/computersicherheit/folien-computersicherheit-ss2007-2007-04-25.pdf> nachlesen.

2.3 Bell-LaPadula-Modell

Ein „Klassiker“: Das Modell von Bell-LaPadula

Das Modell von Bell-LaPadula ist ein Zustandsmaschinenmodell zur Modellierung von Confidentiality-Aspekten im Access Control und entstand im Rahmen der Bemühungen, ein sicheres Multi-User-Betriebssystem zu entwickeln.

Den Ausgangspunkt für das Modell bilden

- Subjektmenge \mathcal{S} ,
- Objektmenge \mathcal{O} ,
- Menge der Zugriffsoperationen $\mathcal{A} = \{read, write, append, execute\}$ und
- Menge L von *security levels* mit einer partiellen Ordnung \leq .

Dabei implizieren *write*-Rechte *read*-Rechte.

Beispiel. Lineare Sicherheitsstufen $unclassified \leq confidential \leq secret \leq top\ secret$

Gerne verwendet man für L eine Verbandstruktur (zu je 2 Elementen existiert kleinste obere und größte untere Schranke).

Beispiel. Etwa $\mathcal{P}\{w, r, x\}$ mit \subseteq als partielle Ordnung. Dabei ist eine Sicherheitsstufe $a \in \mathcal{P}\{w, r, x\}$ eine Menge von Berechtigungen.

Für „ $a \subseteq b$ “ sagt man auch „ a wird von b dominiert“.

Bell-LaPadula modelliert den Systemzustand als Element aus $\mathcal{B} \times \mathcal{M} \times \mathcal{F}$. Dabei ist $\mathcal{B} = \mathcal{P}(\mathcal{S} \times \mathcal{O} \times \mathcal{A})$ die Menge der aktuellen Zugriffe, \mathcal{M} die Menge der Zugriffskontrollmatrizen (ACMs) mit $M \in \mathcal{M}$, $M = (M_{so})_{s \in \mathcal{S}, o \in \mathcal{O}}$ und $\mathcal{F} \subseteq L^{\mathcal{S}} \times L^{\mathcal{S}} \times L^{\mathcal{O}}$ die Menge der „security level assignments“.

Sei etwa $(f_s, f_c, f_o) \in \mathcal{F}$

$f_s : \mathcal{S} \rightarrow L(\in L^{\mathcal{S}})$ beschreibt den maximalen Sicherheitslevel eines Subjekts

$f_c : \mathcal{S} \rightarrow L(\in L^{\mathcal{S}})$ beschreibt den gegenwärtigen Sicherheitslevel eines Subjekts

$f_o : \mathcal{O} \rightarrow L(\in L^{\mathcal{O}})$ beschreibt den Sicherheitslevel eines Objekts

Dabei gilt, f_c muss von f_s dominiert werden, d. h. die aktuelle Einstufung darf nicht höher als die maximale sein.

Definition (ss-Eigenschaft, simple security property). Ein Zustand (b, M, f) genügt der *ss-Eigenschaft*, falls für jedes $(s, o, a) \in b$ mit $a \in \{read, write\}$ folgendes erfüllt ist:

$$f_s(s) \geq f_o(o) \quad (\text{no read up}).$$

Definition (*-Eigenschaft, star property). Ein Zustand (b, M, f) erfüllt die **-Eigenschaft*, falls für jedes $(s, o, a) \in b$ mit $a \in \{append, write\}$ folgendes erfüllt ist:

$$f_c(s) \leq f_o(o) \quad (\text{no write down}).$$

Weiterhin, falls ein $(s, o, a) \in b$ mit $a \in \{append, write\}$ existiert, dann muss $f_o(o') \leq f_o(o)$ gelten, wenn immer $(s, o', a') \in b$ und $a' \in \{read, write\} \rightarrow$ kein Nachrichtentransfer von „high level subject“ zu „low level subject“.

Es gibt zwei Lösungsansätze, um das Problem, das durch die Verhinderung jeglichen Informationsflusses von einem „high level subject“ zu einem „low level subject“ entsteht, zu lösen:

- befristeter Downgrade via f_c
- „trusted subjects“, die die *-Eigenschaft verletzen dürfen

Bemerkung. Ein „trustworthy“ Subjekt ist nicht das gleiche wie ein „trusted“ Subjekt. Subjekte, die als „trustworthy“ bezeichnet werden, brechen keine security policy, „trusted“ Subjekte hingegen können eine security policy brechen.

Definition (ds-Eigenschaft, discretionary security property). Die *ds-Eigenschaft* wird von (b, M, f) erfüllt, falls für $(s, o, a) \in b$ stets $a \in M_{so}$ gilt.

Definition. Wenn die drei Eigenschaften ss, *- und ds-Eigenschaft, gelten, nennen wir ein System „sicher“ (*secure*).

Satz (Basic Security Theorem). *Werden ausgehend von einem sicheren Initialzustand nur sichere Übergänge durchgeführt, erhält man einen sicheren Systemzustand.*

Probleme mit BLP:

- Das BLP-Modell berücksichtigt keine verdeckten Kanäle (z. B. Existenz/Nichtexistenz von Dateien).
- Das BLP-Modell erlaubt „sinnlose“ Übergänge. (MacLean 1987: Die Vergabe aller Rechte auf alle Objekten and alle Subjekte verletzt Sicherheitsdefinition des Modells nicht.)

Vorteile von BLP:

- Das BLP-Modell fasst Sicherheitsfragestellungen als Access Control Problem und macht sie damit handhabbar.
- Bei (eingeschränkten) Betriebssystemen und Datenbankanwendungen ist eine befriedigende Modellierung möglich.

2.4 Chinese Wall Model

Anders als beim Bell-LaPadula-Modell hängen Zugriffsrechte beim Chinese Wall Model (Brewer, Nash, 1989) von der Vergangenheit ab.

Motivation: Vermeidung von Interessenskonflikten bei Unternehmensberatung. Es darf keinen Informationsfluss geben, der Interessenskonflikte verursacht.

„Bestandteile“:

- Menge von Firmen \mathcal{C}
- zu jeder Firma ex. eine Menge von Objekten, deren Gesamtheit \mathcal{O} ist.
- Die Beratermenge bildet die Menge der Subjekte \mathcal{S} .

- $y : \mathcal{O} \rightarrow \mathcal{C}$ ordnet jedem Objekt die zugehörige Firma (*company data set*) zu.
- $x : \mathcal{O} \rightarrow \mathcal{P}(\mathcal{C})$ liefert die „conflict of interest class“ eines Objektes (wohin sollen keine Informationen über ein Objekt fließen).
- Sicherheitsmarke (*security label*) eines Objektes o : $(x(o), y(o))$
- „sanitised information“ $x(o) = \emptyset$ (keine kritischen Informationen enthalten)

Interessenskonflikte können aus vergangenen Zugriffen resultieren. Um dies modellieren zu können, verwendet man eine Matrix

$$M = (M_{so})_{s \in \mathcal{S}, o \in \mathcal{O}} \text{ mit } (M_{so}) = \begin{cases} true & , \text{ falls } s \text{ Zugriff auf } o \text{ hatte} \\ false & , \text{ sonst (} s \text{ hatte nie Zugriff auf } o \text{)} \end{cases}$$

Sicherer Initialzustand $M = (false)_{s \in \mathcal{S}, o \in \mathcal{O}}$, kein Konflikt der Subjekte.

Definition (ss-Eigenschaft, simple security property). $s \in \mathcal{S}$ erhält Zugriff auf $o \in \mathcal{O}$ nur, falls für alle o' mit $M_{so'} = true$ folgendes gilt: $y(o) = y(o')$ oder $y(o) \notin x(o')$.

Definition (*-Eigenschaft). $s \in \mathcal{S}$ erhält Schreibzugriff auf Objekt $o \in \mathcal{O}$ nur, falls s keinen Lesezugriff auf ein Objekt o' hat mit $y(o) \neq y(o')$ und $x(o') \neq \emptyset$.

2.5 Access Control in realen Systemen

2.5.1 Unix/Linux

Unter Unix gehören Dateien/Devices einem Benutzer und einer Gruppe. Darauf basierend werden Rechte für diesen Benutzer (*user/owner*), die Gruppe (*group*) und den Rest (*others*) vergeben. Bei gleicher uid (*user id*) haben die Userrechte Wirkung, ist die effektive gid (*group id*) des Prozesses gleich, die Gruppenrechte, andernfalls other. Es gibt jeweils folgende Rechte:

r	Read	Lesen
w	Write	Schreiben
x	eXecute	Ausführen/Zugriff auf Verzeichnisse

Zusätzlich gibt es folgende Kennzeichnungen:

suid	set user id on execution	Prozess läuft mit Dateibesitzer als neuer uid
sgid	set group id on execution	Prozess läuft mit Gruppenrechten der Datei
t	„sticky bit“	früher Hinweis an Speicherverwaltung → Datei wird häufig benutzt, bitte im Speicher lassen

Besonderheiten bei Verzeichnissen

Die Rechte bei Verzeichnissen bestimmen das Lesen (Anzeigen, ls) von Verzeichnissen, das Schreiben (Anlegen und Löschen) und Zugriff auf Dateien. Insbesondere können auch Dateien

eines anderen Users gelöscht werden, wenn man Schreibrechte auf das Verzeichnis hat (unabhängig von den Rechten auf die Datei). Rechteänderungen sind aber nur dem Besitzer möglich. `chmod` hat keine Bedeutung, `sgid` heißt meistens, dass in diesem Verzeichnis angelegte Dateien der Gruppe des Verzeichnisses gehören. Das „sticky bit“ wird oft verwendet, um zu kennzeichnen, dass nur der Besitzer einer Datei diese löschen kann (z. B. `/tmp`).

Besonderheiten/Ausnahmen:

- Hard links beim Löschen beachten
- Root darf alles!
- Filesystemspezifische Attribute (z. B. *immutable*, *append only*)
- ACLs: rwx-Angaben für User und Gruppen, wieder hat User Vorrang, bei Gruppen genügt es, wenn eine Gruppe entsprechende Rechte hat. Zusätzlich gibt es eine Maske zum Überschreiben der Rechte.
- Capabilities: spezifische Rechte, Abschwächung von root.

Weitere Informationen finden sich in [Gol05] Kapitel 6.4, [man1], [man2] und [man3].

