Zur Verantwortung der Informatiker

Andreas Pfitzmann, Universität Hildesheim, Institut für Informatik, W-3200 Hildesheim

Zusammenfassung
Es wird motiviert und gefordert, präzise zu dokumentieren, wer bei der Konstruktion von informationstechnischen Systemen wofür verantwortlich ist. Dies könnte das Spezifische eines zu schaffenden Berufsrechts für Informatiker sein.

1 Motivation

Wir alle wissen, daß Menschen hin und wieder Fehler machen. Die heutigen informationstechnischen Systeme (IT-Systeme) sind vermutlich einerseits die

- kompliziertesten von Menschen geschaffenen Systeme, andererseits die
- am freiesten von Menschen geschaffenen, vgl. Kapitel 2.

Ersteres sollte uns davor warnen, ihnen blind zu vertrauen, denn je komplizierter etwas ist, desto größer ist die Wahrscheinlichkeit, daß darin Fehler enthalten sind.

Letzteres fordert uns heraus. Denn die vorhandene Freiheit sollte es uns erlauben, IT-Systeme so zu strukturieren, daß sie aus wohlverstandenen, oft geprüften Teilen wohlverstanden zusammengesetzt werden. Dann wäre der bereits heute zu beobachtende Einsatz von Informationstechnik auch in kritischen Bereichen zumindest technisch zu rechtfertigen. Leider wird es dazu nötig sein, auf zumindest manche Zusammensetzungsprinzipien und Konstruktionstechniken zu vertrauen. In ihnen müssen Fehler unbedingt vermieden werden, etwa indem jedes kritische Programm von sehr vielen gelesen und testiert wird.

Deshalb plädiere ich im folgenden, nicht aus „Irren ist menschlich“ auf „Entwurfsfehler machen ist keine Schande“ zu schließen und dies zu „Noch besser ist unklar zu lassen, was überhaupt ein Fehler ist und wo ggf. er steckt und wer ihn verbrochen hat“ zu steigern. Ich befürworte im folgenden eine möglichst präzise Dokumentation, wer bei der Konstruktion eines IT-Systems wofür verantwortlich ist.

Zumindest manche mit der Konstruktion von IT-Systemen Beschäftigte werden eine solche präzise Dokumentation nicht *wollen*, da viele Menschen lieber nicht kontrollierbar arbeiten und ungern Fehler zugeben. Außerdem gibt es chaotisch strukturierte IT-Systeme, wo man diese präzise Dokumentation beim besten Willen im nachhinein nicht erreichen *kann*. Sicherlich gibt es auch Aufgabenstellungen, wo präzise Dokumentation, wer bei der Konstruktion eines IT-Systems wofür verantwortlich ist, beim heutigen Stand der Kunst auch bei geeigneter Strukturierung des IT-Systems nicht befriedigend erreichbar ist. All dies stellt aus meiner Sicht hauptsächlich die betroffene Anwendung der Informationstechnik in Frage und nicht meinen Vorschlag.

2 Das Spezifische der Informatik

Im Gegensatz zu den Natur- bzw. Ingenieurwissenschaften hat der Informatiker nicht primär die *vorgegebene* materielle Natur zu erforschen bzw. gegen die Widerspenstigkeit und Knappheit von Materie und Rohmaterialien anzukämpfen. Der Informatiker kann, von materiellen Grenzen wie Entfernung, Reibung, Verschleiß und Alterung durch den informationstechnischen Fortschritt zunehmend befreit, seine informationstechnischen Systeme fast beliebig *gestalten* oder auch *wuchern lassen*. Neben der für die meisten praktischen Anwendungen irrelevanten „naturgegebenen“ Beschränkung auf das überhaupt genügend effizient Berechenbare und der durch menschliche Organisationsformen gegebenen Beschränkung bzgl. Zeit, Personal und Geld erlebt der Informatiker als Begrenzung – zugespitzt: Gegner – hauptsächlich

seine eigenen begrenzten gedanklichen Fähigkeiten. Zusätzlich setzen in der Praxis mangelnde Motivation, Konzentration und Sorgfalt noch engere Grenzen.

Da die meisten Menschen ungern persönliches Versagen erleben oder gar zugeben und da bei IT-Systemen ein Abschieben auf vorgegebene äußere Naturgesetze und begrenzte materielle Ressourcen meist nicht möglich ist, lassen die meisten Menschen – teils unbewußt – die genaue Urheberschaft und folglich auch die Verantwortung für IT-Systeme heute weitgehend unklar. Beispielsweise ist einem Programm weder anzusehen noch bisher technisch zugeordnet, wer was entworfen, programmiert oder geändert hat und welche Annahmen dabei jeweils über die Umgebung des Programms (andere, insbesondere aufgerufene, Programme; die ausführende Maschine; den Benutzer, der wiederum ein Programm oder auch ein Mensch sein kann) gemacht wurden.

Ebenfalls unklar ist heute die Qualifikation der Entwerfer von IT-Systemen: Jeder darf IT-Systeme entwerfen, produzieren, betreiben, verbinden und modifizieren.

3 Skizze des Spezifischen eines Berufsrechts für Informatiker

Jemand sollte nur dadurch **Informatiker** werden, daß er nach Durchlaufen einer gründlichen fachlichen Ausbildung neben allgemeinen Regeln für Wissenschaftler und Ingenieure folgende informatikspezifische Regeln explizit als verbindlich für seine Arbeit anerkennt:

1. Für jedes (Sub-)System, an dem ein Informatiker mitarbeitet, macht er sich und allen Mitarbeitern sowie durch geeignete Dokumentation den Benutzern klar, um welchen *Systemtyp* (präzis definiert; vage definiert; sozio-technisch, d.h. manche Menschen werden als Teil des IT-Systems betrachtet) es sich handelt, vgl. [Pfit_90]. Damit haben die Schöpfer eines (Sub-)Systems und dessen Benutzer die gleichen Erwartungen an dessen technische Beherrschbarkeit und damit seine maximale Vertrauenswürdigkeit.

2. Ein Informatiker entwirft, produziert, betreibt, verbindet oder modifiziert nur (Sub-)Systeme, nachdem er sich in der dem Systemtyp angemessenen Weise *über alle Schnittstellen informiert und diese vollständig verstanden hat*. Ist ihm dies nicht möglich, so lehnt er die Arbeit am (Sub-)System ab, da zumindest ihm dieses System zu komplex ist, als daß er es im technischen Sinne beherrschen könnte. Findet sich kein Informatiker, der dies kann, so muß das (Sub-)System weiter unterteilt werden oder es müssen einfachere Schnittstellen geschaffen werden.

   Die angesprochene Möglichkeit der Arbeitsverweigerung muß durch eine Anpassung des Arbeitsrechts oder eine spezielle Verpflichtung unterstützt werden, durch die Organisationen/Firmen überhaupt erst das Recht zum Entwurf, der Produktion, dem Betrieb, dem Verbinden und Modifizieren von IT-Systemen erhalten, s.u. Andernfalls wäre zu befürchten, daß der Auftraggeber auf – sei es auch qualitativ ungenügender – Erfüllung seines Auftrags besteht und sich ersatzweise zur Übernahme der „persönlichen“ Verantwortung (siehe 4. Regel) bereit erklärt, wodurch in der Praxis wieder der heutige Zustand hergestellt wäre. Leider ist die Abwesenheit von subtilem Druck auf Informatiker, auch für nicht vollständig Verstandenes Erklärungen abzugeben, nur sehr schwer überprüfbar, was das größte Hindernis bei der Einführung des Vorgeschlagenen in der Praxis sein dürfte.

3. Für jedes entworfene, produzierte, betriebene, verbundene oder modifizierte (Sub-)System *S* unterschreibt der Informatiker eine **Erklärung** folgender Art:

   „Wenn *S* die an seinen Schnittstellen zugesicherten Dienste korrekt zur Verfügung stehen, stellt *S* an seinen Schnittstellen die zu erbringenden Dienste korrekt zur Verfügung.“

Dabei kann durchaus vereinbart werden, daß bei sogenannten fehlertoleranten IT-Systemen beim Auftreten von Fehlern, die die Fehlervorgabe nicht überschreiten [Echt_90], der Umfang der zu erbringenden Dienste – wie spezifiziert – sinkt. Die Erklärung bezeichnet den Systemtyp (präzis definiert, vage definiert, sozio-technisch), der die Präzision des Korrektheitsbegriffs festlegt.

Das (sinnvollerweise: digitale) Unterschreiben sowie die Zusammensetzung der einzelnen Zusicherungen zu umfassenderen kann (und aus Aufwandsgründen: muß) eine Entwurfsumgebung übernehmen. Vorausgesetzt wird für die Sicherheit der persönlichen digitalen Signaturen natürlich, daß jeder einen (kleinen) Rechner ausschließlich unter seiner persönlichen Kontrolle hat.

4. Jeder Informatiker übernimmt die volle persönliche *Verantwortung* für seine unterschriebenen Erklärungen, an die im Einzelfall auch strafrechtliche und/oder zivilrechtliche Konsequenzen geknüpft werden können. Bei kritischen Systemen etwa sollte man fordern, daß nur Informatiker beteiligt sind, die noch nie eine falsche Erklärung abgegeben haben. Erscheint dies zu rigoros, so kann man verschiedene Sorgfaltsstufen einführen und nur fordern, daß der Informatiker noch nie eine falsche Erklärung in dieser oder einer höheren Sorgfaltsstufe abgegeben hat. Ähnlich kann man im zivilrechtlichen Bereich verfahren und die persönliche finanzielle Haftung (etwa durch eine hinterlegte Summe oder Berufshaftpflicht-versicherung) begrenzen, statt sie auf das gesamte Vermögen zu erstrecken. (Insbesondere bei der zivilrechtlichen Haftung kann es bei abhängig beschäftigten Informatikern sinnvoll sein, sie weiterhin dem Arbeitgeber aufzuerlegen.)

   Die Verteilung und – sofern erfolgt – die Begrenzung der Verantwortung ist dem Benutzer des (Sub-)Systems mitzuteilen, so daß er auch hieraus auf den Grad der Vertrauenswürdigkeit schließen kann.

5. Konstruiert ein Informatiker aus Subsystemen ein größeres System, so hat er einerseits die Zusicherungen von Eigenschaften der verwendeten Subsysteme auf Plausibilität und Glaubwürdigkeit zu prüfen: Wer sind die Entwerfer, Produzenten? Was ist ihr informatischer Leumund? Wurden die Subsysteme einer Prüfung durch Dritte unterzogen? Wer hat nach welchen Kriterien geprüft?

   Andererseits sollte er im Zweifelsfall defensiv vorgehen und auf das Subsystem verzichten oder zumindest die zugesicherten Leistungen nicht ausreizen.

6. Um Innovation und Experimentieren weiterhin zu ermöglichen, kann es IT-Systeme geben, die nicht nach obigen Grundsätzen entwickelt wurden. Sie müssen deutlich als solche gekennzeichnet sein und ihr Einsatz sollte die Zustimmung aller Betroffenen voraussetzen.

Um Mißverständnissen vorzubeugen: Insbesondere vage definierte Systemtypen (z.B. Anwendungssysteme, das typische Arbeitsfeld von angewandten, im Gegensatz zu Kerninformatikern) sowie die Erweiterung von (Anforderungs-)Spezifikationen ermöglichen den erwünschten und bereits heute zu beobachtenden pragmatischen Fortschritt im Leistungsspektrum von IT-Systemen weiterhin.

Es sei darauf hingewiesen, daß die geforderte Registrierung, welche Informatiker (noch) in welchen Sorgfaltsstufen beteiligt sein dürfen, nicht unbedingt die Realisierung einer zentralen Überwachungsinstanz bedeutet. Dies kann auch dezentral sichergestellt werden, wozu der „credential“-Mechanismus von David Chaum verwendet werden kann [Chau2_90].

Von der Verbindlichkeit obiger 6 Regeln für *alle* Informatiker verspreche ich mir:

a) Die in der Literatur oft geforderte Spezifikation von überschaubaren (Sub-)Systemen wird tatsächlich praktiziert (Regeln 1-3).
b) Spezifikation und Implementierung werden unverfälschbar dokumentiert (Regel 3). Insbesondere digitales Unterschreiben sichert nicht nur den Benutzer eines (Sub-

)Systems vor dem Entwerfer, Produzenten etc., denn der kann den Inhalt der Erklärung nicht bestreiten. Sondern es sichert auch den Entwerfer, Produzenten etc. eines (Sub-)Systems vor dem Benutzer, denn dieser kann nur zugesicherte Dienste fordern und keine für die Diensterbringung notwendigen Voraussetzungen weglassen, indem er sich auf mündliche Absprachen beruft oder die Dokumentation fälscht. Ebenfalls kann der Benutzer beispielsweise nicht Programme verfälschen und behaupten, diese seien ihm geliefert worden und hätten ihm Schaden verursacht.

c) Informatiker werden auf ihren persönlichen „guten Namen“ achten, so daß der Personenbezug der Regeln 3 und 4 zu individuell höherer Sorgfalt anhält, als wenn nur am Schluß die ein Produkt verkaufende Firma eine Erklärung der in Regel 3 beschriebenen Art unterschreiben würde, was für Produkthaftung völlig hinreichend wäre.