2.5.2 Windows NT / Windows 2000

Auf NTFS-Dateisystemen (und in der Registry) besteht die Möglichkeit, Zugriffsbeschränkungen durchzuführen. Dazu haben Dateien einen Besitzer, der die Rechte der Datei bearbeiten darf, diese werden in ACLs gespeichert. Diese besteht aus folgenden Einträgen:

- SID (*Security Identifier*): User/Gruppe, auf die sich der Eintrag (ACE) bezieht.
- Typ: Allow/Deny
- Rechte

Diese Access Control Entries sollten in der Reihenfolge zuerst Deny, dann Allow sortiert sein und werden beim Zugriff der Reihenfolge nach abgearbeitet, bis das benötigte Recht auftaucht und für den Betroffenen entweder erlaubt oder verweigert wird. Trifft kein Eintrag zu, wird der Zugriff verweigert. Optional können auch Rechte vererbt werden. Dann wird bis zur Wurzel evaluiert, ob Rechte zutreffen.

Es können sehr detaillierte Berechtigungen vergeben werden, vereinfacht durch Standardbelegungen. „Vollzugriff“, „Ändern“, „Lesen und Ausführen“, „Ordnerinhalt auflisten“, „Lesen“ und „Schreiben“.

Zu beachten ist dabei:

- Vererbung
- Attribute (z. B. schreibgeschützt, versteckt)
- Besitzer einer Datei dürfen Berechtigungen immer ändern

- Vollzugriff auf Verzeichnisse bedeutet Löschmöglichkeit
- Gruppe „Jeder“ sind nicht alle Benutzer (z. B. nicht „Anonymous Logons“)
- Privileg „Bypass traverse checking“ erlaubt Zugriff auf Dateien in einem Verzeichnis, auch wenn der Benutzer das Recht „Ordner durchsuchen“ nicht hat. Dieses Privileg haben standardmäßig selbst normale Benutzeraccounts.
- Privilegien können mehr erlauben (z. B. für Backup)
- Sonderbehandlungen für Administratoren

Weitere Informationen finden sich in [Gol05] Kapitel 7.4, [Bis02] Kapitel 15.1.4, [Mic] und [Cyg].

Spezielle Berechtigungen	Vollzugriff	Ändern	Lesen und Ausführen	Ordnerinhalt auflisten	Lesen	Schreiben
Ordner durchsuchen/Datei ausführen	Ja	Ja	Ja	Ja	Nein	Nein
Ordner auflisten/Daten lesen	Ja	Ja	Ja	Ja	Ja	Nein
Attribute lesen	Ja	Ja	Ja	Ja	Ja	Nein
Erweiterte Attribute lesen	Ja	Ja	Ja	Ja	Ja	Nein
Dateien erstellen/Daten anhängen	Ja	Ja	Nein	Nein	Nein	Ja
Attribute schreiben	Ja	Ja	Nein	Nein	Nein	Ja
Erweiterte Attribute schreiben	Ja	Ja	Nein	Nein	Nein	Ja
Unterordner und Dateien löschen	Ja	Nein	Nein	Nein	Nein	Nein
Löschen	Ja	Ja	Nein	Nein	Nein	Nein
Berechtigungen lesen	Ja	Ja	Ja	Ja	Ja	Ja
Berechtigungen ändern	Ja	Nein	Nein	Nein	Nein	Nein
Besitzrechte übernehmen	Ja	Nein	Nein	Nein	Nein	Nein
Synchronisieren	Ja	Ja	Ja	Ja	Ja	Ja

Wichtig: Gruppen oder Benutzer mit der Berechtigungsstufe Vollzugriff für einen Ordner können Dateien in diesem Ordner löschen. Dies gilt unabhängig von der Berechtigung, die zum Schutz der Dateien festgelegt wurden.

3 Authentifikation

Definition. Authentifizierung bindet eine Identität an ein Subjekt.

Ansätze, mit denen eine (System-externe) Entität dem System die Identitätsprüfung ermöglicht:

- Die Entität *weiß* etwas (Kennwort, geheime Information).
- Die Entität *besitzt* etwas (Karte, Marke).
- Die Entität *ist* etwas (Iris-Scan, Fingerabdrücke).
- Die Entität *kann* etwas (Rechenleistung, Verschlüsselung).
- Die Entität *befindet sich* an einer bestimmte Stelle (GPS-Koordinaten, festes Terminal).

Ein Authentifikationssystem besteht aus 5 Komponenten

- Menge \mathcal{A} der *Authentifikationsinformation*. Die Information, mit der die Identität bewiesen wird.
- Menge \mathcal{C} der *Komplementärinformation*, die das System speichert, um die Authentifikationsinformation zu validieren.
- Menge \mathcal{F} der *Komplementierungsfunktionen*, die aus Authentifizierungsinformation die zugehörige Komplementärinformation ableitet. Dabei ist $\mathcal{F} \subseteq \mathcal{C}^{\mathcal{A}}$.
- Menge \mathcal{L} der *Authentifikationsfunktionen*, die die Identifikation verifizieren. Dabei ist $\mathcal{L} \subseteq \{\text{true}, \text{false}\}^{\mathcal{C} \times \mathcal{A}}$.
- Menge \mathcal{S} der *Auswahlfunktionen*, die einer Entität das Anlegen, Ändern oder Entfernen der Authentifikationsinformation und Komplementärinformation erlauben.

3.1 Typische Anwendung: Kennworte

Definition. Ein Kennwort ist eine Information, die mit einer Entität assoziiert ist und deren Identität bestätigt.

3.1.1 Kennwort im System explizit abspeichern

(Ein) Problem: Wird die entsprechenden Komplementärinformation offengelegt, sind damit sämtliche Kennworte kompromittiert.

Lösung: Statt dem Passwort wird ein Hashwert des Passworts unter einer von mehreren (z. B. 4096) möglichen Hashfunktionen und das *salt* (Funktionsbeschreibung) gespeichert.

Wörterbuchangriffe: Trial & Error-Ansatz zum Erkennen von Kennworten

Typ 1: Komplementärinformation ist bekannt, ebenso die Komplementierungsfunktion (*offline*)

Bsp.: Passwortraten bei bekanntem passwd-File

Typ 2: Komplementärinformation oder Komplementierungsfunktion unbekannt

Bsp.: Einloggen bei bekanntem Accountnamen

Für jeden Typ existieren entsprechende Schutzmechanismen.

3.1.2 Problem der Wahl guter Kennworte

Zufallsstrings erschweren Wörterbuchangriffe, sind aber im Allgemeinen nicht vernünftig zu merken.

Lösungsansätze:

„**key crunching**“: Verwendung langer Schlüsselbegriffe, die mit einer kryptologischen Hashfunktion „komprimiert“ werden (MD5, SHA-256)

Verschleierung aufgeschriebener Kennworte: Anwendung einer einfachen (häufig wechselnden) Transformationsregel auf aufgeschriebene Kennworte

proaktive Kennwortwahl: Versuch bereits bei der Kennwortwahl schwache Kandidaten zu identifizieren

zeitliche Variation des Kennwortes: Hat den Nachteil, dass eventuell leicht erratbare Varianten gewählt werden, z. B.: *Anna01*, *Anna02*, ...

3.1.3 Verhinderung von Online-Angriffen

Backoff-Technik: Bei *exponential backoff* wartet das System nach n Fehleingaben x^{n-1} Sekunden vor Akzeptieren der nächsten Eingabe (x Systemparameter) (kann für DoS-Angriffe ausgenutzt werden)

Disconnection: Abbruch (ggf. selektiv) der Netzverbindung nach n Fehleingaben

Disabling: Blockieren des Accounts nach n Versuchen (kann für DoS-Angriffe ausgenutzt werden)

Jailing: Begrenzter (kontrollierter) Zugriff wird (trotz fehlerhafter Authentifikation) gewährt und überwacht. Oftmals mit Vorgabe vermeintlich attraktiver Ziele (*honey-pots*) und unter Vorspiegelung falscher Systemparameter, z. B. langsame Netzwerke

CAPTCHAs: (Referenz: www.captcha.net) Die Idee ist hier, menschliche Benutzer von Skripten zu trennen – nur menschliche Benutzer sollen den Test bestehen, dies aber mit *sehr* geringem Aufwand. Um damit Kennworte zu schützen, müssen sowohl das CAPTCHA richtig gelöst als auch das Kennwort korrekt sein.

Wesentliche Anforderung an ein CAPTCHA (*Completely Automated Public Turing test to tell Computers and Humans Apart*): Die Rätsel müssen automatisch generiert und verifiziert werden können. Gängig ist die Verfremdung von Texten durch variierende Fonts, wechselnde Farben, übereinandergeschriebene Texte, die erkannt werden müssen. Ein Hauptproblem bei diesen Ansätzen ist der Ausschluss von Benutzergruppen (z. B. Blinde).

3.2 Alternative Authentifikationsmechanismen

3.2.1 Challenge-Response-Verfahren

In einer Registrierungsphase erhält der Benutzer ein Geheimnis s , welches die korrekte Beantwortung von Fragen (*challenges*) erlaubt.

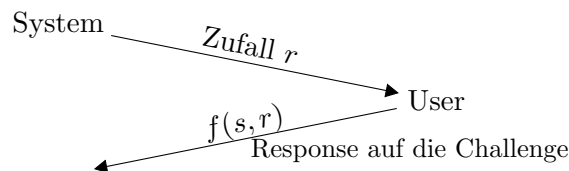


Abbildung 3.1: Grundablauf Challenge-Response

Beachte: Es darf nicht möglich sein, anhand abgehörter Authentifizierungen das Geheimnis s zu rekonstruieren.

3.2.2 SPEKE (Simple Password Exponential Key Exchange)

Ein Protokoll, welches eine Authentifikation mit einem Geheimnis s geringer Entropie erlaubt und dabei auch einen gemeinsamen Schlüssel zwischen System und User etabliert, ist SPEKE (siehe [Jab96]).

SPEKE kann in zwei Schritte unterteilt werden, wobei im ersten (authentifiziert) ein DH-Schlüsselaustausch vorgenommen wird und im zweiten Teil wird die Kenntnis des Schlüssels bestätigt.

Öffentlich ist eine zyklische Gruppe G , die geeigneten kryptologischen Annahmen genügt und als Erzeuger der Gruppe wird ein Passwort-abh. Element gewählt.

Eine typische Konstruktion: G als Menge der quadratischen Reste von p (G Untergruppe von \mathbb{Z}_p^\times der Ordnung q), wobei $p = 2q + 1$ mit p und q Primzahlen ($p, q \in \mathbb{P}$). Dabei wird $g := H(\text{password})^2 \bmod p$ als Quadrat des Hashwertes gewählt, um sicherzustellen, dass g ein

quadratischer Rest ist (also $g \in G$). Da G von der Ordnung $q \in \mathbb{P}$ ist, erzeugt g die Untergruppe G .