   Es sei betont, daß persönlich abgegebene Erklärungen nicht nur erlauben, den „guten Namen“ zu verlieren, sondern auch, ihn zu bewahren: Kein Vorgesetzter und kein Team kann jemandem einfach seine aus seiner Sicht unfertige Arbeit aus der Hand nehmen und zur Benutzung oder gar zum Verkauf freigeben und bei auftretenden Problemen dann doch wieder auf den Urheber verweisen. Außerdem dürfte eine persönliche und unfälschbare Freigabe der eigenen (Sub-)Systeme Arbeitsorganisationsformen mit größerer Autonomie der Arbeitnehmer unterstützen.

d) Durch das Zusammenbinden der Ziele a), b) und c) wird jedes einzelne nachhaltiger erreicht, als wenn es nur einzeln angestrebt würde.

Obigen, auf das Individuum bezogenen Regeln entsprechen solche, denen sich alle Organisationen/Firmen unterwerfen müssen, die IT-Systeme entwerfen, produzieren, betreiben, verbinden oder modifizieren.

Es ist wichtig, sich klarzumachen, daß wenn man Verantwortung Individuen zuordnen *kann*, eine Vergröberung der Zuordnung zu Teams bzw. Organisationen/Firmen *leicht* ist und aus dem Können kein *Müssen* folgt. Das Umgekehrte gilt leider nicht: Das Wissen, daß z.B. ein Team einen Fehler gemacht hat, ermöglicht es im allgemeinen nicht festzustellen, wer im Team ihn gemacht hat.

Mein Vorschlag behindert Arbeitsteilung, insbesondere Software-Engineering, also nicht, sondern unterstützt sie sogar. Am Ende der Entwicklung kann automatisch geprüft werden, ob die zu erbringenden Dienste des Gesamtsystems mittels der abgegebenen Erklärungen aus den zugesicherten Diensten (im Grenzfall: den für zutreffend gehaltenen Naturgesetzen) hergeleitet werden können. Bei erfolgreicher Prüfung unterschreibt die prüfende Instanz eine Erklärung für das Gesamtsystem. Bei fehlgeschlagener Prüfung wird klar, wo Verantwortungsdefizite stecken. (Die in diesem Artikel geforderten Erklärungen können technisch wie im folgenden Artikel [ClPf_91] beschrieben realisiert werden. Zusammen ergibt sich ein umfassendes System zur Übernahme und ggf. späteren Zuweisung von Verantwortung beim Auftreten von „Fehlern“.)

Eine Organisation/Firma erhält das Recht zum Entwurf, der Produktion, dem Betrieb, dem Verbinden oder Modifizieren von IT-Systemen nur dann, wenn

- die Entstehungsgeschichte ihrer (Sub-)Systeme lückenlos durch Erklärungen von Informatikern abgedeckt ist und
- sie keinen Druck auf Informatiker ausübt, für nicht vollständig Verstandenes Erklärungen abzugeben.

Erscheint dies zu rigoros, so können – wie oben bezüglich Individuen beschrieben – verschiedene Sorgfaltsstufen unterschieden werden.

Es bleibt noch festzulegen, welche Institution das Recht zum Entwurf, der Produktion, ... von IT-Systemen verleiht. Die Möglichkeiten reichen von der GI bis zum Wirtschaftsministerium. Da ein erheblicher Teil der IT-Systeme aus dem Ausland importiert wird, stellt sich die Zulassungsfrage auch dort oder spätestens beim Import von Informationstechnik.

4 Was ist von wem zu tun?

Zunächst müssen diese informatikspezifischen Regeln auf Relevanz und vor allem Vollständigkeit geprüft werden. Ggf. sind weitere herauszuarbeiten.

Es muß diskutiert werden, ob und ggf. durch welche natur- oder ingenieurwissenschaftliche Kodizes oder gar allgemeinmenschliche Kodizes sie ergänzt werden sollen, vgl. die angeführte Literatur.

Es ist zu entscheiden, was ethischer Kodex werden und was zu Recht gerinnen soll.

Danach ist noch viel Formulierungssorgfalt nötig.

Bezüglich einer Einführung eines ethischen Kodexes bzw. Berufsrechts für Informatiker hoffe ich einerseits auf die Unzufriedenheit der *Benutzer* heutiger IT-Systeme, die oftmals nicht primär dem Menschen Arbeit abnehmen, sondern ihn durch Verschleierung der Verantwortlichkeiten letztlich von aller Verantwortung befreien. Man denke nur an in Verwaltungen häufig gehörte Entschuldigungen: „Das kann unser Computer leider nicht." oder „Da muß unser Computer einen Fehler gemacht haben."

Andererseits hoffe ich auf das Eigeninteresse der Informatiker als Berufsgruppe, es nicht schwarzen Schafen und, wegen der weitreichenden Konsequenzen von Schlampigkeit, Zeitdruck etc., auch nicht grau gesprenkelten zu überlassen, weiterhin das Berufsbild in der interessierten Öffentlichkeit zu prägen.

5 Was bleibt trotz Dokumentation der Verantwortlichkeiten und Berufsrecht offen?

Keine noch so genaue Dokumentation der Verantwortlichkeiten und auch kein Berufsrecht für Informatiker kann verhindern, daß

- unerkannte „Fehler" dauerhafte Auswirkungen haben, etwa vertrauliche Information von einem Trojanischen Pferd über einen verdeckten Kanal zu einem interessierten Empfänger übertragen wird, und daß
- auch der erste erkannte Fehler nicht wiedergutzumachende Auswirkungen haben kann, etwa das Versagen einer Kernkraftwerkssteuerung.

Hier helfen nur vorbeugende Schutzmaßnahmen, z.B. wo möglich bereits das Vermeiden der Erfassungsmöglichkeit von Daten und die Begrenzung des Schadenspotentials von Anwendungen, ggf. also auch der Verzicht auf solche mit zu hohem Schadenspotential.

Eine bessere Kontrolle der entworfenen IT-Systeme ist der hier als Notmaßnahme vorgeschlagenen Kontrolle ihrer Entwerfer und der bereits weitgehend praktizierten Kontrolle ihrer Benutzer (etwa durch intensive Protokollierung ihres Tuns) also vorzuziehen. Hierfür wird rechnergestützte Verifikation zukünftig hoffentlich einen wesentlichen Beitrag leisten können. Dort, wo sie erfolgreich angewendet werden kann, kann und sollte die Kontrolle der Entwerfer von IT-Systemen entsprechend reduziert werden.

Ein Dankeschön

Für Kritik und Anregungen danke ich Gerrit Bleumer, Manfred Böttger, Dirk Fox, Birgit Pfitzmann, Prof. Dr. Reinhard Voßbein, Michael Waidner und zahlreichen Diskutanten im GRVI / GDD / DVD / DGIR / GI-Diskursprojekt „Rechtliche Beherrschung der Informationstechnik" herzlich.

Zitierte Literatur

Chau2_90 David Chaum: Showing credentials without identification: Transferring signatures between unconditionally unlinkable pseudonyms; Intern. Conf. on Cryptology, Sydney, Australia, January 1990, AUSCRYPT '90, LNCS 453, Springer-Verlag, Berlin 1990, 246-264.

ClPf_91 Wolfgang Clesle, Andreas Pfitzmann: Rechnerkonzept mit digital signierten Schnittstellenprotokollen erlaubt individuelle Verantwortungszuweisung; Datenschutz-Berater 15/8-9 (1991).

Echt_90 Klaus Echtle: Fehlertoleranzverfahren; Studienreihe Informatik, Springer-Verlag, Heidelberg 1990.

Pfit_90 Andreas Pfitzmann: Entwicklungslinien der Informationstechnik und Informatik und ihre Auswirkungen auf rechtliche Beherrschung; Datenschutz und Datensicherung DuD 14/12 (1990) 620-627.

Literatur zu Ethische Kodizes für Wissenschaftler und Ingenieure

Association for Computing Machinery: Bylaw 21. ACM Code of Professional Conduct; Communications of the ACM 23/7 (1980) 425.

Edward C. Bertnolli, Robert Alden, William Middleton, William Tackaberry, Steven Unger: Proposed new Code of Ethics released by ad hoc committee; The Institute, IEEE, 14/3 (1990) 5.

C. Dianne Martin, David H. Martin: Professional Codes of Conduct and Computer Ethics Education; Computers & Society 20/2 (1990) 18-29.

William W. Middleton: Does the IEEE Code of Ethics need fixing? IEEE-USA unit's proposed Code; The Institute, IEEE, 14/7 (1990).

Edward A. Torrero: Board backs new code; IEEE Code of Ethics; The Institute, IEEE, 14/9 (1990) 2.

Panel to eye response to plan for changes in Ethics Code; The Institute, IEEE, 13/4 (1989) 11.

Emerson W. Pugh: Must we give up ethics to eat?; The Institute, IEEE, 13/4 (1989) 6.

Hal Sackman: Preliminary IFIP Code of Ethics; IFIP Newsletter December 1989.

Verein Deutscher Ingenieure (VDI): Bekenntnis des Ingenieurs; Düsseldorf, den 12. Mai 1950.

Literatur zu Verantwortung von Wissenschaftlern und Ingenieuren

Donald Christiansen: Ethical dilemmas revisited; IEEE spectrum 26/4 (1989) 21.

Literatur zu Verantwortung von Informatikern

Joachim Biskup: Verantwortung des Hochschullehrers für Informatik; Informatik-Spektrum 12/3 (1989) 158-161.

Committee on Public Policy: Draft 7B: Entity Position Paper on Software Vandalism; Computer, IEEE, 22/7 (1989) 83-84.

Wolfgang Coy: Brauchen wir eine Theorie der Informatik?; Informatik-Spektrum 12/5 (1989) 256-266.

Michael C. McFarland: Urgency of ethical standards intensifies in computer community; Computer, IEEE, 23/3 (1990) 77-81.

Susan J. Harrington, Rebecca L. McCollum: Lessons from Corporate America applied to Training in Computer Ethics; ACM SIGSAC Review 8/3 (1990) 23-28.

Harold Joseph Highland: Random Bits & Bytes; Computers & Security 7/5 (1988) 439-449.

Hans Lenk: Können Informationssysteme moralisch verantwortlich sein?; Informatik-Spektrum 12/5 (1989) 248-255.

Bernd Lutterbeck: Positionspapier zum Thema: Ethische Grundlagen der Informatik: Die Verantwortung der Informatiker; Arbeitstreffen Theorie der Informatik vom 2. bis 5. Oktober 1990 in Bederkesa, Mai 1990.

Nilakantan Nagarajan: What's Computer Ethics, anyway?; SIGSAC Review, acm, 8/2 (1990) 24-32.

Richard Parker: Computer-related crime: Ethical considerations (with applications for teaching computer literacy classes); ACM SIGSAC Review 8/3 (1990) 13-22.

Ralph J. Preiss: Action needed on computer ethics legislation; Computer, IEEE, 23/2 (1990) 74.

Karl-Heinz Rödiger, Wolfgang Coy, Günther Feuerstein, Rolf Günther, Werner Langenheder, Bernd Mahr, Peter Molzberger, Hartmut Przybylski, Horst Röpke, Eva Senghaas-Knobloch, Birgit Volmerg, Walter Volpert, Hellmut Weber, Herbert Wiedemann: Informatik und Verantwortung; Arbeitskreis 8.3.3 "Grenzen eines verantwortbaren Einsatzes von Informationstechnik" der Gesellschaft für Informatik; Informatik-Spektrum 12/5 (1989) 281-289.

Eric A. Weiss: The XXII Self-Assessment: The Ethics of Computing; Communications of the ACM 33/11 (1990) 110-132.

Jan Witt: Dogma und Skepsis; Gedanken zur Angemessenheit der aktuellen Technik-Kritik im Bereich der Informatik; Informatik-Spektrum 12/5 (1989) 274-280.

Rechnerkonzept mit digital signierten Schnittstellenprotokollen erlaubt individuelle Verantwortungszuweisung

Wolfgang Clesle, ANT Nachrichtentechnik, W-7150 Backnang

Andreas Pfitzmann, Universität Hildesheim, Institut für Informatik, W-3200 Hildesheim

Zusammenfassung

Bei der Realisierung von informationstechnischen (IT) Systemen können Fehler nicht verhindert werden. Leider kann bei den heute üblichen Rechnern ein aufgetretener Fehler im allgemeinen nicht lokalisiert und schon gar nicht geklärt werden, wer ihn begangen hat. Die Beseitigung dieser Mängel würde natürlich zur mittelfristigen Steigerung der Vertrauenswürdigkeit der IT-Systeme erheblich beitragen.

Im folgenden wird deshalb ein Rechnerkonzept vorgeschlagen, das mittels digitaler Unterschriften unfälschbare Protokollierung ermöglicht. Mit dieser können aufgetretene Fehler lokalisiert und die Verantwortung zweifelsfrei den Urhebern zugewiesen werden. Danach wird die technische Realisierung des vorgeschlagenen Rechnerkonzeptes betrachtet. Software-Schutz und Redundanzmaßnahmen werden darauf übertragen. Zuletzt wird untersucht, inwieweit das Rechnerkonzept die Bedrohung durch Computerviren reduziert.