SPEKE (*Simple Password Exponential Key Exchange*)

Grundsätzlicher Ablauf in zwei Phasen

1. Etablierung des gemeinsamen Schlüssels unter Verwendung des gemeinsamen Kennwortes (siehe Abb. 3.2)
2. Bestätigung der Kenntnis des Schlüssels und damit Verifikation der korrekten Passwortkenntnis (siehe Abb. 3.3)

Typische Realisierung auf DH-Basis: Öffentlich ist eine „safe prime“ $p = 2q + 1, p, q \in \mathbb{P}$.

In Phase 1 wird ein DH-Austausch mit Kennwort-abhängigem Erzeuger der Untergruppe (der quadratischen Reste) mit der Ordnung q von \mathbb{Z}_p^\times gemacht. Sei hierzu π das gemeinsame Kennwort, H eine kryptographische Hashfunktion (wie z. B. SHA-256).

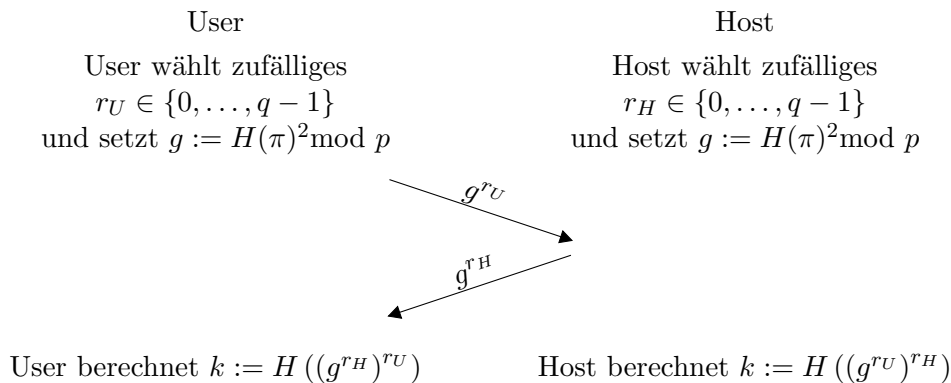


Abbildung 3.2: Phase 1

Die Anwendung der Hashfunktion H bei der Berechnung des Schlüssels soll sicherstellen, dass das Gruppenelement $g^{r_H r_U}$ zu einem sicheren Schlüssel bekannter Länge und hoher Entropie wird. Werden nur bestimmte Bits der Gruppenelementdarstellung genommen, könnten diese leicht zu erraten sein.

SPEKE erlaubt einen (effizienten) Schlüsselaustausch, ohne dass ein Offline-Wörterbuchangriff auf das Kennwort π ermöglicht wird. Abhängig von den konkreten Anwendungsanforderungen werden die einzelnen Phasen unterschiedlich gestaltet.

Varianten wie in Abb. 3.4 sind unter Berücksichtigung moderner Sicherheitskriterien für *key establishment* entworfen, wo gefordert wird, dass ein Angreifer den korrekten Schlüssel nicht von einem Zufallsstring unterscheiden können darf. Um solche Probleme zu umgehen, kann man zur Verifikation des Besitzers des Schlüssels einen unabhängigen Hilfwert berechnen:

$$\text{session_key} := H(k\|0)$$

$$\text{control_key} := H(k\|1)$$

Das vom Benutzer eingegebene Kennwort wird jeweils gehasht und bei Übereinstimmung von Hashwert und gespeichertem Wert wird der Zugang gewährt und das korrekte Kennwort ersetzt den gespeicherten Wert.

3.3 Hardware-Unterstützung bei Authentifikationsverfahren

3.3.1 Token

Der User erhält eine challenge und gibt diese (und evtl. ein Kennwort) in ein Gerät ein und erhält Antwort auf die challenge von dem Gerät.

3.3.2 Zeitbasiert

Alle z. B. 60 Sekunden wird eine Zahl $[0 \dots 10^n - 1]$ angezeigt. Das System, bei dem die Anwendung erfolgt, hat entsprechendes Gerät zur Verifikation. Zur Anmeldung ist (Einweg-)Kennwort und aktuelle Nr. erforderlich.

3.3.3 Positionsbasiert

Spezialhardware kombiniert Zeit, Geheimnis und GPS-Position.

3.3.4 Biometrie

Gängig sind Fingerabdruckerkenner (optisch, elektr. Widerstand), Stimmerkennung (kombinierbar mit *pass phrase*), Iriserkennung, Retina-Scan (Blutgefäße im Augenhintergrund), Gesichtserkennung (Infrarot-Temperaturverteilung, optisch)

3.3.5 Gegenseitige Authentifikation

Zur Vermeidung von „Austausch“-Angriffen muß sich auch das Gerät beim Benutzer authentifizieren, Beispiel Passmaze-Protokoll [[Bro05](#)].

3.3.6 Authentifikation mit geringen Hardwaremöglichkeiten

Auch für sehr kleine Chips wie z. B. RFID-Tags gibt es spezielle Authentifikationsprotokolle, etwa HB+ [[JW05](#)].

4 Schwachstellen in Programmen

Zu sicheren Computersystemen gehören nicht nur Sicherheitsmechanismen, sondern auch konkrete Implementierungen. Bestimmte Fehler werden häufig gemacht. Treten diese z. B. in SUID-Programmen, Netzwerkdiensten oder Systemfunktionen auf, kann dies Auswirkungen auf die Sicherheit des gesamten Systems haben. Typische Fehler (vor allem bei hardwarenahen Programmiersprachen wie C) sind z. B.:

Buffer Overflow

Stack Overflow: Überschreiben der Rücksprungadresse (oder anderer Daten) und dadurch Ausführen ungewollten Codes

Heap Overflow: Überschreiben von Datenfeldern, die an das eigentlich zu ändernde Feld angrenzen, dadurch z. B. setzen anderer Passwörter, Zerstörung der Strukturen des Speichermanagement und dadurch Ausführen von Code.

Integer Overflow und Signed-Unsigned-Bugs: schwer zu entdeckende Fehler, die durch Überschreiten des Wertebereichs/falscher Vorzeicheninterpretation z. B. falscher Puffergrößen oder Übergehen von (Sicherheits-) Abfragen ermöglicht.

falsche Unicode-Behandlung: Durch falsche Puffergrößen, fehlendes Markieren von Sonderzeichen oder Anzeige „ähnlicher“ Zeichen können Buffer Overflows, „Terminalattacken“ oder „falsche“ Benutzerreaktionen ausgelöst werden.

Signalhandling: Ausnutzen von *race conditions*.

Temporäre Dateien/Links: Überschreiben von Dateien oder Benutzern durch geschicktes Setzen von Links, auf die dieser Benutzer schreibt.

chroot: Wird chroot ohne Vorsicht als Sicherheitsfeature verwendet, wird oft übersehen, dass sich daraus (manchmal) ausbrechen lässt.

shell-escapes: Ermöglicht u. U. Shell in SUID-Anwendungen.

Umsetzen von Zeigern (Hardware: Fön): Ausführen falschen Programmcodes.

Sonderzeichenausgabe: unerwartete Anzeige führt zu Terminalproblemen oder Benutzeraktionen → richtiges Überprüfen aller Ein- und Ausgaben notwendig.

4.1 Fehler in Webanwendungen

SQL-Injection: ungenügende Eingabeüberprüfung ermöglicht direkten Zugriff auf die Datenbank (oder mehr)

XSS-Bugs: Einfügen von HTML-/(JavaScript-)Code → Cookie-Klau

File Upload: ../../etc/passwd als Pfad für Datei

PHP: Auto Globals/URL-Include führt zum Ausführen von beliebigem Code

POST ist von Hand möglich, keine Sicherheitsmaßnahme

Falsche Annahmen über Umfang der Sicherheit sollten vermieden werden (Beispiel Federated-Identity-Management-Protokoll).

5 Netzwerksicherheit

Der Datentransport darf aus dem Blickwinkel der Sicherheit nicht bloß abstrakt betrachtet werden, sondern das verwendete Netzwerk ist zu berücksichtigen. Einzelheiten wie Umordnung, Verlust von Paketen, Adressierung, Routing usw. können die Sicherheit stark beeinflussen, hinzu kommen andere Angriffe auf Rechner.

5.1 Schichtenmodell

Zur einfacheren Beschreibung wird oft das ISO/OSI (ISO Open Systems Interconnections)-Schichtenmodell verwendet.

Application
Presentation
Session
Transport
Network
Link
Physical

Tabelle 5.1: ISO/OSI-Schichtenmodell

Für den Protokollentwickler gibt es dann virtuelle Verbindungen auf Ebene N. Die unteren Ebenen werden abstrahiert und kapseln die Daten der übergeordneten Schicht beim Senden bzw. packen sie beim Empfangen aus.

5.2 ISO/OSI Security Architecture

Die ISO/OSI Security Architecture definiert folgende Dienste, die meist mit kryptographischen Mitteln erreicht werden sollen:

Vertraulichkeit

Integrität

Data origin authentication: Urheber

Peer-entity authentication: verifiziert die Identität eines Kommunikationspartners in der selben Protokollschicht

Non-repudiation: Nichtabstreitbarkeit:

- proof of origin
- proof of delivery/proof of submission
- proof of receipt

Ein sicheres Protokoll soll nicht durch unsichere tiefere Schichten gefährdet werden (Ausnahme: Anonymität, hier kann dies leider vorkommen).

5.3 TCP/IP

Application	HTTP, SMTP
Transport/Session	TCP, UDP
Internet	IP
Interface	Hardware

Tabelle 5.2: Vereinfachtes Schichtenmodell für TCP/IP

IP-Datenverkehr wird auf unterschiedliche Arten geschützt, meistens auf Applikationsebene (z. B. SSL/TLS, aber auch damit oft „Tunneling“ möglich). Eine andere Möglichkeit auf „Internet“-Schicht ist IPSEC, damit werden Möglichkeiten zur Authentifikation (IP authentication header AH) und Verschlüsselung (IP encapsulated security payload ESP) geboten. Außerdem werden Schlüsselmanagementmechanismen und Methoden zur Vereinbarung von security policies für IP-Verbindungen bereitgestellt. Neben dem Transport der Daten gibt es bei IPSEC auch die Option des Tunneling, z. B. für Virtual Private Networks. Näheres findet sich in den Folien zur Vorlesung unter <http://iaks-www.ira.uka.de/home/roehrich/lehre/ss2007/computersicherheit/folien-computersicherheit-ss2007-2007-06-27.pdf>.

5.4 Bedrohungen für Rechner in Netzwerken

- Belauschen/Unterdrücken/Verändern der Daten
- „Portscans“: Ermitteln der Dienste
- „Fingerprinting“: Ermitteln der OS-Version u. ä. für gezielte Angriffe
- Angriffe auf Routing
- Schwachstellen in anderen Protokollen, die für den Netzwerkverkehr benötigt werden (z. B. Namensauflösung/DNS)
- Denial of Service
- Angriffe von innen
- automatische Scans auf Schwachstellen

- Viren/Würmer
Arten von Viren auf Clients:
 - früher: Bootsektorviren, EXE/COM-Viren
 - Trojaner/Dialer
 - Makroviren/E-Mail-Würmer
 - Server-Würmer, die Schwachstellen ausnutzen (IIS/SQL-Server)
- Backdoors

5.5 Schutzmaßnahmen

5.5.1 Firewalls

Sogenannte „bastion hosts“, besonders gesicherte Rechner, die Netzwerke von einander trennen, z. B. „sichere“ Firmennetzwerke und unsicheres Internet. Auf den Firewalls sollten nur die absolut notwendigsten Dienste laufen und so wenig Benutzer wie möglich Zugang haben (im Normalfall nur der Administrator).

Verschiedene Arten von Firewalls:

- Paketfilter: reines Aussortieren nach Adresse/Dienst o. ä.
- Stateful Inspection: Berücksichtigen der Verbindungsinformationen
- Proxy/Application Level: eigene Applikationen zur Protokollumsetzung
- NAT: network address translation
- Personal Firewalls auf Clients

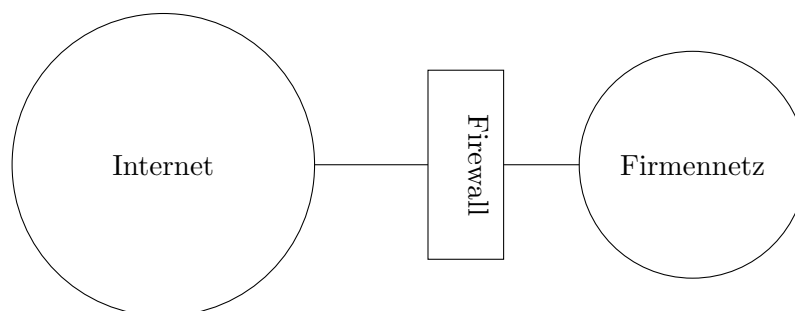


Abbildung 5.1: Firewall bei einfacher Netzstruktur

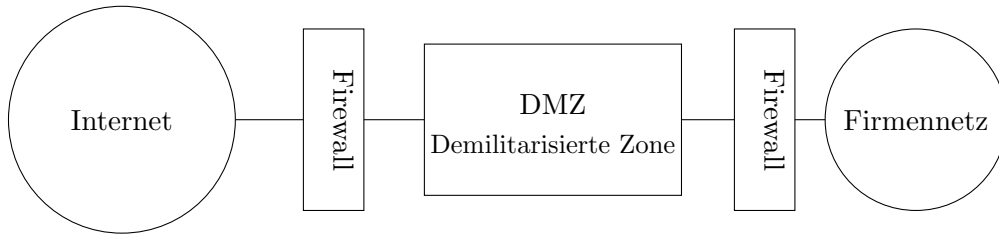


Abbildung 5.2: Firewallstruktur mit Servern nach „außen“

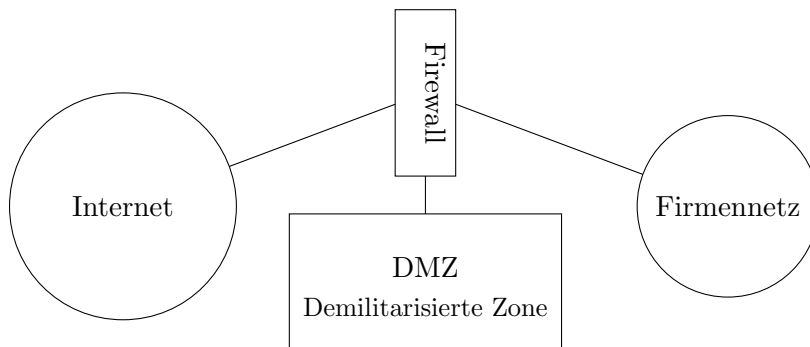


Abbildung 5.3: Variante mit einer Firewall

5.5.2 E-Mail-Sicherheit

Wünsche:

- Anonymität
- Integrität
- Vertraulichkeit/Geheimhaltung
- nicht fälschbare Absenderadresse
- kein Spam
- Authentizität
- Auslieferungsgarantie
- „funktionierender“ Client
- keine Viren/Würmer
- Überwachbarkeit

5.6 Key Management

Key Management bezieht sich auf die Verteilung kryptographischer Schlüssel, sowie zugehöriger Erstellungs-, Wartungs- und Widerrufsmechanismen und die Zuordnung Identität ↔ Schlüssel.

Ein *Interchange Key* ist ein kryptographischer Schlüssel, der einer Person zugeordnet ist.

Ein *Session Key* ist ein Schlüssel für eine bestimmte Kommunikationsverbindung.

Interchange Keys sind langlebiger und werden u. U. auch zur Authentifikation und unabhängig vom Start oder Ende einer Kommunikationsverbindung benutzt. Session Keys hingegen werden pro Verbindung erneut gewählt, dadurch wird immer ein frischer Schlüssel verwendet, z. B. Replay-Angriffe (*replay attacks*) werden erschwert.

5.7 Key Exchange/Key Establishment

klassisch: Trusted Third Party (z. B. Kerberos)

Public Key: Key Transport, verschlüsselt mit Public Key (einfach), Diffie-Hellman

Authenticated Key Exchange [MQV]

Password Based Key Exchange: SPEKE

Key Confirmation ist oft noch zusätzlich sinnvoll, damit wird sichergestellt, dass jeder Teilnehmer den richtigen Schlüssel erhalten hat.

5.7.1 Key Generation

Zufallserzeugung ist problematisch, Lösung: Hardware anstatt Pseudozufallszahlengenerator, z. B. Quanteneffekte, Radioaktivität

Notbehelf: Tastatur-/Mauseingaben, Netzwerke o. ä.

5.8 Public Key Infrastructure (PKI)

„Definition“: Ein (*digitales*) *Zertifikat* ordnet einem öffentlichen Schlüssel eindeutig einen Inhaber zu.

Eine *Public Key Infrastructure (PKI)* ist die Gesamtheit der organisatorischen Maßnahmen, die für die vertrauenswürdige Ausgabe und Verifikation von Zertifikaten notwendig ist.

Aufgaben einer PKI sind z. B. die Ausstellung, Verteilung, Prüfung und der Widerruf digitaler Zertifikate.

5.8.1 Merkle Tree-Authentication-Schema

Merkle stellte 1979 ein Verfahren vor, bei dem Zertifikate in einer Datei gespeichert werden können, Änderungen an einem Zertifikat ändern die Datei, d. h. Reduzierung auf ein Integritätsproblem. Kryptographische Hashfunktionen können benutzt werden, um Änderungen in dieser Datei aufzudecken. Hier werden die Checksummen rekursiv definiert.

Seien dazu

- Y_i ($i = 1, \dots, n$) die zu schützenden Zertifikate,
- $f : D \times D \rightarrow D$ eine Funktion, die Paare von Bitstrings auf Bitstrings abbildet, und
- $h : \mathbb{N} \times \mathbb{N} \rightarrow D$ eine Funktion, die natürliche Zahlen auf Bitstrings wie folgt abbildet:

$$h(i, j) = \begin{cases} f\left(h\left(i, \lfloor \frac{i+j}{2} \rfloor\right), h\left(\lfloor \frac{i+j}{2} \rfloor + 1, j\right)\right) & , i < j \\ f(Y_i, Y_j) & , \text{sonst} \end{cases}$$

Der „Hash“ der ganzen Datei, auch *Wurzel* genannt, ist $h(1, n)$, dieser Wert muss sicher gespeichert werden (so dass sichergestellt ist, dass der korrekte Wert zur Überprüfung genommen wird).

Beispiel für Überprüfung von Y_3 : $h(3, 3) = f(Y_3, Y_3)$

$h(3, 4) = f(h(3, 3), h(4, 4))$

$h(1, 4) = f(h(1, 2), h(3, 4))$

In dem Beispiel, das in Abb. 5.4 gezeigt wird, benötigt man zur Verifikation von Y_3 entweder die Zertifikate Y_1, \dots, Y_4 oder den Authentifikationspfad (in Abb. 5.4 mit gestrichelten Kreisen gekennzeichnet).

Zur Verifikation müssen also entweder alle Zertifikate oder zumindest der Authentifikationspfad, d. h. alle die zusätzlich zur Verifikation nötigen Hashes, vorhanden sein. Der Vorteil dieser Verifikation ist, dass nur dieser Wurzelwert bekannt und die Zertifikationsdatei öffentlich sein muss. Bis da sind auch keine digitalen Signaturen erforderlich. Meistens ist dieses Verfahren aber allein aufgrund der Größe und Verteilung der Teilnehmer unpraktikabel.

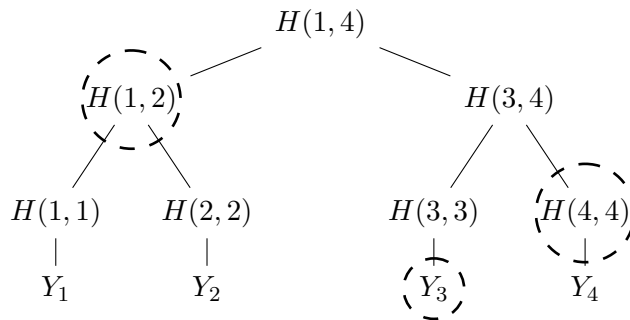


Abbildung 5.4: Beispiel mit 4 Zertifikaten

5.8.2 SPKI – Simple Public Key Infrastructure

SPKI ist eine auf Einfachheit und Erweiterbarkeit hin entwickelte PKI-Form, die hauptsächlich zur Authentifizierung dient. SPKI bietet drei Arten von Zertifikaten:

- ID (Zuordnung $\langle \text{Name}, \text{Schlüssel} \rangle$)
- Attribute ($\langle \text{Autorisation}, \text{Name} \rangle$) und
- Autorisation ($\langle \text{Autorisation}, \text{Schlüssel} \rangle$).