Bisher wurde das Konzept für reale Rechner und Anwendungen nicht konkretisiert. Deshalb liegt keine quantitative Analyse des sicherlich beträchtlichen Aufwands und des bei einem realen System sicherlich einzuschränkenden „zweifelsfrei“ vor. Da jedoch Alternativen zum Beschriebenen schwer vorstellbar sind, sollte dies eher Anlaß zum Mit- und Weiterdenken als zum Beiseitelegen und Abwarten sein.

1 Motivation

Informationstechnische Systeme (IT-Systeme) beeinflussen immer mehr Bereiche des täglichen Lebens. Diese Systeme werden (hoffentlich!) gemäß einer Spezifikation realisiert. Bei der Realisierung der Systeme werden je nach Größe des Gefährdungspotentials Techniken angewendet, um die Sicherheit des Systems (im Sinne von Ausfallsicherheit und Verfügbarkeit) gemäß der Spezifikation zu erfüllen.

Bei der Realisierung solcher Systeme werden Werkzeuge (Compiler, Editoren, CAD, CIM,..) und existierende Subsysteme (Datenbanken, PC) benutzt. Das realisierte System wird im besten Fall gemäß den allgemein anerkannten Richtlinien entworfen und die Qualität durch entsprechende Maßnahmen gesichert. Da die Verifikation für größere Entwürfe noch nicht möglich ist, werden in der Praxis Codeinspektionen und Tests durchgeführt, um die Korrektheit zu überprüfen.

Bei der Realisierung größerer Systeme wird von der Korrektheit der Subsysteme und Werkzeuge ausgegangen oder es werden Redundanztechniken benutzt, um Fehler zu tolerieren. Aber auch im letzteren Fall kann ein solches System derartige oder so viele Fehler enthalten, daß die Spezifikation nicht erfüllt ist.

Neben „Fehlern“ aus Flüchtigkeit oder Überforderung gibt es natürlich auch bewußt böswillige „Angriffe“, die unter dem Schlagwort „Trojanische Pferde“ (abgekürzt: **TP**) zumindest außerhalb der Informatik seit vielen Jahrhunderten bekannt sind [Home_??, Denn_82, Denn_85].

Die Entstehung von Fehlern und Trojanischen Pferden kann in der Praxis und mit dem heutigen Stand der Qualitätssicherung nicht verhindert werden, und somit kann der Hersteller von Werkzeugen Fehler oder Trojanische Pferde in viele Produkte, die mit diesen Werkzeugen hergestellt werden, einbringen [Pfit_89, Pfit_90]. Es sollte aber zumindest möglich sein, den Hersteller von Produkten mit Fehlern oder Trojanischen Pferden zur Verantwortung zu ziehen [Pfit5_91].

Die Leistung, die ein TP erbringt, ist vom gutgläubigen Benutzer im allgemeinen nicht gewünscht. Deshalb entstehen unvorhergesehene Resultate, die vielleicht nur zeitweise auftreten. Bei solchen zeitweise „falschen“ Leistungen ist der Nachweis dieser falschen Leistung gegenüber Dritten (z.B. vor Gericht) sehr schwierig und eine Schuldzuweisung kann deshalb nur in Ausnahmefällen erfolgen.

Falls aus diesen Fehlern ein Schaden resultiert, muß ein Verantwortlicher gefunden werden, der für den entstandenen Schaden haftet. Dazu werden in Informationssystemen die Aktionen protokolliert. Die korrekte Zuweisung einer Verantwortung ist aber nur möglich, falls diese Protokollierungen weder durch Benutzer noch durch Betreiber oder Hersteller modifiziert oder umgangen werden können.

Im folgenden wird ein neues Rechnerkonzept vorgeschlagen, mit dem es möglich ist, die fehlerhafte Komponente zweifelsfrei zu bestimmen, um im Fehlerfalle die Verantwortung zuzuweisen oder diese Komponente zu korrigieren. Dies dürfte wesentlich zur Vertrauenswürdigkeit von IT-Systemen beitragen.

Die Basis der Vertrauenswürdigkeit im Geschäftsleben bilden schriftliche Verträge. Hierbei übernimmt eine Person für eine von ihr erbrachte Leistung im allgemeinen die Verantwortung durch ihre **Unterschrift**, die als unfälschbar betrachtet wird. Wenn sich dann ein Fehler bei der erbrachten Leistung herausstellt, kann der Verantwortliche anhand des Vertrages ermittelt und anhand der Unterschrift zur Verantwortung gezogen werden.

Eine solche Vorgehensweise wird deshalb auch auf Software (SW) und Hardware (HW) umfassende Produkte übertragen, um eine Fälschung der Protokollinformation auszuschließen. Somit ist die Verantwortung für bestimmte Leistungen des Produkts geklärt. Dies ist insbesondere für Produkte wichtig, zu deren Herstellung Werkzeuge benutzt werden. Vereinfacht ausgedrückt bedeutet dies:

Wenn ein System eine Leistung erbringt, so muß der Hersteller des Systems die Verantwortung für diese erbrachte Leistung übernehmen, d.h. für eine falsch erbrachte Leistung muß ein Nachweis gegenüber den dafür Verantwortlichen erbracht werden können, so daß die Schuld richtig zugewiesen werden kann.

Deshalb werden die Unterschriften aus dem Geschäftsleben in den Bereich der IT-Systeme übertragen, um die Grundlage für mögliche Verantwortung zu schaffen.

Weil aber Programme nur mit entsprechender Hardware lauffähig sind, müssen bei einer solchen Übertragung neue Objekte betrachtet werden. Diese neuen Objekte heißen Instanzen und werden im nächsten Kapitel behandelt.

Die Verantwortung für falsch erbrachte Leistungen muß den fehlerhaften Instanzen zugewiesen werden können, aber auch innerhalb einer Instanz muß die Verantwortungszuweisung an den Hersteller des Rechners oder des Programms möglich sein. Die Verantwortung und deren Zuweisung wird in Kapitel 3 untersucht.

2 Beweisbare Kooperation zwischen Instanzen

Um Verantwortung in IT-Systemen herzustellen, muß zuerst geklärt werden, welches aktive Objekt (Subjekt genannt) Leistungen erbringt und welches Subjekt diese Leistungen unterschreiben kann. Danach muß untersucht werden, wie eine solche Unterschrift realisiert werden kann.

2.1 Instanzen als Subjekte

Ein Programm erbringt gewisse Leistungen nur dann, wenn es auf einem Rechner läuft; ansonsten ist das Programm inaktiv (sozusagen passives Objekt).

Als Subjekte können somit nur Programme in Verbindung mit einer gewissen Hardware-Firmware-Umgebung aufgefaßt werden. Diese Betrachtung ergibt die folgende Definition.

Definition der Instanz

Eine **Instanz** ist das Subjekt, das aus einem Programm des Programmherstellers *X* und der notwendigen Hardware-Firmware-Umgebung (Prozessor mit Speicher und E/A-Möglichkeiten und meist auch noch Teile des Betriebssystems, so daß das Programm lauffähig ist und die geforderte Leistung erbringen kann) des Rechnerherstellers *Y* gebildet wird. Das gleiche Programm kann in mehreren Hardware-Firmware-Umgebungen Teil verschiedener Instanzen sein. Ebenso kann dieselbe Hardware-Firmware-Umgebung mit unterschiedlichen Programmen verschiedene Instanzen bilden.

Alle Ausgabedaten einer Instanz werden von dieser unterschrieben und alle Eingabedaten werden, soweit dies möglich ist, auf eine vorhandene Unterschrift geprüft.

Die unterschriebenen Ausgaben, die von einer Instanz erzeugt werden, heißen **Dokumente**.

Die Verantwortungszuweisung zwischen dem Programmhersteller und dem Rechnerhersteller innerhalb einer Instanz wird in Kapitel 3 betrachtet.

Digitale Signatursysteme zur Sicherung gegen späteres Abstreiten

Die Grundforderungen an eine digitale Signatur [DiHe_76] sind:

1. Niemand außer der Instanz selbst sollte fähig sein, eine Nachricht mit der zu dieser Instanz gehörenden Signatur zu versehen.
2. Jeder kann nachprüfen, ob eine Nachricht mit einer zu einer bestimmten Instanz gehörenden Signatur versehen ist.

Der einfache Ansatz, auf die Einzigartigkeit der Instanznamen zu vertrauen und diese wie herkömmliche Unterschriften der Erklärung beizufügen, genügt obigen Anforderungen leider nicht, denn jeder, der einmal eine unterschriebene Erklärung von einem Benutzer erhalten hätte, könnte den beigefügten digitalen Instanznamen auf beliebig viele weitere Nachrichten kopieren.

Obiges Problem löst ein **digitales Signatursystem** (Bild 1). Jede Instanz wählt eine Zufallszahl und generiert sich von ihr ausgehend ein spezielles Funktionenpaar (s,t); die Signierfunktion s dient zum Unterschreiben von Nachrichten und ist nur dem Träger des Instanznamens bekannt, während das Testprädikat t dazu dient, zu testen, ob Nachrichten mit s unterschrieben wurden. Das Testprädikat t wird jedem bekannt gegeben.

Bei praktischem Einsatz besteht diese Beschreibung des Testprädikats nur noch in einem **Schlüssel**, der in eine ansonsten allgemein bekannte Testfunktion eingesetzt wird. Sie sollte standardisiert sein, damit jeder jede Unterschrift mittels einer effizienten Implementierung testen kann.

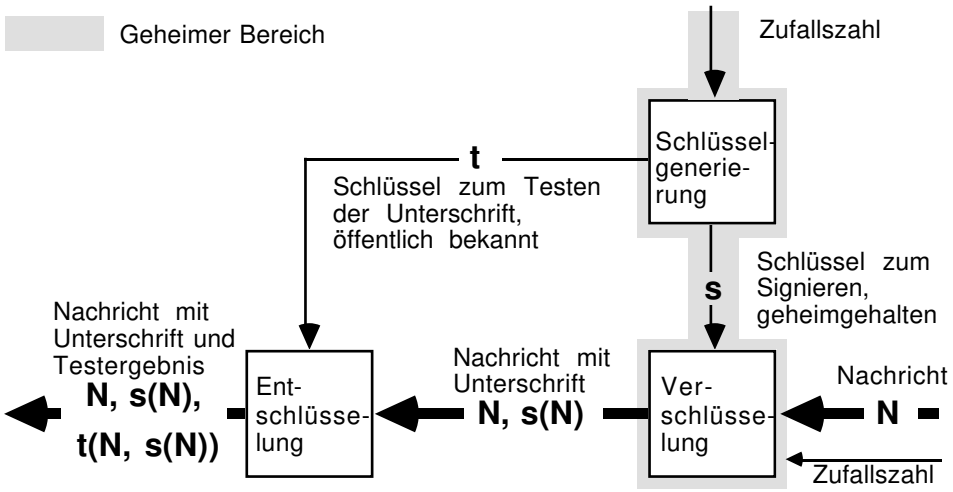

Bild 1: Signatursystem
(= Glasvitrine mit Schloß, es gibt nur einen Schlüssel, um etwas hineinzutun)

Ein einfaches und bekanntes Signatursystem erhält man bei Verwendung des asymmetrischen Kryptosystems RSA [RSA_78, Denn_82].

Eine empfehlenswerte Alternative zu RSA stellt das GMR-Signatursystem dar [GoMR_88].

Dieses System ist ähnlich effizient wie RSA [FoPf_91], bietet jedoch den Vorteil, daß es selbst gegen die denkbar stärksten (d.h. adaptiven aktiven) Angriffe als kryptographisch sicher bewiesen wurde: sein Brechen ist äquivalent zur Faktorisierung einer gegebenen Zahl bestehend aus zwei großen Primzahlen eines bestimmten Typs.

Wo möglich sollte das GMR- dem RSA-Signatursystem vorgezogen werden.

Jedes Signatursystem (wie auch jedes asymmetrische Konzelationssystem und jede kollisionsfreie Hash-Funktion, vgl. spätere Kästen) kann mit hinreichend großem Aufwand gebrochen werden, auch wenn die Systeme so dimensioniert werden, daß nach heutigem Kenntnisstand die schnellsten im nächsten Jahrhundert zu erwartenden Rechner Jahrmilliarden rechnen müßten. Leider ist nämlich nicht völlig ausgeschlossen, daß wesentlich effizientere Algorithmen zum Fälschen von Signaturen gefunden werden. In üblichen Signatursystemen trägt das Risiko hierfür der Unterzeichner, da er nicht beweisen kann, daß die „Fälschung“ nicht von ihm stammt. Durch ein **Fail-Stop-Signatursystem** [PfWa_91, BIPW_91] kann dies vermieden werden: Wird eine Unterschrift gefälscht, so kann der angebliche Unterzeichner diese Fälschung jedem Dritten nachweisen.