SPKI verwendet „Simple Distributed Security Infrastructure“ (SDSI)-Namen, um global eindeutige Bezeichnungen zu erhalten (dies ist oft ein Problem). Dazu werden lokal eindeutige Namen zusammengebaut:

george: (name fred)

ist der SDSI-Name „fred“ in dem durch „george“ definierten Namensraum. Ebenso:

fred: (name sam),

zusammengesetzt dann

george: (name fred sam).

Global eindeutig werden diese SDSI-Namen, indem (Hashes von) Public Keys als Bezeichnung verwendet werden:

(name (hash sha1 |...|) jim therese)

Weiterhin gibt es Konventionen, um Namen anderer Systeme (z. B. X.509) in dieses Format umzuwandeln.

Zur Autorisation werden normalerweise direkt SPKI-Zertifikate

authorization \rightarrow key

verwendet, die der jeweils Berechtigte ausstellt. Damit und mit einer selbst unterschriebenen Anforderung wird dann die Ressource vom Inhaber des key benutzt. Für Logging usw. kann dann zusätzlich die Kette

authorization \rightarrow key \rightarrow name

benutzt werden.

SPKI bietet darüber hinaus Möglichkeiten, Rechte zu deligieren und sogenannte Threshold-Zertifikate auszustellen, bei denen es eine vorgegebene Anzahl k von Subjekten braucht, um ein bestimmtes Recht zu erhalten.

5.8.3 PKIX/X.509

PKIX/PKIs-basierend auf X.509 beruhen auf dem Prinzip der signierten Zertifikatsketten, die (normalerweise) in einer baumartigen Hierarchie angeordnet sind. Das X.509-Directory Authentication Framework wird in vielen anderen Protokollen (z. B. SSL/TLS) verwendet und ist sehr weit verbreitet.

X.509v3-Zertifikate enthalten hauptsächlich folgende Elemente:

1. Version: 1, 2, 3 (abhängig davon, welche erweiterten Elemente vorhanden sind)
2. Seriennummer: wird von der Zertifikationsstelle vergeben und muss pro Zertifikationsstelle eindeutig sein
3. Signaturalgorithmus plus Parameter
4. Distinguished Name (DN) des Ausstellers, der diesen eindeutig repräsentiert
5. Gültigkeitsdauer: Zeitintervall
6. DN des Inhabers
7. Public Key Information: Algorithmus, Parameter, Public Key
8. Issuer's Unique Identification
9. Subject's Unique Identification
10. Erweiterungen
11. Signatur von 1-10

Einsatz Zertifikate werden von einer Certification Authority (CA) ausgestellt, die u. U. eine Registration Authority (RA) benutzt, um die angegebenen Daten des Benutzers zu verifizieren (oder sie selbst erledigt dies).

Zur Überprüfung von Zertifikaten hat der Überprüfende normalerweise eine Liste von ein oder mehreren CAs (und deren Public Keys), denen er vertraut. Dann wird versucht, einen Pfad aus Zertifikaten zum Herausgeber des zu prüfenden Zertifikats zu finden und damit das Zertifikat zu überprüfen. Meistens geschieht die Überprüfung in einer baumartigen Hierarchie.

Darüber hinaus gibt es die Möglichkeit der *cross certification*, dabei zertifizieren sich CAs gegenseitig.

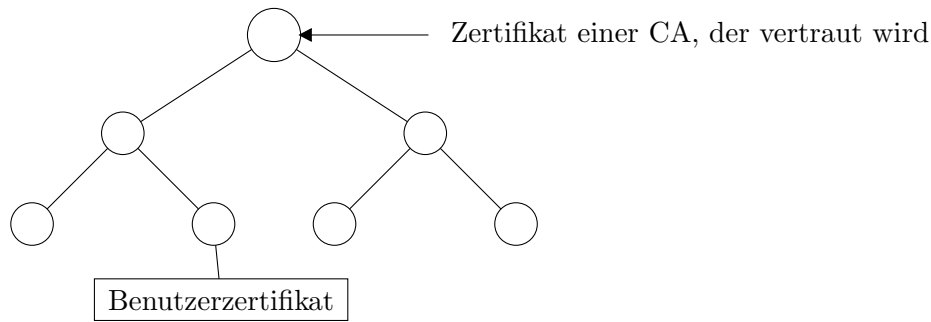


Abbildung 5.5: Zertifikatshierarchie

X.509-Zertifikatswiderruf – CRL Um (z. B. durch Kompromittierung) ungültig gewordenen Zertifikate zu widerrufen, werden zumeist sogenannte *Certificate Revocation Lists* (CRLs) verwendet. Dies sind Listen einer CA, die widerrufenene Zertifikate enthalten und als ganzes signiert sind. Diese Listen enthalten neben dem Ausstellungsdatum das Datum der nächsten geplanten CRL. Zertifikate dürfen erst wieder aus den CRLs entfernt werden, wenn ihre Gültigkeit abgelaufen ist. Um Platz zu sparen gibt es sogenannte Delta CRLs, die nur die Änderungen zur letzten CRL enthalten. Eine andere Möglichkeit, um Zertifikate zu überprüfen, ist das *Online Certificate Status Protocol* (OCSP), bei dem online angefragt wird, ob ein Zertifikat gültig ist. Man erhält dann eine signierte Antwort.

5.8.4 OpenPGP

Ein weiteres häufig eingesetztes PKI-Format ist die Idee des „Web of Trust“, wie es z. B. im OpenPGP-Standard eingesetzt wird. Durch die Software PGP/GnuPG erlangte es eine relativ große Bekanntheit. Bei diesem Format können zu einem Public Key direkt mehrere Signaturen anderer Beglaubigter gehören. Ein OpenPGP-Zertifikat besteht aus einem *Public-Key-Paket* und anschließenden *Signature-Paketen*.

OpenPGP-Public-Key-Paket:

1. Version: 3 oder 4
2. Erstellungszeitpunkt
3. Gültigkeitsdauer
4. Public-Key-Algorithmus und Parameter
5. Public Key

OpenPGP-Signature-Paket:

1. Version: 3
2. Signatortyp

3. Zeitpunkt der Signaturerstellung
4. Key Identification des Unterzeichners
5. Public-Key-Algorithmus
6. Hashalgorithmus
7. Teil des signierten Hashwertes
8. Signatur

Version 4 ähnlich, aber komplexer.

OpenPGP-Zertifikate unterscheiden sich in der Anwendung dadurch, dass ein expliziter Vertrauensstatus angegeben ist. Bei GnuPG hat ein Benutzer die Möglichkeit, das Vertrauen, das er in einen bestimmten Schlüssel setzt, explizit und unabhängig von einer Signatur zu setzen. Standardmäßig wird einem Schlüssel vertraut, wenn man ihn entweder selbst unterschrieben hat, er von jemandem unterschrieben wurde, dem man voll vertraut, oder wenn er von drei teilweise vertrauenswürdigen Schlüsseln unterschrieben wurde und die Zertifikatskette nicht länger als fünf ist. Dies erhöht die Fehlertoleranz, da oft von unerfahrenen Benutzern signiert wird.

Widerruf von Zertifikaten wird durch von Benutzern selbst angelegte Widerrufszertifikate erreicht. Ein Problem ist die Verteilung (u. U. Keyserver).

5.9 Key Escrow

Ein *Key-Escrow-System* (Schlüsselhinterlegung) ist ein System, in welchem eine dritte Partei einen kryptographischen Schlüssel erlangen kann. Einsatzzwecke können in Firmen oder in der Strafverfolgung liegen.

Gewünschte Eigenschaften:

1. Unabhängigkeit vom verwendeten kryptographischen System
2. Ende-zu-Ende-Vertraulichkeit garantiert, außer es werden die hinterlegten Schlüssel verwendet
3. Einbindung in den Schlüsselaustausch
4. Authentifikation aller Parteien, insbesondere der Benutzer der hinterlegten Schlüssel
5. Beachtung von Zeiteinschränkungen

Ein Key-Escrow-System besteht aus drei Komponenten, ein Benutzersicherheitsteil, einer Key-Escrow-Komponente und einer Datenhinterlegungskomponente.

Beispiel: Clipper-Chip

Chaffing and Winnowing (Spreu und Weizen)

Chaffing and Winnowing ist ein System, das entworfen wurde, um aufzuzeigen, dass Key Escrow allein für Verschlüsselungsschlüssel nicht genügt, sondern auch für Authentifikationsschlüssel erforderlich ist. Dies ermöglicht allerdings Impersonisationsangriffe.

Idee:

Nachricht	„Spreu“	HMAC
H		1234
	<i>Z</i>	<i>4798</i>
A		7538
	<i>B</i>	<i>8110</i>
L		3978
	<i>E</i>	<i>7209</i>
L		4900
	<i>A</i>	<i>1141</i>
O		1973

Der Empfänger kann durch seine Kenntnis des Authentifikationsschlüssels entscheiden, ob ein Zeichen mit einer korrekten HMAC versehen wurden. Damit kann er die Zeichen der Nachricht von den mit Zufallswerten „authentifizierten“ Zeichen („Spreu“) trennen.

6 Sichere Wahlen

Durch aufgedeckte Schwachstellen in verwendeten Wahlmaschinen geraten Wahlen mit Wahlmaschinen in Verruf. Hier soll nun nicht weiter auf die Fehler existierender Wahlmaschinen eingegangen werden, sondern es sollen im folgenden einige Ansätze (es gibt noch viele andere Vorschläge) dargestellt werden, wie durch neue Wahlverfahren die Korrektheit und Überprüfbarkeit einer Wahl garantiert werden kann. Durch einen Beleg an den Wähler wäre dies einfach zu erreichen, allerdings ermöglicht ein Beleg, auf dem direkt steht, wie abgestimmt wurde, Erpressung und Stimmenkauf, da der Wähler damit beweisen kann, wie er abgestimmt hat. Im folgenden sollen nun Verfahren vorgestellt werden, die die Korrektheit erreichen, ohne daß auf dem Beleg für jedermann ersichtlich ist, wie abgestimmt wurde. Ist eine Wahl vor Erpressung sicher, ist die Wahl auch geheim, Erpressungsschutz betrachtet sogar den stärkeren Fall, daß der Wähler sich bewußt „falsch“ verhält, um sein Wahlgeheimnis u. U. zu verletzen, auch dies soll nicht (nachvollziehbar) möglich sein.

6.1 Papierbasierte Verfahren

6.1.1 ThreeBallot

ThreeBallot [Riv06] ist ein papierbasiertes Wahlverfahren (mit Hilfsmaschine ...), das von Ronald Rivest vorgestellt wurde. Ziel ist die Überprüfbarkeit der eigenen Stimmabgabe durch einen „Multi-Ballot“, ohne erpeßbar zu werden (was nur mit Einschränkungen erreicht wird). Die Grundidee des Verfahrens besteht darin, daß jeder Wähler *drei* Stimmzettel erhält (bzw. einen dreifachen Stimmzettel, der getrennt wird). Diese soll er nun so ausfüllen, daß *jeder* Kandidat eine Stimme bekommt, der Kandidat, den man eigentlich wählen will, aber zwei, vgl. Abb. 6.1.

Die Stimmen für die Kandidaten sind dabei zufällig auf die drei Stimmzettel zu verteilen, jeder Kandidat muß (in der Summe der drei Stimmzettel) mindestens eine Stimme haben, ein Kandidat darf zwei Stimmen haben (je nach Wahlverfahren auch mehrere, falls man mehrere Kandidaten wählen darf, keiner im Falle der Enthaltung). Jeder Stimmzettel trägt außerdem eine zufällige Seriennummer.

Sind die drei Stimmzettel ausgefüllt, werden sie in eine (vertrauenswürdige) Prüfmaschine eingegeben. Diese überprüft, ob die Stimmzettel korrekt ausgefüllt sind, und signalisiert dies (etwa durch einen Bestätigungsaufdruck). Der Wähler kann dann einen der drei Stimmzettel auswählen und erhält von diesem eine Kopie, die er überprüfen kann und bei der geheim bleibt, welcher der drei Stimmzettel kopiert wurde. Alle drei (Original-)Stimmzettel werden dann in die Wahlurne geworfen.

Für die Auszählung werden alle abgegebenen Stimmzettel veröffentlicht und die Stimmen addiert, die Stimmen für einen Kandidaten minus der Anzahl der Wähler ergibt dann die eigent-

Stimmzettel		Stimmzettel		Stimmzettel	
A	<input checked="" type="checkbox"/>	A	<input type="checkbox"/>	A	<input type="checkbox"/>
B	<input type="checkbox"/>	B	<input checked="" type="checkbox"/>	B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>	C	<input type="checkbox"/>	C	<input type="checkbox"/>
D	<input type="checkbox"/>	D	<input type="checkbox"/>	D	<input checked="" type="checkbox"/>
E	<input type="checkbox"/>	E	<input checked="" type="checkbox"/>	E	<input type="checkbox"/>
2357235		5792854		7486860	

Abbildung 6.1: ThreeBallot-Stimmzettel für B

liche Stimmenanzahl, da jeder Kandidat durch jeden Wähler eine zusätzliche Stimme erhalten hat.

Der Wähler kann (probabilistisch) die Wahl überprüfen, indem er überprüft, ob der kopierte Stimmzettel in der Liste der veröffentlichten Stimmzettel enthalten ist, Fälschungen können damit mit hoher Wahrscheinlichkeit entdeckt werden ($1/3$ für jede gefälschte Stimme). Der Schutz vor Erpressung und Stimmenkauf soll dadurch geschehen, daß man nur einem der drei Stimmzettel alleine die Wahl nicht ansehen kann.

Angriffe gegen dieses Verfahren gibt es einige, zum Teil werden sie schon im ursprünglichen Papier dargestellt. V. a. wird Vertrauen in den Abgabeprozess (und die Prüfmaschine/Kopieroperation) benötigt, außerdem sollte die Zufälligkeit der Wahlen sichergestellt werden, um Angriffe durch das Vorschreiben bestimmter Ausfüllmuster zu erschweren, aber auch das kann wiederum zu Stimmenkaufangriffen führen. Zudem ist das Verfahren recht aufwendig und daher wohl auch nur im Zusammenhang mit Wahlmaschinen praktisch einsetzbar.

6.1.2 Punchscan

Das von David Chaum entwickelte Verfahren Punchscan [Cha06] ist ein weiteres papierbasiertes Wahlverfahren, das auch schon mehrmals praktisch erprobt wurde. Bei Punchscan erhält man einen zweiteiligen Stimmzettel, beide Teile werden übereinandergelegt, nur zusammen enthalten sie die vollständige Stimminformation, beide Zettel enthalten eine (gleiche) Seriennummer. Der obere Zettel listet die Kandidaten zusammen mit einem Symbol (z. B. Buchstaben) in zufälliger Zuordnung, außerdem enthält er für jeden Kandidaten ein Loch, durch das man den unteren Stimmzettel sehen kann. Dieser enthält unter den Löchern des oberen Zettels sichtbar die Symbole der Kandidaten, in (unabhängiger) zufälliger Reihenfolge, vgl. Abb. 6.2.

Um für einen Kandidaten zu stimmen, sucht man sich auf dem oberen Zettel das Symbol für diesen Kandidaten und kreuzt dann das Loch mit diesem Symbol an, so daß sowohl der obere

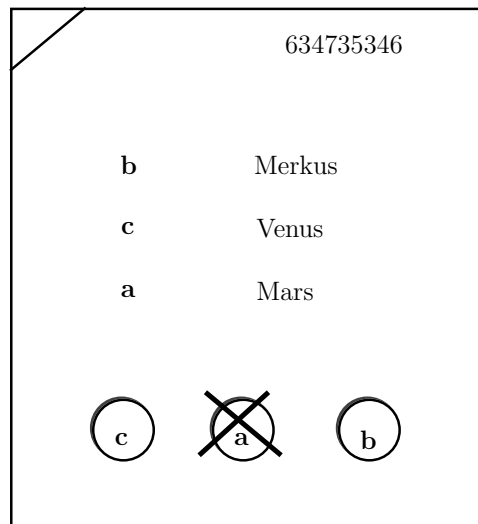


Abbildung 6.2: Punchscan-Stimmzettel für Mars

als auch der untere Stimmzettel an dieser Stelle markiert ist. Einer der Zettel (zufällig gewählt) wird nun vernichtet, der andere dann eingescannt und der Wähler kann eine Kopie mit nach Hause nehmen.

Für die Auszählung werden alle abgegebenen Stimmzettelhälften veröffentlicht, der Wähler kann überprüfen, ob seine Kopie dabei ist. Die Wahlorganisation kann außerdem anhand der Seriennummer die Zuordnung der angekreuzten Leerstelle bzw. des angekreuzten Symbols wiederherstellen und damit das Ergebnis ermitteln. Durch einen kryptographischen Beweis wird außerdem sichergestellt, daß dieser Schritt korrekt und ohne Fälschungen erfolgt (Details dazu in den auf [Cha06] verlinkten Papieren). Der Schutz vor Erpressungen wird dadurch erreicht, daß man dem einzelnen Belegzettel nicht ansieht, für welchen Kandidaten abgestimmt wird, da die entsprechende genaue Information auf dem anderen (vernichteten) Zettel enthalten ist.

Auch dieses Verfahren ist für einen Wähler nicht ganz problemlos zu benutzen, außerdem muß sehr stark auf ein genaues Einhalten gewisser Schritte bei der Wahl geachtet werden (z. B. wann entschieden wird, ob man den unteren oder oberen Zettel vernichtet), ansonsten sind Angriffe möglich.

6.2 Verfahren mit Wahlmaschinen

6.2.1 Verfahren nach Moran und Naor

Auf der Crypto 2006 stelle Moran und Naor ein erpressungssicheres Wahlverfahren mittels einer Wahlmaschine vor [MN06]. Es beruht auf der Annahme, daß der Wähler in der Wahlkabine allein ist und durch den zeitlichen Ablauf des Abstimmvorgangs innerhalb der Wahlkabine die Korrektheit sichergestellt wird, wobei der Wähler Zufallswahlen treffen muß, um die Maschine zu kontrollieren. Ein ähnliches Vorgehen wird auch von anderen Wahlverfahren verwendet.

Die Wahl läuft dabei wie im folgenden beschrieben ab.

1. Die Wahlmaschine präsentiert die Kandidaten und der Wähler wählt seinen gewünschten Kandidaten aus.
2. Die Wahlmaschine präsentiert Eingabefelder für die *nicht* gewählten Kandidaten, in die der Wähler Zufallsdaten einträgt.
3. Es wird der Beginn eines Beleges ausgedruckt, der Wähler kann verifizieren, daß dies geschehen ist, den Inhalt aber noch nicht sehen.
4. Nun wird ein Eingabefeld für den gewünschten Kandidaten angezeigt, in das der Wähler Zufallsdaten einträgt.
5. Die Wahlmaschine druckt den vollständigen Beleg, der Wähler überprüft, daß dieser die Kandidaten (in festgelegter (etwa alphabetischer) Reihenfolge) mit den jeweils eingegebenen Zufallsdaten korrekt auflistet, außerdem enthält er noch weitere Zeilen (nur maschinenverständlich), die in Schritt 3 gedruckt wurden. Falls alles in Ordnung ist, bestätigt der Wähler dies.
6. Die Wahlmaschine zertifiziert den Beleg und händigt ihn dem Wähler aus.

Für die Ergebnisermittlung veröffentlicht die Wahlmaschine nun alle Belege (der Wähler kann überprüfen, daß seiner dabei ist) und einen Beweis für das gültige Addieren der Stimmen. Die Korrektheit wird durch eine Beweistechnik sichergestellt, die es trotzdem nicht ermöglicht, einem Beleg anzusehen, für welchen Kandidaten gestimmt wurde. Die in Schritt 3 auf den Beleg gedruckten Daten enthalten für *jeden* Kandidaten einen Zero-Knowledge-Beweis, daß dies eine gültige Stimme für den jeweiligen Kandidaten ist. Als Zufall für den Zero-Knowledge-Beweis werden dabei die für den entsprechenden Kandidaten eingegebenen und auf dem Beleg ausgedruckten Zufallsdaten benutzt. Da die Wahlmaschine den Zufall aber für die nicht gewählten Kandidaten schon bei der Erstellung des Beweises kannte, kann sie den Beweis simulieren, für den wirklich ausgewählten Kandidaten ist dies nicht möglich, da die Wahlmaschine zum Zeitpunkt der Beweiserstellung den Zufall noch nicht kennt, daher muß sie in diesem Fall einen echten Beweis antreten. Somit ist der Wähler selbst von der korrekten Zählung seiner Stimme überzeugt. Nun ist garantiert, daß die Stimmzettel korrekt ausgewertet werden, durch die Zero-Knowledge-Eigenschaft der Beweise ist dem Beleg aber nicht anzusehen, welcher Beweis echt und welcher simuliert ist. Ein Auszählverfahren, das zufällig Kontrollwerte aufdeckt, garantiert dann noch die richtige Ergebnisermittlung.