Eine Instanz unterschreibt alle ihre Ausgabedaten mit einer digitalen Signatur (siehe Kasten), dem Analogon zur (eigenhändigen) Unterschrift im Geschäftsleben. Alle Dokumente können auf eine vorhandene Unterschrift geprüft werden, und somit kann die Authentizität der Dokumente sichergestellt werden. Falls es sich bei Daten um Fälschungen handelt, d.h. die Unterschrift nicht korrekt ist, werden diese nicht als Dokumente bezeichnet. Ein Dokument ist somit eine von einer beliebigen Instanz korrekt unterschriebene Ausgabe. Dadurch ist auch sichergestellt, daß für jedes Dokument ein Urheber existiert.

In Bild 2 wird die logische Struktur einer Instanz dargestellt.

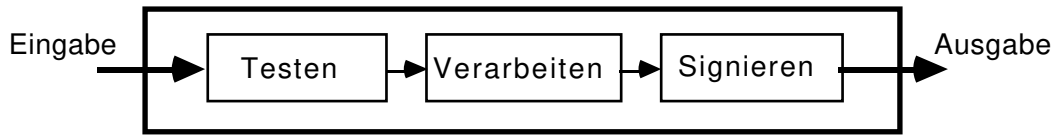

Bild 2: Logische Struktur einer Instanz

Um Eingaben in ein aus Instanzen bestehendes System machen zu können, gibt es **Eingabe-Instanzen**, die unsignierte Eingaben verarbeiten können. Diese erzeugen als Ausgaben ebenfalls Dokumente.

Im folgenden wird folgende Notation benutzt:

Die Instanz **I** signiert ihre Ausgaben mit $\mathbf{s_I}$.

Die Unterschrift einer Instanz I kann mit dem Testprädikat $\mathbf{t_I}$ überprüft werden.

Eine Instanz I kann zur Erbringung der geforderten Leistung eine andere Instanz S benutzen. Dies entspricht einer Zerlegung der Instanz in Subinstanzen I_1 und I_2, wobei die Instanzen I_1, S und I_2 hintereinandergeschaltet werden. D.h. die Instanz I_1 generiert als Zwischenresultat die notwendige Ausgabe, die als Eingabe der Instanz S dient, die Ausgabe von S dient als Eingabe der Instanz I_2, die Instanz I_2 bearbeitet den Auftrag der Instanz I zu Ende und liefert das entsprechende Dokument als Ausgabe.

Auch wenn sich mehrere Instanzen gegenseitig zur Erbringung der geforderten Leistung benutzen und ggf. parallel arbeiten, kann man den Ablauf sequentialisiert betrachten und so als Hintereinanderschaltung von Instanzen der oben geschilderten Art auffassen.

Mehrere Instanzen können aus Effizienzgründen zu einer neuen Instanz zusammengefaßt werden. Diese neue Instanz übernimmt die Verantwortung der eingebetteten Instanzen.

Falls nur die Korrektheit der E/A-Relation sichergestellt werden soll, kann bei einem vorhandenen Absoluttest (Prädikat, das wahr liefert, falls das Resultat r zur Eingabe e gemäß der Spezifikation richtig ist) für die Spezifikation der eingebetteten Instanzen die Verantwortung direkt übernommen werden. Falls kein solcher Absoluttest vorhanden ist und die Verantwortung innerhalb der Gesamtinstanz zuweisbar sein muß, müssen Protokollierungen (Kapitel 2.2.2) vorgenommen werden.

Die Struktur der Dokumente wird im nächsten Unterkapitel betrachtet.

2.2 Dokumente als Objekte

Die Dokumente müssen die Anforderungen für mögliche Verantwortungszuweisungen erfüllen. Diese Anforderungen ergeben sich aus den Rahmenbedingungen, etwa dem nötigen Umfang einer Verantwortungszuweisung.

Folgende Anforderungen an Dokumente sind möglich:

- Die vollständige Rekonstruktion der Entstehungsgeschichte muß mit den Dokumenten möglich sein.
- Nur die E/A-Relation einer Instanz muß überprüfbar sein.

Wenn die erbrachte Leistung einer Instanz überprüft werden muß, dann muß neben der Ausgabe auch die Eingabe unterschrieben sein, so daß die Spezifikation gemäß dieser Eingabe überprüft werden kann.

Beispiel:

Die Funktion **f** soll die Summe von zwei ganzen Zahlen berechnen.

Um die Einhaltung der Spezifikation zu überprüfen, werden auch die Eingabedaten benötigt.

Falls die Instanz, die die Funktion f berechnet, ein falsches Resultat geliefert hat, kann diese Instanz zur Verantwortung gezogen werden, weil die E/A-Relation von ihr unterschrieben wurde.

Für manche Anwendungen ist es sinnvoll, die gesamte Entstehungsgeschichte der Dokumente rekonstruieren zu können. Wenn beispielsweise in der Programmentwicklung nur bestimmte Werkzeuge benutzt werden dürfen, so kann die Verwendung dieser Werkzeuge durch Inspektion der Entstehungsgeschichte des Programms überprüft werden.

Es soll im folgenden die Struktur der Dokumente betrachtet werden, wenn die gesamte Entstehungsgeschichte rekonstruierbar sein soll; zur Überprüfung der E/A-Relation braucht diese Struktur nur leicht modifiziert zu werden.

2.2.1 Struktur eines Dokuments

Damit die Richtigkeit einer Unterschrift geprüft werden kann, muß der Name der erzeugenden Instanz im Dokument enthalten sein oder der Prüfer des Dokuments muß die erzeugende Instanz kennen.

Um die Entstehungsgeschichte des Dokuments eindeutig rekonstruieren zu können, müssen *alle* Eingabedokumente und die daraus resultierenden Ausgabedaten im Ausgabedokument einer Instanz enthalten sein.

Schließlich muß das Ausgabedokument eine Signatur enthalten. Somit ergibt sich die folgende Definition für Dokumente:

Definition der Dokumente einer Instanz

Ein **Dokument** ist ein Ausgabeobjekt einer Instanz. Das Dokument gliedert sich in folgende Teile:

- Name der erzeugenden Instanz **I**, als Name des Rechners und des Programms,
- Nachricht N der Instanz,
- Signatur $\mathbf{S_I}$ der Instanz für diese Nachricht als $S_I = s_I(N)$.

Die Nachricht besteht hierbei aus einem **Präfixteil** (N^P), in dem die Eingabedokumente (und somit auch die Art des Auftrags) enthalten sind, und einem **Nutzdatenteil** (N^N), der die erzeugten Daten enthält. Falls zur Erbringung der geforderten Leistung andere Instanzen benutzt werden, sind im Präfixteil die Ausgabedokumente der benutzten Instanzen enthalten. Der Präfixteil eines Dokuments entspricht somit einem Protokoll seiner Entstehungsgeschichte.

Diese Struktur wird in Bild 3 anschaulich dargestellt, wobei die betrachtete Instanz X Dokumente von den Instanzen A_i, $i \in \{1,..,k\}$ erhält.

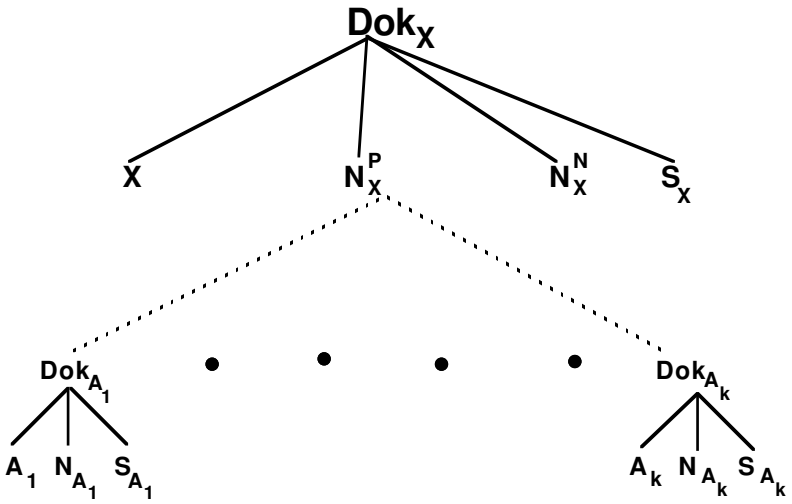

Bild 3: Struktur eines Dokuments

Eine Instanz kann mehrere Dokumente erzeugen und alle diese Dokumente besitzen eine solche Struktur.

Falls mehrere Instanzen nacheinander an der Entstehung eines Dokuments beteiligt sind, entsteht eine Hierarchie von Dokumenten. Mit dieser Hierarchie von Dokumenten kann die Verantwortung an eine Instanz zugewiesen werden. Die Zuweisung von Verantwortung an Instanzen wird in Kapitel 3 betrachtet.

Wenn nur die E/A-Relation einer Instanz überprüft werden muß, reduziert sich das Präfix auf die signierten Nutzdaten der Eingabedokumente und repräsentiert somit die Eingabe der Instanz. Für diesen Fall kann eine reduzierte Struktur der Dokumente benutzt werden, vgl. Bild 4.

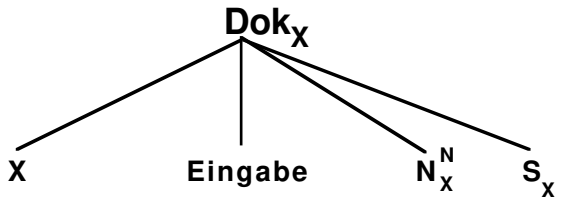

Bild 4: Struktur eines reduzierten Dokuments

Weil das Präfix in Dokumenten sehr groß werden kann, müssen Maßnahmen betrachtet werden, die diesen Teil reduzieren. Deshalb sollen im nächsten Unterkapitel die möglichen Optimierungen hinsichtlich der Größe des Präfixes untersucht werden.

2.2.2 Optimierungen der Größe des Präfixes

Weil die Dokumente aufgrund des Präfixes sehr groß werden können, sollen hier zwei Optimierungsverfahren vorgestellt werden, mit denen es möglich ist, die Dokumentengröße zu verkleinern und trotzdem die Möglichkeit zur Rekonstruktion der gesamten Entstehungsgeschichte zu erhalten.

Das erste Verfahren benutzt *Verweise innerhalb des Dokuments*, um Daten zu verlagern. Wenn Teile der Eingabedaten in den Ausgabedaten benutzt werden, werden die zu verlagernden Teile des Präfixes durch entsprechende Verweise ersetzt, und dieser Teil wird im Nutzdatenteil entsprechend markiert.

Beim zweiten Verfahren werden neue Dokumente generiert, die auf das Originaldokument verweisen. Diese Dokumente werden deshalb Verweisdokumente genannt. Dieses Verfahren soll im folgenden näher betrachtet werden, da ein solches Verweisdokument vielfältigen Einsatz finden kann.

Kollisionsfreie Hash-Funktionen zum Erzeugen sicherer Merkmale

Eine Funktion h heißt **kollisionsfrei**, wenn es unmöglich ist, eine Kollision von h zu finden, d.h. zwei Nachrichten $N_1 \neq N_2$ mit $h(N_1) = h(N_2)$.

Hash-Funktionen bilden beliebig lange Argumente auf Bilder konstanter Länge ab.

Funktionen, die sowohl kollisionsfrei als auch Hash-Funktionen sind, heißen **kollisionsfreie Hash-Funktionen**. Sie können kollisionsfrei allerdings nur unter einer kryptographischen Annahme sein, da eine Hash-Funktion nicht injektiv sein kann. Für die beschriebene Anwendung ist es wünschenswert, daß eine global feste Funktion h verwendet werden kann, die also für alle Beteiligten kollisionsfrei sein muß. Deshalb bietet es sich an, von den beiden üblichen kryptographischen Annahmen „Faktorisieren gegebener Zahlen, die aus zwei genügend langen Primfaktoren bestehen“ und „das Ziehen diskreter Logarithmen“, ist zu schwierig, letztere zu verwenden [Damg_88].

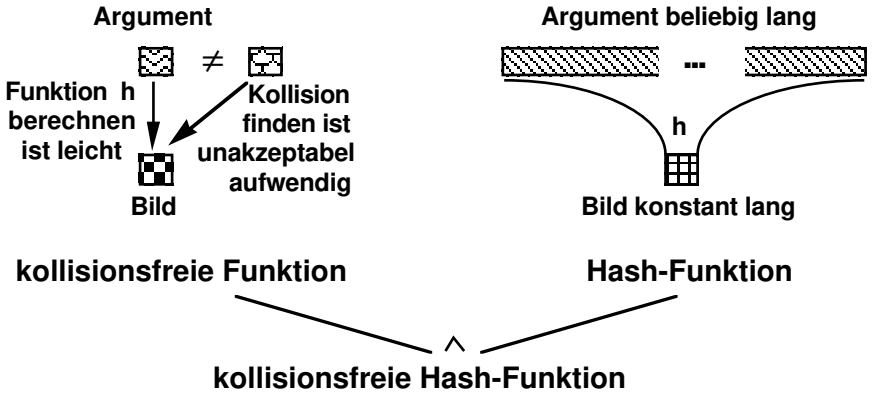

Bild 5: Kollisionsfreie Hash-Funktion

Es wird folgende Notation zur Definition des Verweisdokuments benutzt:

- **Y** sei eine Instanz, die das Originaldokument (welches auch ein Verweisdokument sein kann) $\mathbf{Dok}_Y^o$ erzeugt.
- **h** sei eine *kollisionsfreie Hash-Funktion* (vgl. Kasten), die für jede Nachricht ein eindeutiges Merkmal berechnet.
- **X** sei die Instanz, die das Verweisdokument erzeugt.

Definition eines Verweisdokuments

Ein **Verweisdokument** Dok^V_X ist ein Dokument, das auf ein Originaldokument Dok^O_Y, das nicht in Dok^V_X enthalten ist, so verweist, daß die Eindeutigkeit von Dok^O_Y sichergestellt ist. Dok^O_Y kann selbst wieder ein Verweisdokument sein. Am Ende der Verweiskette muß sich stets ein „normales" Dokument befinden.

Ein solches Verweisdokument besteht aus dem Namen der das Verweisdokument herstellenden Instanz (X), dem eindeutigen Merkmal des Originaldokuments ($h(Dok^O_Y)$) und der Signatur (S_X) der Instanz für dieses Verweisdokument.

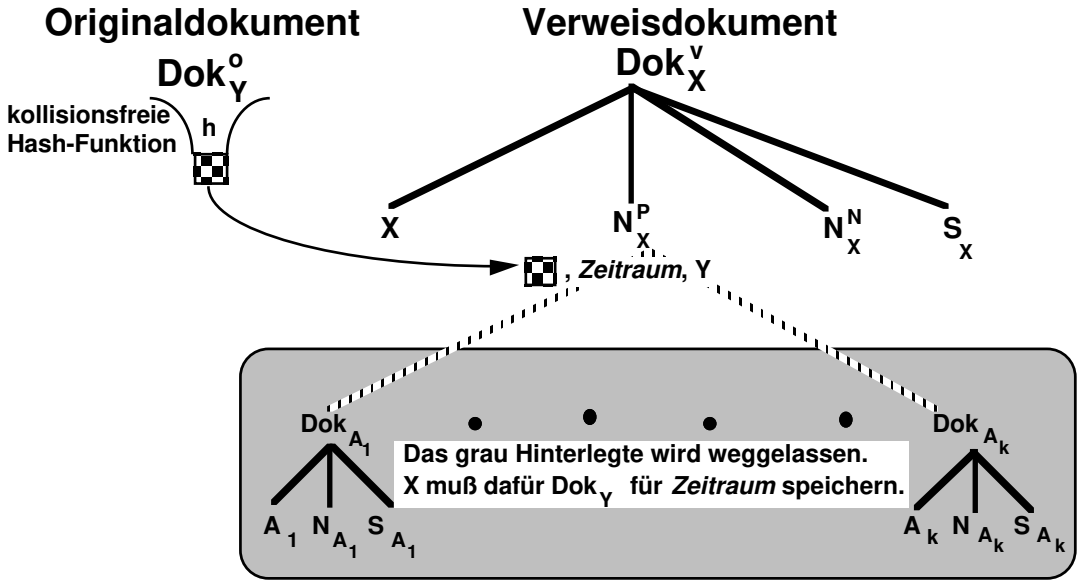

Bild 6: Struktur eines Verweisdokuments

Der Hersteller des Verweisdokuments übernimmt für einen zuvor definierten Zeitraum die Verantwortung zur Speicherung des Originaldokuments, d.h. er muß das Originaldokument bei Bedarf in diesem Zeitraum vorlegen können. Deshalb muß er das Originaldokument für obigen Zeitraum vor Verlust schützen (z.B. auf privatem Permanentspeicher). Im Verweisdokument sind noch Verwaltungsinformationen (z.B. Zeitraum, während dem das Originaldokument gespeichert wird; Name der Instanz, die das Originaldokument erzeugt hat) enthalten.

Mit den Verweisdokumenten können Dokumente verlagert werden (sinnvoll bei großen Dokumentinhalten), vgl. Bild 6. Verweisdokumente können deshalb zum Beispiel für Optimierungen, Aufträge und Quittungen benutzt werden.

Es bietet sich an, spezielle Protokollierungsinstanzen als Archive für Originaldokumente und Generierer für Verweisdokumente zu schaffen. Eine Protokollierungsinstanz speichert in Verwahrung gegebene Originaldokumente für vorgegebene Zeiträume und generiert entsprechende Verweisdokumente. Bei Vorlage eines ihrer Verweisdokumente liefert jede Protokollierungsinstanz eine Kopie des zugehörigen Originaldokuments.

Im folgenden wird kurz die Eigenschaft eines Verweisdokuments erläutert:

Aufgrund der Eigenschaften von h können mit akzeptablem Aufwand keine zwei verschiedenen Nachrichten N_1 und N_2 gefunden werden, so daß $h(N_1)=h(N_2)$ ist. Da das Verweisdokument von X unterschrieben ist, kann nur er der Hersteller des Verweisdokuments sein. Deshalb muß er das Originaldokument Dok^O_Y (und wegen h nur genau dieses) einer dritten Instanz zur Überprüfung im spezifizierten Zeitraum vorlegen können.

Die Instanzen X und Y müssen nicht verschieden sein.

Mit den Dokumenten ist es möglich, den Instanzen bzw. deren Herstellern die Verantwortung zuzuweisen. Deshalb soll im nächsten Unterkapitel die Verantwortung von Herstellern und eine mögliche Verantwortungszuweisung betrachtet werden.

3 Verantwortungsbestimmung

In diesem Kapitel soll die Verantwortung eines Herstellers definiert werden, um die Grundlage für eine Schuldzuweisung zu schaffen. Da bei realen Systemen durch technologische Rahmenbedingungen eine eindeutige Verantwortung (d.h. die Schuld ist mit Wahrscheinlichkeit 1 erwiesen) im allgemeinen nicht möglich ist, muß die Verantwortung zu einer (1-ε)-Verantwortung (wobei ε klein ist) abgeschwächt werden.

Eine Verantwortungszuweisung findet dann statt, wenn eine Instanz eine nichtspezifizierte Leistung erbracht hat, d.h. der Aspekt der Verweigerung einer Diensterbringung wird hier nicht betrachtet. Muß die Verweigerung einer Diensterbringung behandelt werden, so kann man die beschriebenen Protokolle um unterschriebene Dienstleistungsangebote und Auftragsbestätigungen ergänzen. Erhält eine Instanz innerhalb der vereinbarten Zeit keine Auftragsbestätigung, so muß sie ihren Auftrag gegenüber einem unabhängigen Zeugen wiederholen, der den Auftrag dann weitergibt. Wird das Ausbleiben einer Auftragserledigung moniert, muß der Auftragnehmer sein Arbeitsergebnis ebenfalls über einen unabhängigen Zeugen dem Auftraggeber zukommen lassen.

3.1 Verantwortung

Die Aufgabe des Programmherstellers besteht darin, zu einer gegebenen Spezifikation ein Programm zu erzeugen, das auf einem bestimmten Rechner diese Spezifikation erfüllt.

Im folgenden betrachten wir auch indeterministische Programme, allerdings ohne die Wahrscheinlichkeitsverteilung ihres Verhaltens. Probabilistische Programme, die eine Lösung nur mit einer gewissen Wahrscheinlichkeit finden, oder die z.B. einen Schlüssel eines Kryptosystems gleichverteilt wählen sollen, werden also nicht betrachtet.

Damit auch indeterministische Programmläufe nachgerechnet werden können, werden die Zufallszahlen für die zu treffenden Zufallsentscheidungen als Eingaben aufgefaßt und damit im Ausgabedokument aufgeführt.

Um diesen Sachverhalt präziser auszudrücken, wird folgendes Modell benutzt:

- Sei **R** eine Relation, zwischen den Eingabedaten **E** und den Ausgabedaten **A**, d.h. $\mathbf{R} \subseteq \mathbf{E} \times \mathbf{A}$. Diese Relation soll mit einer Instanz realisiert werden.
- Diese Instanz wird durch ein korrektes Programm mit $\mathbf{Spez_{P_Soll}}$ und einen korrekten Rechner mit $\mathbf{Spez_{Rech_Soll}}$ definiert.
- Tatsächlich ist aber vom Programmhersteller $\mathbf{Spez_{P_Ist}}$ und vom Rechnerhersteller $\mathbf{Spez_{Rech_Ist}}$ realisiert.

Mit diesen Notationen kann nun die Verantwortung der Hersteller einer Instanz definiert werden.

Definition der Verantwortung

Die **Verantwortung des Rechnerherstellers** besteht in der Realisierung eines Rechners, so daß $Spez_{Rech_Ist} = Spez_{Rech_Soll}$.

Die **Verantwortung des Programmherstellers** besteht in der Realisierung eines Programms, so daß $Spez_{P_Ist} = Spez_{P_Soll}$.

Somit wird die **Verantwortung der Instanz** durch die Verantwortung des Rechner- und des Programmherstellers realisiert, denn $Spez_{Rech_Soll}$ und $Spez_{P_Soll}$ erzeugen eine Teilmenge von R.

Mit dieser Definition kann die Verantwortung an eine Instanz zugewiesen werden, falls man in ihren Ausgabedokumenten ein Tupel (e,a) $\notin$ **R** findet.

Um die Verantwortung auch bei nicht reproduzierbaren falschen Ausgaben zuweisen zu können, werden Dokumente benötigt, denn es muß auch die Verantwortung zugewiesen werden können, wenn der Rechner nur transient fehlerhaft ist.

3.2 Verantwortungszuweisung

Um die Verantwortung innerhalb einer Instanz zuweisen zu können, sollen zuerst die verschiedenen Möglichkeiten von fehlerhaften Ausgaben einer Instanz untersucht werden.

Dazu soll die folgende Tabelle mit allen verschiedenen Fehlerfällen benutzt werden. Die Einträge der Tabelle sagen aus, ob (e,a) $\in$ **R** oder (e,a) $\notin$ **R**, wenn eine *geeignete* Eingabe **e** (d.h. enthalten Programm bzw. Rechner Fehler, so ist dies mit e von einem Absoluttest erkennbar) für jeden Tabelleneintrag gewählt und mit dieser Eingabe e die Ausgabe a erzeugt wurde. Die Beschränkung auf geeignete Eingaben erfolgte, da nicht bei allen e $\in$ **E** Fehler beobachtbar sind, denn für die meisten Eingaben wird eine Instanz die Spezifikation erfüllen und somit ist nichts zu tun.

Programm→ ↓ Rechner	fehlerfrei	fehlerhaft
fehlerfrei	$\in$	$\notin$
fehlerhaft	$\notin$	$\notin$ oder $\in$

Der Fall „Programm und Rechner sind fehlerhaft“ kann auch ein korrektes Resultat erzeugen, da sich die Fehler eventuell kompensieren. Dies wird aber selten auftreten, wenn der Ausgaberaum genügend groß ist.

Aus dieser Tabelle folgt:

Falls es einen korrekten Rechner gibt, der im folgenden **Referenzrechner** genannt wird, dann kann die Verantwortung bei einer falschen Ausgabe richtig zugewiesen werden.

Definition der Verantwortungszuweisung

Wenn mit einer Eingabe e eine falsche Ausgabe a erzeugt wurde, d.h. (e,a) $\notin$ **R**, dann kann die **Verantwortung** an die Instanz **zugewiesen** werden.

Wenn ein Referenzrechner mit der Eingabe e und dem Programm der fehlerhaften Instanz (und ggf. den von ihr getroffenen Zufallsentscheidungen) die Ausgabe a_{Ref} erzeugt, dann wird die **Verantwortung** innerhalb der Instanz an Programm- bzw. Rechnerhersteller folgendermaßen **zugewiesen**.

Wenn a_{Ref} richtig ist, d.h. (e,a_{Ref}) $\in$ R, dann wird dem Rechnerhersteller die Verantwortung zugewiesen.

Wenn a_{Ref} falsch ist, werden folgende Fälle unterschieden:

- Wenn a = a_{Ref} ist, dann wird dem Programmhersteller die Verantwortung zugewiesen.
- Wenn a $\neq$ a_{Ref} ist, dann wird dem Rechnerhersteller und dem Programmhersteller die Verantwortung zugewiesen.

Bemerkung:

Wenn der Rechner bei einer Wiederholung mit Eingabe e (und ggf. den getroffenen Zufallsentscheidungen), mit der notwendigen Vorgeschichte, falls interne Zustände benutzt werden, eine andere Ausgabe a' liefert, dann kann dem

Rechnerhersteller die Verantwortung immer zugewiesen werden. Solche Wiederholungsexperimente sind aber nur bei transienten Fehlern des Rechners geeignet.

Der entscheidende Nachteil an dieser Art der Verantwortungszuweisung ist die Notwendigkeit der Existenz eines Referenzrechners, der die Spezifikation $Spez_{Rech_Soll}$ erfüllt.

In der Praxis ist ein solcher Referenzrechner bestenfalls auf dem Papier gegeben und deshalb muß nach anderen Möglichkeiten gesucht werden, um die Verantwortung zuzuweisen.