Dieses Verfahren benötigt kein Vertrauen in die komplette Wahlmaschine für die Korrektheit der Wahl bis auf gewissen Annahmen an den Drucker. Auch der Schutz vor Erpreßbarkeit ist gegeben, wenn die Isolation des Wählers in der Wahlkabine garantiert werden kann. Nicht berücksichtigt sind allerdings z. B. Angriffe, die dem Wähler per Funk seine einzugebenden Daten übermitteln, ein entsprechender „Brabbelangriff“ wird in [BMQR07] vorgestellt. Dabei wird der Wähler gezwungen, bestimmte Zufallsdaten einzugeben, da sich Menschen aber schlecht Zufallsdaten merken können, können diese vom Wähler nicht ungeordnet werden, somit muß er so abstimmen, wie es der Angreifer wünscht. Durch die Zufallswahlen durch den Benutzer scheint das Verfahren auch für den praktischen Einsatz nicht optimal.

6.2.2 Bingo Voting

Das am E.I.S.S. entwickelte Verfahren Bingo Voting [BMQR07] setzt eine Wahlmaschine und einen vertrauenswürdigen Zufallszahlengenerator ein, um korrekte Wahlen zu erreichen, bei denen der Wähler nicht erpreßbar ist. Eine Wahl besteht dabei aus drei Phasen.

Vorbereitung In der Wahlvorbereitung werden für eine Wahl mit l Wahlberechtigten für jeden Kandidaten P_i l Zufallszahlen r_1^i, \dots, r_l^i erzeugt, daraus werden jeweils Commitments¹ auf (P_i, r_j^i) erzeugt („Dummy-Stimmen“), die veröffentlicht werden. Es wird bewiesen, daß für jeden Kandidaten genau gleich viele Commitments erzeugt wurde (z. B. durch Erzeugen von mehr Commitments und zufälliges Aufdecken ausgewählter).

Wahl Die Wahl läuft wie in Abb. 6.3 beschrieben ab. Der Wähler wählt seinen Kandidaten aus, danach erzeugt der vertrauenswürdige Zufallszahlengenerator eine Zufallszahl, zeigt diese an und übermittelt sie an die Wahlmaschine. Diese druckt einen Beleg aus, der für jeden Kandidaten (geordnet) den Kandidatennamen und eine Zufallszahl enthält: (P_i, s_i) . Dabei muß beim gewählten Kandidaten die Zufallszahl des vertrauenswürdigen Zufallszahlengenerators in der Wahlmaschine stehen, der Wähler sollte dies überprüfen. Die anderen Zufallszahlen müssen Zahlen sein, auf die in der Vorbereitungsphase für den entsprechenden Kandidaten ein Commitment erzeugt wurde (d. h. $s_i = r_j^i$ für ein j).

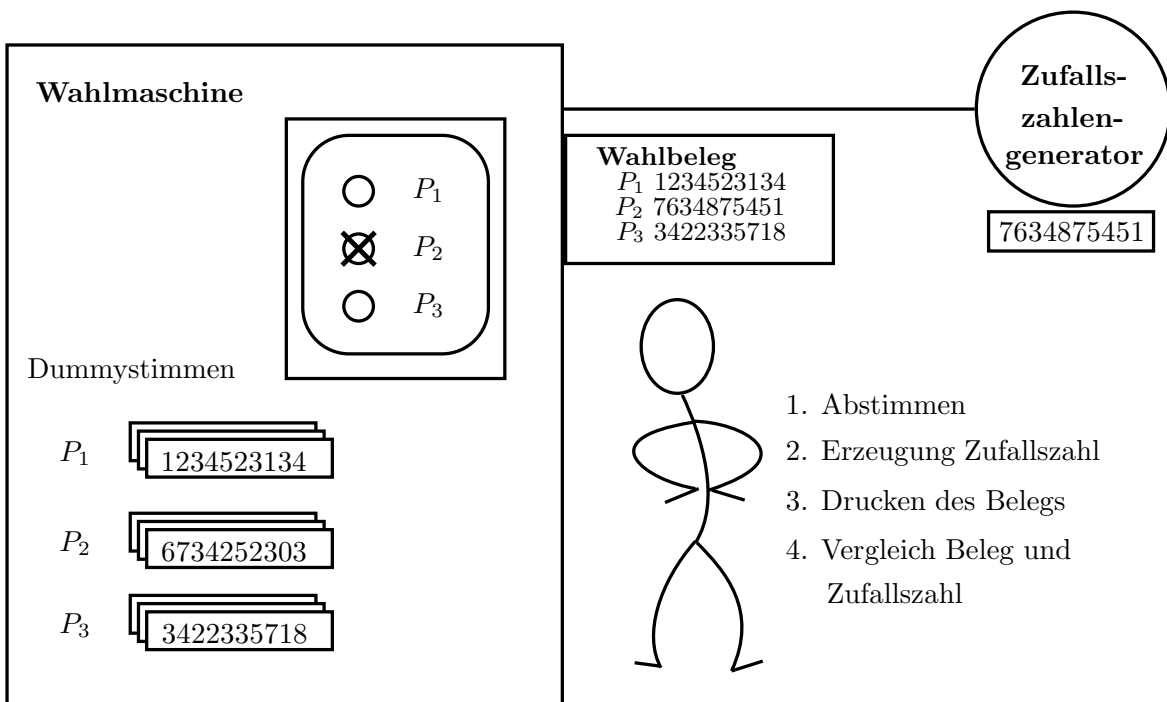


Abbildung 6.3: Wahlablauf bei Bingo Voting

¹Ein Commitment ist eine kryptographische Operation, durch die man sich auf einen Wert festlegt, ohne diesen zu veröffentlichen, ein Beispielverfahren findet sich in Abschnitt 6.2.2.

Nachbearbeitung Die Auszählung der Wahl besteht aus einem Veröffentlichen des Wahlergebnisses zusammen mit einem Beweis der Korrektheit. Dazu werden folgende Daten veröffentlicht:

- die Stimmen für die einzelnen Kandidaten
- alle erzeugten Belege
- eine Liste der ungebrauchten (P_i, r_j^i) (zusammen mit der Information, die benötigt wird, um zu überprüfen, daß sich auf diese Werte zuvor festgelegt wurde)
- Beweis für die Korrektheit, das bedeutet v. a., daß jedes ungeöffnete Commitment auf genau einem Wahlbeleg verwendet wurde.

Anhand dieser veröffentlichten Daten kann nun die Korrektheit der Wahl überprüft werden, der Wähler kann dabei überprüfen, ob sein Beleg richtig veröffentlicht wurde. Für jede Stimme für einen Kandidaten wurde jeweils eine Dummy-Stimme nicht benötigt, d. h. die Anzahl der ungebrauchten Commitments minus der (für alle Kandidaten gleichen) Anzahl Wahlberechtigten, die nicht gewählt haben, ist die Stimmzahl für den jeweiligen Kandidaten. Nun genügt es für die Korrektheit der Wahl zu beweisen, daß auf jedem Wahlbeleg Anzahl Kandidaten minus der einen echten Stimme Dummy-Stimmen erscheinen.

Dieser Beweis kann entweder durch einen großen Zero-Knowledge-Beweis geführt werden, oder er kann, wenn die Commitments eine zusätzliche Homomorphitätseigenschaft besitzen, über sogenanntes „Randomized Partial Checking“ [JJR02] erfolgen. Die zusätzliche Eigenschaft (die z. B. Pedersen-Commitments haben, s. 6.2.2) bedeutet, daß man Commitments in neue Commitments maskieren kann, die ein Commitment auf den alten Wert (unter Verwendung neues Zufalls) sind, dies läßt sich auch beweisen. Damit kann ein Beweis für die Korrektheit eines Stimmzettels nun wie folgt aussehen:

- Es wird ein neues Commitment (P_i, s_i) für die echte Stimme des Wählers mit der vom vertrauenswürdigen Zufallszahlengenerator erzeugten Zufallszahl s_i und dem gewählten Kandidaten P_i erzeugt.
- Die Commitments auf die für diesen Beleg verwendeten Dummy-Stimmen werden ausgewählt und zusammen mit dem neuen Commit veröffentlicht. Jeder kann überprüfen, daß dies ein neues und ansonsten alte Commitments aus der Vorbereitsphase sind.
- Die Commitments werden zu neuen Commitments auf dieselben Werte maskiert und zufällig permutiert, die neue Liste der Commitments wird veröffentlicht.
- Der letzte Schritt wird mit der neuen Commitmentliste wiederholt, so daß wir eine dritte Commitmentliste erhalten.
- Die Commitments der dritten Liste werden aufgedeckt, sie müssen den Inhalt des Belegs (bis auf die Reihenfolge) wiedergeben, d. h. jeder Kandidat kommt einmal vor und ihm zugeordnet ist die auf dem Beleg angegebene Zufallszahl.
- Es wird zufällig (externer Zufall) ausgewählt, ob für alle Commitments der zweiten Liste die Zuordnung zu den Commitments der ersten oder der dritten Liste aufgedeckt wird, und bewiesen, daß dies korrekt geschehen ist.

Dadurch kann die Korrektheit der Wahl garantiert werden durch die Voraussetzung eines vertrauenswürdigen Zufallszahlengenerators. Ein solcher Zufallszahlengenerator in der Wahlkabine könnte etwa vergleichbar einer Lottomaschine oder eines Bingo-Käfigs (daher der Name Bingo Voting) sein, diese Maschinen arbeiten auf recht einsichtige Weise, so daß unentdeckte Manipulationen sehr schwierig wären. Der Erpressungsschutz ist durch eine Ununterscheidbarkeit der Dummy-Stimmen für die anderen Kandidaten und der echten Zufallszahl für den gewählten erreicht.

Das Verfahren stellt keine hohen Anforderungen an den Wähler, er muß nur wählen und sollte zwei Zahlen vergleichen, die restliche Verifikation läuft öffentlich. Allerdings wird einiges an organisatorischem Aufwand für die Wahlvorbereitung und die Sicherung der Dummy-Stimmen vor unbefugtem Zugriff benötigt.