3.3 Verantwortung bei Approximation eines Referenzrechners

Das Problem der Existenz eines Referenzrechners gehen wir an, indem wir versuchen, einen Referenzrechner zu approximieren. Hierfür benutzen wir Redundanztechniken und statt der Verantwortung wird eine abgeschwächte Verantwortung benutzt, die ($1-\varepsilon$)-Verantwortung genannt wird.

3.3.1 Approximation eines Referenzrechners

Wir versuchen, mit diversitär entwickelten Rechnern (im folgenden **Ersatzrechner** genannt) einen Referenzrechner für eine *bestimmte* Eingabe e zu realisieren. Der explizit betrachtete Rechner, dessen Schuld bzw. Unschuld bewiesen werden soll, wird **Primärrechner** genannt.

Wir benutzen das folgende vereinfachende Modell zur Approximation des Referenzrechners:

Es seien 2n+1 diversitär entwickelte Rechner (1 Primärrechner und 2n Ersatzrechner) gegeben, die für eine bestimmte Eingabe e mindestens mit der Wahrscheinlichkeit α ein korrektes Resultat a erzeugen, d.h. $P((e,a) \in \mathbf{R}) \geq \alpha$ für $e \in \mathbf{E}$.

Die Korrektheit der Ausgaben a_j der $Rechner_j$ sei dabei stochastisch unabhängig, was für die Realität eine gewagte Annahme sein dürfte; $j \in \{1,..,2n+1\}$; dies entspricht dann einer Binomialverteilung.

Mit diesem Modell kann für *eine Eingabe e ein Referenzrechner* mit der Wahrscheinlichkeit $\mathbf{P_{Ref}}$ durch verschiedene Entscheidungsfunktionen *simuliert* werden.

Im folgenden soll die absolute Mehrheit als Entscheidungsfunktion benutzt werden. Dies ist auch für indeterministische Programme möglich, wenn die Ersatzrechner die vom Primärrechner getroffenen Zufallsentscheidungen verwenden. Hiermit ergibt sich für eine Eingabe e:

$$P_{Ref} = P(\exists J \subseteq \{1,..,2n+1\}: |J| \geq n+1 \text{ und } (e,a_j) \in \mathbf{R} \text{ für } j \in J)$$

Die Wahrscheinlichkeit P_{Ref} wird mit

$$P_{Ref} \geq \sum_{i=n+1}^{2n+1} \binom{2n+1}{i} \alpha^i (1-\alpha)^{2n+1-i} \quad \text{für } \alpha \in [0,1), \text{ und}$$

$$P_{Ref} = 1 \quad \text{für } \alpha = 1$$

berechnet.

Damit die Wahrscheinlichkeit, daß kein Referenzrechner approximiert werden kann, klein wird, muß $\alpha > 0{,}5$ und n genügend groß sein. P_{Ref} ist eine Funktion von n und α, d.h. $P_{Ref} = f(n,\alpha)$. Im folgenden wird $\delta_{\mathbf{n}} := 1 - P_{Ref}$ bzw. $\delta_n := 1 - f(n,\alpha)$ bei 2n+1 Rechnern benutzt.

Dabei gilt für P_{Ref} bzw. δ_n:

$$\lim_{n\to\infty} \delta_n = 0 \qquad \text{für } \alpha > 0{,}5$$

Beispiel: Für $\alpha = 0{,}9$ und $n = 1$ bzw. $n = 2$ erhält man $P_{Ref} = 0{,}972$ bzw. $P_{Ref} \approx 0{,}991$.

3.3.2 Verantwortungszuweisung

Im folgenden wird die Zuweisung der (1-δ)-Verantwortung und die (1-ε)-Verantwortung definiert, die die Wahrscheinlichkeit der Existenz eines Referenzrechners für eine Eingabe e mit obiger Entscheidungsfunktion ausnutzt.

Die Zuweisung der (1-δ)-Verantwortung erfolgt nur bei der Existenz eines approximierten Referenzrechners für eine Eingabe e. Hierbei ist δ die Wahrscheinlichkeit, bei der *keine* Verantwortung zugewiesen wird, d.h. es wird *vergeblich versucht,* die Verantwortung zuzuweisen.

Falls kein Referenzrechner approximierbar ist, kann zum Beispiel die Zuweisung vertagt werden und ein erneuter Versuch der Zuweisung mit einer größeren Anzahl von Ersatzrechnern kann erfolgen.

Definition der Zuweisung der (1-δ)-Verantwortung

Sei eine Konstante $\delta \in (0,1)$ gegeben.

Die (1-δ)-Verantwortung wird zugewiesen, wenn mit einer Wahrscheinlichkeit $P_{Ref} \geq 1-\delta$, d.h. $\delta_n \leq \delta$, ein approximierter Referenzrechner für die Eingabe e gemäß einer Entscheidungsfunktion existiert.

Wenn ein solcher Referenzrechner existiert, erfolgt die Zuweisung der (1-δ)-Verantwortung analog zur oben definierten Verantwortungszuweisung.

Im folgenden wird die Zuweisung der (1-ε)-Verantwortung definiert, wobei immer eine Verantwortung zugewiesen werden soll, oder schärfer formuliert: Der Fall keiner Zuweisung sei nicht erlaubt. Hierbei ist ε die zulässige Wahrscheinlichkeit, mit der die Verantwortung *falsch* zugewiesen wird.

Wenn $\alpha > 0{,}5$ ist und genügend viele Rechner vorliegen, wird die Wahrscheinlichkeit ε_n für eine falsche Zuweisung der Verantwortung mit dieser Entscheidungsfunktion klein. Die Wahrscheinlichkeit ε_n kann mit $\varepsilon_n = (1-P_{Ref})(1-\alpha)$ bzw. $\varepsilon_n = (1-P_{Ref})\alpha$ für $\alpha \geq 0{,}5$ bzw. $\alpha < 0{,}5$ berechnet werden.

Definition der Zuweisung der (1-ε)-Verantwortung

Sei eine Konstante $\varepsilon \in (0,1)$ gegeben.

Die (1-ε)-Verantwortung wird zugewiesen, wenn die Wahrscheinlichkeit $\varepsilon_n \leq \varepsilon$ ist.

Falls ein approximierter Referenzrechner für die Eingabe e gemäß einer Entscheidungsfunktion existiert, erfolgt die Zuweisung der (1-ε)-Verantwortung analog zur oben definierten Verantwortungszuweisung.

Falls kein Referenzrechner existiert, heißt der Primärrechner „fehlerfrei“ $:\Leftrightarrow \alpha \geq 0{,}5$.

In diesem Fall wird dem Programmhersteller die **(1-ε)-Verantwortung zugewiesen**, wenn der Primärrechner „fehlerfrei“ ist; ansonsten dem Rechnerhersteller.

Die (1-ε)-Verantwortung bzw. die (1-δ)-Verantwortung setzt die Existenz mehrerer diversitär entwickelter Rechner mit derselben Soll-Spezifikation $Spez_{Rech_Soll}$ voraus. Hierbei wird nicht die Korrektheit des Rechners, sondern nur die richtige Ausgabe bei einer bestimmten Eingabe vorausgesetzt. Dies entspricht einer schwächeren Voraussetzung als der Existenz eines Referenzrechners und somit wird α im Regelfall relativ groß sein.

Die Wahrscheinlichkeiten ε und δ können durch andere Entscheidungsfunktionen noch verkleinert werden, jedoch gehen dann die Verteilungen des Ausgaberaums in die Berechnung der Wahrscheinlichkeit P_{Ref} und somit in ε und δ ein.

Weitere Entscheidungsfunktionen für die Existenz eines Referenzrechners bei einer Eingabe e sind:

- Relative Mehrheit der Ausgaben a_j.
- Bilden von Klassen mit gleichen Ausgaben. Diese Klassen werden dann bezüglich ihrer Korrektheit bewertet und die Klasse mit der höchsten Bewertung wird als Referenzrechner gewählt.

Falls keine Ersatzrechner vorhanden sind, ist eine Verantwortungszuweisung ebenfalls möglich, indem der Rechner zum Beispiel in einem „Einzelschrittmodus“ simuliert wird. Dies kann entweder „auf Papier“ erfolgen, was in der Praxis aufgrund der Programmgröße kaum möglich ist, oder mit entsprechenden Werkzeugen, die aber für eine halbwegs effiziente Simulation korrekt sein müssen.

Traut man diesen Werkzeugen nämlich nicht, muß bei jedem Einzelschritt die Spezifikation des Befehls überprüft werden. Dieser Einzelschrittmodus entspricht der Simulation eines Referenzrechners. Eine analoge Verantwortungszuweisung ist möglich.

Mit Dokumenten kann eine Verantwortungszuweisung auch bei transienten Fehlern erfolgen, da in den Dokumenten die Eingabedaten enthalten sind. Im folgenden wird die technische Realisierung der Dokumente und der nötigen Signierrechner betrachtet.

4 Technische Realisierung

In diesem Kapitel wird die Signatur einer Instanz untersucht und es werden verschiedene Realisierungsmöglichkeiten vorgeschlagen und bewertet. Danach wird die Struktur der möglichen Rechner aufgezeigt; diese neuen Rechner werden Signierrechner heißen.

4.1 Signatur einer Instanz

In den vorherigen Kapiteln wurde dargestelt, daß eine Instanz ihre gesamten Ausgaben (mit den Eingaben) unterschreiben muß. Aus der Definition des Signaturschemas (siehe erster Kasten) geht hervor, daß eine Signatur nur mit einem Geheimnis – im folgenden als **Signiergeheimnis** bezeichnet – realisiert werden kann (in Bild 1 ist dies die Zufallszahl oben rechts).

Der Rechner und das Programm können solche Signiergeheimnisse benutzen. Hierbei stellt sich die Frage, in welcher Reihenfolge signiert werden soll und wie man ein solches Geheimnis sicher speichern kann. Diese Frage wird im nächsten Unterkapitel untersucht.

Da bei dieser Realisierung erhebliche Probleme auftreten, muß nach anderen Lösungen gesucht werden; dies wird auf die Realisierung mit einem eindeutigen Merkmal führen und soll im übernächsten Unterkapitel betrachtet werden.

4.1.1 Realisierung mit Signiergeheimnis

Wie oben erwähnt, benötigt man für eine sichere Signatur (d.h. nicht fälschbar durch einen Angreifer) ein Geheimnis, das vor möglichen Angreifern verborgen sein muß.

Da auch der Rechner unterschreibt, benötigt auch er ein solches Signiergeheimnis. Dieses kann in einer unausforschbaren Komponente (siehe Kasten) gespeichert sein und ist somit vor allen Angreifern aufgrund der Eigenschaften der unausforschbaren Komponente sicher. Der Rechner unterschreibt mit der Signierfunktion s_{Rech}, die auf dem Signiergeheimnis des Rechners basiert; er wird deshalb „Signierrechner“ genannt.

Da aber das Programm ebenfalls unterschreiben bzw. das Programm eindeutig mit der Signatur der Instanz bestimmt sein muß, muß ein Programm ebenfalls ein Signiergeheimnis benutzen. Da das Programm aber erst mit dem Rechner aktiv werden kann, kann ein Programm seine Ausgaben nicht selbständig unterschreiben. Deshalb kennt der Rechner im allgemeinen das Signiergeheimnis des Programms; somit ist das Signiergeheimnis kein „Geheimnis“ mehr.

Da der Rechner im allgemeinen alle Signiergeheimnisse der von ihm ausgeführten Programme kennt, kann er Mißbrauch betreiben.

Ein weiterer Nachteil bei dieser Art der Signatur einer Instanz besteht in der Speicherung des Signiergeheimnisses des Programms. Wenn beim ausführenden Signierrechner keine Maßnahmen für SW-Schutz vorhanden sind, muß dieses Signiergeheimnis irgendwo im Programm im Klartext enthalten sein.

Ein solches Verfahren mit Signiergeheimnis ist nur dann durchführbar, wenn SW-Schutz betrieben wird und wenn der Programmhersteller dem Rechnerhersteller vertraut.

Deshalb wird im nächsten Unterkapitel eine weitere Möglichkeit zur Realisierung der Signatur einer Instanz betrachtet.

Unausforschbare Gehäuse zur Beschränkung der Eingriffsmöglichkeiten des Betreibers

Alle technischen Schutzmaßnahmen bedürfen einer physischen „Verankerung“ in Form eines Systemteils, auf den der betrachtete Angreifer keinen physischen Zugriff hat.

Für die beschriebene Anwendung sollte man versuchen, Gerätegehäuse so zu bauen, daß ein nur durch die derzeit bekannten physischen Naturgesetze beschränkter Angreifer weder unbefugt an die in den Gerätegehäusen gespeicherte Information gelangen noch diese unerkannt unbefugt ändern kann. Ersatzweise können die Ressourcen des Angreifers auf beispielsweise 50 Jahre Zeit und 10^9 DM Kapital beschränkt werden. Aber auch die Realisierung derart nur bedingt unausforschbarer Gehäuse ist technisch aufwendig. Die Sicherheit unausforschbarer Gehäuse ist generell extrem schwer zu validieren.

Physische Schutzmaßnahmen bestehen aus einer Kombination der folgenden Grundfunktionen:

Passive Angriffe, etwa Analyse elektromagnetischer Strahlung, müssen durch *Schirmung* vereitelt werden. (Verfahren und Normen sind unter dem Kürzel TEMPEST bekannt [Horg_85].)

Aktive Eingriffe müssen *erkannt* und bezüglich Zulässigkeit *bewertet* werden.

Aktive Angriffe, d.h. als unzulässig erkannte aktive Eingriffe, müssen *verzögert* und die Informationen müssen vor Ende der Verzögerung ggf. *gelöscht* werden.

Abhängig von Abmessungen und Komplexität ist es sinnvoll, den physisch zu sichernden Systemteil *schalenförmig* zu gestalten [Chau_84]. Hierbei können in einer Schicht, wie z.B. in Bild 7 gezeigt, mehrere Grundfunktionen untergebracht werden. Ebenso können Grundfunktionen in einer Schicht mehrfach erbracht werden, etwa die Erkennung von Eingriffen durch Sensoren für unterschiedliche physische Größen. Alternativ oder als Ergänzung können dieselben Grundfunktionen wiederholt in weiter innen liegenden Schichten auftreten. Dies ist in Bild 7 anhand von 3 Beispielen für den Fall maximal zweifach wiederholten Auftretens von Grundfunktionen gezeigt.

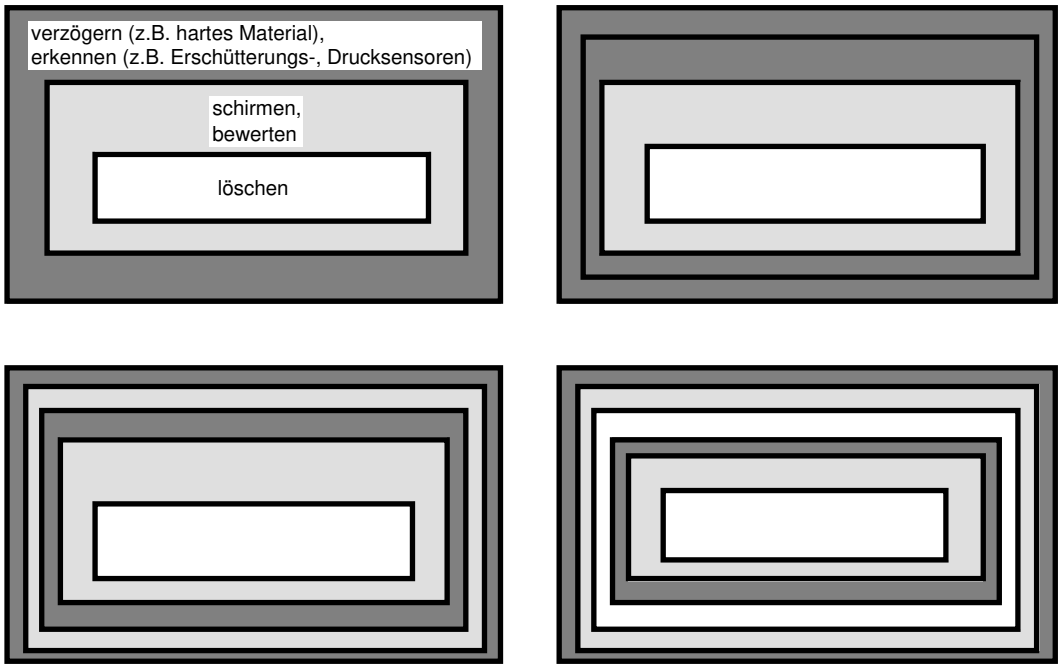

Bild 7: Schalenförmige Anordnung der fünf Grundfunktionen (oben links) und drei Möglichkeiten ihrer wiederholten Anwendung

4.1.2 Realisierung mit eindeutigem und unfälschbarem Merkmal

Weil ein Signiergeheimnis eines Programms beträchtliche Nachteile hervorruft, wird nun ein Verfahren vorgeschlagen, bei dem das Programm nicht unterschreibt, sondern durch die Unterschrift der Instanz (die das Programm umfaßt) eindeutig bestimmt ist.

Um eine sichere Unterschrift erzeugen zu können, wird wiederum ein sicheres Geheimnis benötigt, das wie bei der obigen Realisierung in einer unausforschbaren Komponente enthalten ist. Da der Rechner die aktive Komponente der Instanz ist, kann er mit seiner Signierfunktion alle Ausgaben unterschreiben.

Die Signatur der Instanz besteht dann aus einem eindeutigen Merkmal des Programms und aus der Unterschrift des Rechners. Dieses Merkmal soll nur eine bestimmte Größe haben (Effizienz der Größe des Merkmals); ansonsten kann das Programm selbst als Merkmal benutzt werden.

Dieses Merkmal soll mit einer öffentlich bekannten Funktion h berechnet werden können und darf nicht fälschbar sein, d.h. falls $h(P_1) = h(P_2)$ ist, dann muß $P_1=P_2$ sein. Dies würde die Injektivität von h bedeuten, und somit muß auch diese Bedingung (aufgrund der beschränkten Größe des Merkmals) abgeschwächt werden. Wenn ein Merkmal $h(P_1)$ gegeben ist, dann muß die Berechnung eines *anderen* Programms P_2 mit $h(P_1) = h(P_2)$ praktisch unmöglich sein.

Diese Forderungen werden von kollisionsfreien Hash-Funktionen erfüllt (vgl. Kasten in Kapitel 2.2.2) und somit können diese zur Extraktion von Merkmalen aus Programmen benutzt werden.

Man kann dieses eindeutige, unfälschbare Merkmal somit zur Identifikation von Programmen benutzen. Damit ergibt sich folgende Forderung an die Signatur einer Instanz:

> Die Signatur des Rechners muß das Merkmal und die Nachricht (Kapitel 2) als Ausgabedaten des Programms enthalten, weil nur so die Authentizität des Merkmals und der Nachricht sichergestellt werden kann, d.h. die Signatur wird mindestens mit dem Merkmal und der Nachricht berechnet. Zusätzlich kann der Name des Rechners benutzt werden. Dies ist aber nicht nötig, da der Rechner auch mit seiner Signierfunktion bzw. Testfunktion charakterisiert werden kann.

Der Name einer Instanz ist dann der Name des Rechners und das Merkmal des Programms, die Signatur der Instanz wird mit s_{Rech}(Programm-Merkmal, Nachrichten) oder mit s_{Rech}(Rechner-Name, Programm-Merkmal, Nachrichten) gebildet. Die Struktur des Dokuments entspricht somit unverändert der Struktur aus Kapitel 2.2.

Eine andere Argumentation, die zum gleichen Ergebnis führt, ist folgende: Für den Rechner stellt das Programm eine Eingabe dar, die in seinen Ausgabedokumenten wiederholt werden muß. Um nicht immer das ganze Programm wiederholen zu müssen, verwendet der Rechner grundsätzlich Verweisdokumente, die auf das Programm verweisen.

Im nächsten Kapitel sollen diese beiden Realisierungen verglichen werden, um die Vorteile der Realisierung mit Merkmal hervorzuheben.

4.1.3 Vergleich der beiden Realisierungen

Hier sollen beide Realisierungen der Signatur einer Instanz zusammenfassend verglichen werden. Beide Realisierungen benötigen mindestens ein sicheres Geheimnis, das in einer unausforschbaren Komponente (z.B. Rechner) gespeichert werden muß.

Signatur mit Merkmal	Signatur mit Geheimnis
+ Es wird kein Geheimnis eines Programms benötigt.	- Ein Geheimnis des Programms wird benötigt, welches im allgemeinen nicht vor dem Rechner verborgen werden kann.
+ Es ist nur *eine* „echte“ Signatur notwendig.	- Eine *doppelte* Signierung ist notwendig.
+ Eine Änderung des Programms entspricht einer Änderung der Signatur.	

Beide Varianten können zur Realisierung der Signatur einer Instanz benutzt werden, wobei die Variante mit Merkmal eines Programms besser geeignet ist, da kein zusätzliches Geheimnis benutzt werden muß. Deshalb wird sie im folgenden verwendet.

4.2 Signierrechner

In diesem Kapitel sollen nun die Realisierungen von Rechnern beschrieben werden, die die Anforderungen an die Dokumente und Instanzen erfüllen. Ein solcher Rechner wird im folgenden **Signierrechner** genannt, weil er Daten (mit s_{Rech}) signieren und somit Dokumente erzeugen kann.

Ein Signierrechner ist hierbei eine abgegrenzte Einheit, für die *ein* Hersteller die Verantwortung übernimmt. Es kann sich hierbei um einen Prozessor, eine Chipkarte oder um einen ganzen Rechner handeln. Der Signierrechner verfügt über eine Signierfunktion s_{Rech} und die entsprechende Testfunktion t_{Rech} und die Testfunktionen der dem Rechner bekannten Instanzen.

Da jede Kommunikation des Signierrechners mit Dokumenten durchgeführt wird, sollte ein Signierrechner, um den Kommunikationsaufwand zu minimieren, ein gesamter Rechner sein.

An einen solchen Signierrechner werden für die Verarbeitung von Dokumenten die folgenden Anforderungen gestellt:

- Programme ausführen.
- Erzeugen einer Signatur für eine Instanz; dafür soll das Merkmal eines Programms zusammen mit einer echten Signatur (s_{Rech}) verwendet werden.
  Das Merkmal soll dabei mit einer kollisionsfreien Hash-Funktion **h** berechnet werden.
- Überprüfen der Signatur eines Dokuments, d.h. testen, ob es sich um ein Dokument handelt.
- Fehlermaßnahmen ergreifen, falls dies notwendig sein sollte.

Im folgenden sollen Programme, die mit dem Signierrechner zusammen eine Instanz bilden, betrachtet und die dabei auftretenden Probleme gelöst werden.

Um Programme ausführen zu können, ist ein Lader/Starter notwendig. Dieser Lader/Starter wird ebenfalls noch zum Signierrechner gezählt und stellt deshalb keine separate Instanz dar.

Bevor das geladene Programm gestartet werden kann, muß es auf Vorhandensein einer Signatur geprüft werden, da alle Eingaben in einen Signierrechner Dokumente sein sollen. Aus diesem Grund muß der Signierrechner ein initiales Testprädikat ($\neq t_{Rech}$) besitzen, mit dem er die Dokumente prüfen kann, falls diese nicht von ihm selbst erstellt wurden. Die Initialisierung eines Signierrechners wird in Kapitel 5.2.4 dargestellt.

Zu einem solchen initialen Testprädikat gehört auch eine entsprechende Signierfunktion. Mit dieser initialen Signierfunktion können fremde Programme in das (Signier-

rechner-)System eingebracht werden, d.h. diese Programme werden *ratifiziert*. Diese initiale Signierfunktion kann zum Beispiel durch eine Eingabeinstanz ausgeführt werden oder sie muß auf einem herkömmlichen Rechner oder „von Hand“ berechnet und danach in geeigneter Form (z.B. Bit für Bit) in das System eingebracht werden. Die Person oder die Eingabe-Instanz, die fremde Programme ratifiziert, wird im folgenden als **ratifizierende Instanz** (keine „echte“ Instanz) bezeichnet, da sie mit ihrer Unterschrift ebenfalls eine Verantwortung übernimmt. Im Fehlerfall wird sie die Verantwortung an die Rechner- oder Programmhersteller zuweisen.

Mit dieser ratifizierenden Instanz kann dann eine ganze Hierarchie von echten Instanzen realisiert werden, die ebenfalls Programme für gültig erklären können; dazu muß aber der Lader/Starter geändert werden. Diese Betrachtung erfolgt am Ende des Kapitels.

Bei dieser ratifizierenden Instanz würde die Signierfunktion genügen. Um jedoch strukturelle Konsistenz zu allen Dokumenten herzustellen, sollte diese Instanz ebenfalls wie eine normale Instanz unterschreiben, d.h. zusätzlich ein Merkmal benutzen.

Nachdem ein Programm geladen ist und die Berechnung des Testprädikats ein korrektes Resultat geliefert hat, ist eine Instanz lauffähig. Ansonsten kann dieses Pseudo-Programm-Dokument (nicht korrekt unterschriebene Nachricht, die aber die Struktur eines Dokuments erfüllt) vernichtet werden, weil hierfür keine verantwortliche Instanz existiert, oder es können gewisse Ausnahmebehandlungen unternommen werden.

Das Berechnen der Testprädikate garantiert dabei die Authentizität der geladenen Dokumente. Falls es sich bei dem Dokument um ein Programm handelt, muß noch dessen Merkmal berechnet werden, damit die Ausgaben der jetzt entstandenen Instanz (geladenes, ausführbares Programm und Rechner) signiert werden können. Deshalb sind bei einem Signierrechner neben einer Verarbeitungskomponente die folgenden Komponenten notwendig:

- Komponente zur Berechnung der Testprädikate (Testkomponente),
- Komponente zur Berechnung der Signierfunktion (Signierkomponente) und
- Komponente zur Berechnung des Merkmals (h Berechnungskomponente).