Pedersen-Commitments

Commitment-Verfahren bilden einen wichtigen Grundbaustein für viele kryptographische Protokolle, auch für die Realisierung von Bingo Voting wird ein Commitment-Verfahren benötigt, dafür können z. B. Pedersen-Commitments [Ped91] verwendet werden. Commitment-Verfahren bestehen normalerweise aus zwei Operationen:

1. *Commit/Festlegen*: Es wird aus einer Nachricht m ein Wert berechnet, der veröffentlicht werden kann. Aus diesem Wert kann man nicht auf die Nachricht schließen, man ist aber auf die Nachricht festgelegt.
2. *Unveil/Aufdecken*: Durch Veröffentlichen einer Zusatzinformation (z. B. m selbst und Zufallswahlen) wird m öffentlich und es kann überprüft werden, daß sich im Commit-Schritt wirklich auf m festgelegt wurde.

Intuitiv kann man sich den Commit-Schritt so vorstellen, daß m in einen öffentlichen Tresor gelegt wird, der Schlüssel aber bei der commitenden Partei bleibt. Im Unveil-Schritt wird dann der Schlüssel herausgegeben.

Normalerweise betrachtet man bei Commitments zwei Sicherheitsforderungen: sie müssen *concealing* oder *hiding* (verbergend) sein, d. h. aus der nach dem Commit-Schritt veröffentlichten Information kann nicht auf die Nachricht geschlossen werden, und sie müssen *binding* (bindend) sein, d. h. im Unveil-Schritt ist es nicht möglich, ein Commitment auf eine Nachricht m korrekt für eine Nachricht m' mit $m \neq m'$ zu öffnen. Diese Forderungen sind unter verschiedenen Annahmen erfüllbar.

Pedersen-Commitments beruhen nun auf der Annahme, daß das diskrete Logarithmus-Problem in einer zyklischen Gruppe G von Ordnung q schwer ist. $h, g \in G$ seien Erzeuger von G , so daß der Commitende $\log_g h$ nicht kennt. Um sich nun auf eine Nachricht $m \in \mathbb{Z}_q$ zu committen, wählt man ein zufälliges $r \in \mathbb{Z}_q$, das Commitment ist dann $h^m g^r$. Das Aufdecken geschieht durch Veröffentlichen von m und r , der Empfänger der Nachricht kann dann überprüfen, ob er damit denselben Wert wie das ursprüngliche Commitment errechnet. Das Commitment ist unter der DLOG-Annahme bindend; denn könnte es auf einen anderen Wert m' unter Verwendung von r' aufgedeckt werden, könnte man $\log_g h = \frac{r-r'}{m'-m}$ berechnen, da $h^{m'} g^{r'} = h^m g^r$ und somit $h^{m'-m} = g^{r-r'}$. Die Geheimhaltung ist sogar informationstheoretisch garantiert, da

g^r ein zufälliges Element von G ist, wenn r zufällig gezogen wurde, damit ist auch $h^m g^r$ ein zufälliges Element von G .

Für Bingo Voting wird außerdem eine Maskierungseigenschaft benötigt, dies kann wie im folgenden beschrieben erreicht werden. Sei $c = h^m g^r$ ein Commitment auf m . Wähle ein zufälliges $s \in \mathbb{Z}_q$, $c' = c g^s = h^m g^{r+s}$ ist dann ein maskiertes Commitment c immer noch auf die Nachricht m . Durch Veröffentlichen von s kann gezeigt werden, daß c und c' Commitments auf m sind, ohne m selbst aufzudecken.

7 Sicherheitsbewertung

Wesentliche Dokumente:

- Trusted Computer Security Evaluation Criteria („Orange Book“, TCSEC) [US Department of Defense, 1987]
- Information Technology Security Evaluation (ITSEC) [Commission of the European Community, 1991]
- Common Criteria (for Information Security Evaluation) [Regierungsorganisationen aus Kanada, Frankreich, Deutschland (BSI), Niederlande, UK, USA, ...]

Ziel der Evaluierung

Produktbezug: OS → generische Sicherheitsanforderungen (z. B. Sicherheitsklassen im Orange Book)

Systeme: Menge von Produkten soll für bestimmten Zweck genutzt werden → ITSEC

Orange Book trennt zwischen:

Evaluierung: (Prüfung der Anforderungen) besitzt Sicherheitsprodukt die gewünschten Eigenschaften

Zertifizierung: Ist ein Produkt geeignet für eine bestimmte Anwendung

Akkreditierung: Entscheidung, dass ein Produkt für eine Anwendung genutzt wird

Methode der Evaluierung

Produktorientiert: Prüfen und Testen des Produkts

Prozessorientiert: Schwerpunkt auf Dokumentation und Prozess der Produkterstellung

Struktur der Bewertungskriterien

Funktionalität: „security features“ – DAC, MAC, Authentication, ...

Effektivität: Taugen die Mechanismen für die Anforderungen

Assurance: Gründlichkeit der Evaluierung

Die Evaluationsklassen des Orange Book behandeln alle drei Kriterien simultan, ITSEC nicht. Wieso zertifiziert man? Image, einige Kunden verlangen Zertifizierung, rechtliche Absicherung.

7.1 Orange Book

Vier *security divisions* und sieben *security classes*

D Minimal Protection (eingereicht und nicht bestanden)

C Discretionary Protection

C1 Discretionary Security Protection

C2 Controlled Access Protection

B Mandatory Protection

B1 Labelled Security Protection

B2 Structured Protection

B3 Security Domains

A1 [Verified Protection]

Wesentlich ist der lineare Aufbau → die Anforderungen steigen von Stufe zu Stufe, wobei insbesondere die Forderungen an den Einsatz formaler Beweise und Modelle ansteigen. Zwei Stufen sind in diesem Modell immer mit einander vergleichbar.

7.2 ITSEC

Anderer Ansatz: ITSEC. ITSEC trennt zwischen *objective* (wieso will ich eine Funktionalität?), *function* (was passiert?) und *mechanisms* (wie wird es gemacht).

Um Produkte und Systeme berücksichtigen zu können, bezieht sich ITSEC auf TOEs (Targets Of Evaluation). Dies können sein (u. a.) security objectives, Aussagen über Systemumgebung, Sicherheitsfunktionen, behauptete Minimalstärken, ...

TOE-Funktionalität wird entweder individuell spezifiziert oder durch Referenz auf eine von zehn vordefinierten Klassen (F1-F10). F1-F5 entsprechen dabei C1, B1, B2, B3, A, F6-F10 sind anwendungsspezifisch, z. B. F9 *high confidentiality* für Kryptodevices, F7 *high availability* Klassen sind nicht mehr inkrementell, theoretisch können neue Klassen hinzugefügt werden.

Daneben gibt es sieben Evaluation Level E0-E6 mit aufsteigendem Zutrauen.

Bezug auf Konstruktion, Betrieb des TOE mit Auflage/Spezifikation bezüglich Design, Implementierung, Entwicklungsumgebung, Konfigurationskontrolle, Betrieb, Auslieferung, ...

Typischer Level für kommerzielle Produkte E3 → sicheres Betriebssystem F2+E3.

Orange Book	ITSEC
D	E0
C1	F1+E2
C2	F2+E2
B1	F3+E3
B2	F4+E4
B3	F5+E5
A1	F5+E6

Tabelle 7.1: Vergleich der Einteilungen zwischen Orange Book und ITSEC

Literaturverzeichnis

- [And01] ANDERSON, ROSS: *Security Engineering*. Wiley, 2001.
- [Bis02] BISHOP, MATTHEW A.: *Computer Security. Art and Science*. Addison-Wesley, 2002. ISBN 0-201-44099-7.
- [BMQR07] BOHLI, JENS-MATTHIAS, JÖRN MÜLLER-QUADE und STEFAN RÖHRICH: *Bingo Voting: Secure and coercion-free voting using a trusted random number generator*. <http://eprint.iacr.org/2007/162>, 2007.
- [Bro05] BROWN, DANIEL R. L.: *Prompted User Retrieval of Secret Entropy: The Passmaze Protocol*, 2005. <http://eprint.iacr.org/2005/434>.
- [Bun06] BUNDESAMT FÜR SICHERHEIT IN DER INFORMATIONSTECHNIK (Herausgeber): *IT-Grundschutzhandbuch (IT-Grundschutz-Kataloge, BSI-Standards)*. Bundesanzeiger-Verlag, 2005, 2006. ISBN 3-88784-915-9, ISBN 3-89817-539-547-2, <http://www.bsi.de/gshb/index.htm>.
- [Cha06] CHAUM, DAVID: *Punchscan*, 2006. <http://punchscan.org/>.
- [Cyg] CYGWIN: *Cygwin-Dokumentation*.
- [Gol05] GOLLMANN, DIETER: *Computer Security*. John Wiley & Sons, 2. Auflage, 2005. ISBN 0-470-86293-9.
- [Jab96] JABLON, DAVID P.: *Strong password-only authenticated key exchange*. ACM SIGCOMM Comput. Commun. Rev., 26(5):5–26, 1996. <http://grouper.ieee.org/groups/1363/passwdPK/contributions/jablon.pdf>.
- [JJR02] JAKOBSSON, MARKUS, ARI JUELS und RONALD L. RIVEST: *Making Mix Nets Robust For Electronic Voting By Randomized Partial Checking*. In: *USENIX Security Symposium*, Seiten 339–353, 2002.
- [JW05] JUELS, ARI und STEPHEN A. WEIS: *Authenticating Pervasive Devices with Human Protocols*. In: SHOUP, VICTOR (Herausgeber): *Advances in Cryptology – CRYPTO 2005*, Band 3621 der Reihe *Lecture Notes in Computer Science*, Seiten 293–308. Springer, 2005. http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/11535218_18.
- [mana] *Man-Page acl(5)*.
- [manb] *Man-Pages capabilities(7)*.
- [manc] *Man-Pages chmod(1, 2)*.

- [Mic] MICROSOFT: *Microsoft Knowledge Base*. <http://support.microsoft.com/>.
- [MN06] MORAN, TAL und MONI NAOR: *Receipt-Free Universally-Verifiable Voting With Everlasting Privacy*. In: DWORK, CYNTHIA (Herausgeber): *Advances in Cryptology – CRYPTO 2006*, Band 4117 der Reihe *Lecture Notes in Computer Science*, Seiten 373–392. Springer, August 2006.
- [Ped91] PEDERSEN, TORBEN PRYDS: *Non-interactive and Information-Theoretic Secure Verifiable Secret Sharing*. In: FEIGENBAUM, JOAN (Herausgeber): *Advances in Cryptology – CRYPTO '91: Proceedings*, Band 576 der Reihe *Lecture Notes in Computer Science*, Seiten 129–140. Springer, 1991.
- [Riv06] RIVEST, RONALD L.: *The ThreeBallot Voting System*, Oktober 2006. Draft online available at time of writing <http://theory.lcs.mit.edu/~rivest/Rivest-TheThreeBallotVotingSystem.pdf>.