In Bild 8 wird ein Signierrechner dargestellt, der die obigen Komponenten enthält. Diese Komponenten können entweder als HW-Komponenten oder als Programme realisiert sein. Sie gehören aber zum Signierrechner, somit zur Verantwortung des Rechnerherstellers, und sind deshalb *im* Signierrechner dargestellt.

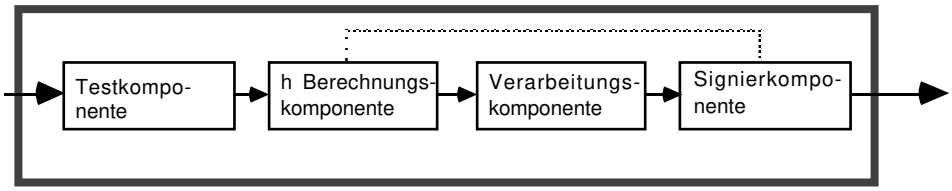

Bild 8: Signierrechner

Alle diese Komponenten müssen in einer unausforschbaren Komponente enthalten sein; ansonsten wird der Hersteller keine Verantwortung für diese Komponenten übernehmen, denn ihre Resultate wären ungeschützt und Mißbrauch durch Außenstehende (z.B. Betreiber) wäre möglich.

Mit einem solchen Rechner können nun Instanzen realisiert werden, wenn diese zusätzlichen Komponenten korrekt sind.

Die Instanzen erstellen Dokumente; das zugehörige Präfix kann entweder vom Programm oder vom Signierrechner erzeugt werden.

Falls es vom Rechner generiert wird, ist dessen Vollständigkeit sichergestellt. Da aber verschiedene Programme nicht alle Eingabedokumente im Präfix aufführen müssen (z.B. weil diese Dokumente schon irgendwo gespeichert sind oder falls Verweisdokumente zur Optimierung benutzt wurden), sollte das Präfix vom Programm erstellt werden. Um quasi die Vollständigkeit für eine spätere Verantwortungszuweisung (seitens des Rechnerherstellers) zu garantieren, kann von jedem Daten-Eingabedokument ebenfalls ein Merkmal mit **h** berechnet werden, und dieses kann dann vom Rechner zur Erweiterung des Präfixes benutzt werden.

Mit den Signaturen einer Instanz kann aber nicht nur die Authentizität der Daten sichergestellt werden, sondern es können damit auch die Zugriffsrechte kontrolliert werden, wenn der Lader/Starter modifiziert wird. Dazu werden die Dokumente nicht nur auf ein gültiges Testprädikat untersucht, sondern der enthaltene Name (Merkmal) wird ebenfalls betrachtet. Damit kann ein Dokument eindeutig einer Instanz zugewiesen werden und dadurch kann die Zugriffskontrolle durch den Lader/Starter erfolgen; er prüft, ob die Eingabedokumente von den entsprechenden Instanzen stammen. Falls dies nicht der Fall ist, können entsprechende Maßnahmen ergriffen werden.

Falls man statische Zugriffsrechte benutzt, genügt es, wenn die ratifizierende Instanz die Namen der Instanzen, von denen Dokumente benutzt werden dürfen, in einen entsprechenden Teil eines Programm-Dokuments speichert.

Für dynamische Zugriffsrechte kann eine Instanz benutzt werden, mit der der Lader/Starter kommuniziert oder es werden entsprechende Tabellenverfahren benutzt.

Mit diesen Signierrechnern kann also ein System aus Instanzen realisiert werden, die ausschließlich Dokumente zur Kommunikation benutzen.

Im nächsten Kapitel werden die Signierrechner in ein System übertragen, in dem Software-Schutz möglich ist.

5 Software-Schutz

Da nicht nur dem Benutzer von SW und HW Schaden durch deren Hersteller, sondern auch dem Hersteller Schaden durch den Benutzer (insbesondere von SW) entstehen kann (z.B. Programm-Kopien werden erstellt und ohne Zahlung einer Lizenzgebühr benutzt oder weitergegeben), müssen sich die Hersteller vor den Benutzern schützen. Deshalb soll zunächst der SW-Schutz untersucht werden. Danach werden die Anforderungen des SW-Schutzes, die auch gewisse HW-Maßnahmen erfordern, untersucht und auf Instanzen und Signierrechner übertragen.

5.1 SW-Schutz durch Verschlüsselung

Der Schaden (z.B. entgangener Gewinn, Verlust von Ideen), der durch unerlaubte Replikation von Programmen entsteht, ist nicht unbeträchtlich. Deshalb werden zunehmend SW-Schutzmaßnahmen benutzt, um eine unerlaubte Replikation zu verhindern. Da die Besitzer ihre Rechner im allgemeinen manipulieren können, sind die meisten dieser Schutzmechanismen zu brechen. Deshalb müssen für ausreichenden SW-Schutz ebenfalls unausforschbare Komponenten benutzt werden und zugleich dürfen die Daten und Programme auf den Externspeichern (z.B. Disketten, Bänder) nicht als Klartext zugänglich sein. Zudem muß auch die Ausgabe von geschützten Programmen im Klartext auf den Bildschirm unmöglich sein. Deshalb sind die meisten Werkzeuge, die in der Programmentwicklung benutzt werden, nicht direkt einsetzbar oder es müssen HW-Maßnahmen ergriffen werden, um Mißbrauch mit den gegebenen Werkzeugen durch den Besitzer zu verhindern.

Der Benutzer darf bestimmte Speicherbereiche (z.B. Schlüsselbereich) niemals als Klartext lesen und auf andere Bereiche (z.B. Programmbereich) nur manchmal zugreifen (z.B. ein eigenes Programm kann in diesem Bereich manipuliert werden, während gekaufte Programme, die geschützt sind, nicht gelesen werden dürfen) und wiederum auf andere Speicherbereiche (z.B. echte Daten) darf er beliebig zugreifen.

Ausreichender SW-Schutz [WhCo_87, Wein_87] kann aus den obigen Gründen nur mit unausforschbaren Komponenten und kryptographischen Methoden erreicht werden.

Konzelationssysteme zum Schutz vor unbefugter Kenntnisnahme

Verschlüsselung der Daten mittels eines Konzelationssystems (von lateinisch: concelare = = englisch: conceal = geheim halten) soll garantieren, daß der Inhalt einer gesendeten Nachricht nur den Besitzern eines bestimmten Schlüssels zugänglich ist. (Gute und ausführliche Einführungen in das Thema Kryptographie findet man z.B. in [Denn_82, Hors_85, Bras_88, DaPr_89], etwas ausführlichere als in diesem Kasten z.B. in [Pfit_89, PWP_90].)

Hinsichtlich der erlaubten und möglichen Verteilungen dieser Schlüssel unterscheidet man zwischen **symmetrischen** und **asymmetrischen** Konzelationssystemen. Erstere werden oftmals auch konventionelle Konzelationssysteme, letztere Konzelationssysteme mit öffentlichen Schlüsseln genannt.

In einem symmetrischen Konzelationssystem (Bild 9), wozu alle klassischen Konzelationssysteme gehören, wird die Kommunikation zwischen zwei Partnern dadurch gesichert, daß beide einen gemeinsamen geheimen Schlüssel kennen, der sowohl zur Ver- als auch zur Entschlüsselung dient. Damit wird ein Schlüssel nicht einer bestimmten Instanz, sondern einer bestimmten Kommunikationsbeziehung zugeordnet.

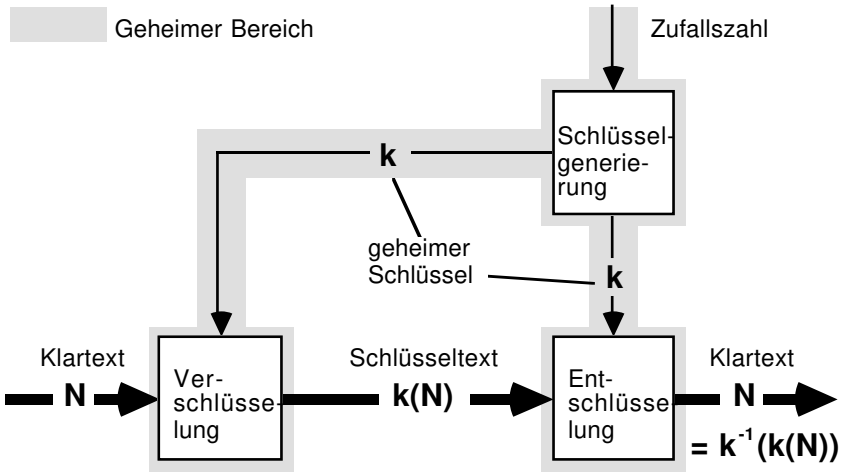

Bild 9: Symmetrisches Konzelationssystem
(= Kasten mit Schloß, es gibt zwei gleiche Schlüssel)

Die Idee der asymmetrischen Konzelationssysteme (Bild 10) wurde erst 1976 veröffentlicht [DiHe_76] und löst das Schlüsselverteilproblem auf überraschend einfache Art: Statt einen einzigen Schlüssel zum Ver- und Entschlüsseln zu verwenden, verteilt man diese Funktion auf zwei zusammengehörige Schlüssel c und d. Der Schlüssel d soll nur zum Entschlüsseln (dechiffrieren) dienen und muß natürlich geheimgehalten werden, weshalb er auch privater Schlüssel genannt wird. Der Schlüssel c hingegen soll nur das Verschlüsseln (chiffrieren), nicht jedoch das Entschlüsseln ermöglichen, weshalb er veröffentlicht werden kann und daher auch öffentlicher Schlüssel genannt wird. Insbesondere darf man also keine realistische Möglichkeit haben, ein unbekanntes d aus dem zugehörigen c herzuleiten.

Rein theoretisch könnte man d aus c natürlich durch Durchprobieren aller möglichen Schlüssel bestimmen, denn nur ein richtiges d entschlüsselt alle mit c verschlüsselten Nachrichten richtig. Es muß daher so viele Möglichkeiten geben, daß dies praktisch keinerlei Erfolg verspricht.

Außerdem müssen, damit niemand kurze Standardnachrichten erraten und mit dem öffentlichen Schlüssel testen kann, Nachrichten mehrere Verschlüsselungen

besitzen. Dies kann man erreichen, indem man die Nachrichten vor der Verschlüsselung mit zufällig gewählten Zeichenfolgen verlängert oder indem man ein bereits indeterministisch verschlüsselndes Konzelationssystem (probabilistic encryption, [GoMi_84, BlGo_85]) verwendet (hierfür dient die Zufallszahl unten links in Bild 10).

Einfacher als das Herleiten von d aus c muß hingegen das Generieren eines beliebigen Paares von zueinander passenden c und d sein, da dies jede Instanz am Anfang selbst durchführen muß.

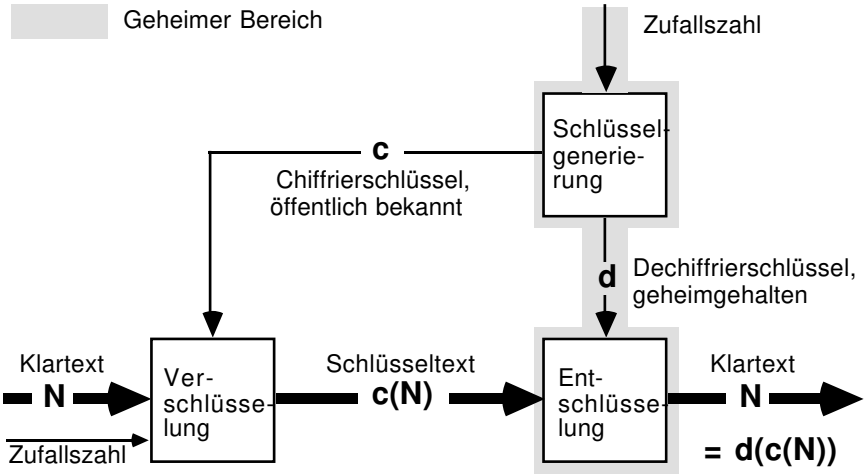

Bild 10: Asymmetrisches Konzelationssystem
(= Kasten mit Schnappschloß, es gibt nur einen Schlüssel: Jeder kann etwas hineintun, aber nur einer kann es herausholen.)

Für den SW-Schutz werden also Rechner benötigt, die mindestens ein Konzelationssystem (siehe Kasten) enthalten. Um den Schlüsselverwaltungsaufwand zu minimieren, werden sinnvollerweise asymmetrische Konzelationssysteme benutzt. Ansonsten wird für jeden Programmhersteller, von dem Programme gekauft werden, ein Schlüssel benötigt, weil dieser sonst fremde Programme in einem solchen System auswerten und mißbrauchen könnte.

Mit asymmetrischen Konzelationssystemen kann der Schlüsselverwaltungsaufwand erheblich reduziert werden, weil für jeden Rechner nur ein Chiffrierschlüssel bekannt sein muß, den alle Programmhersteller benutzen können. Der Rechner kann über einen geheimen Dechiffrierschlüssel verfügen, der niemandem (selbst nicht dem Herstellers der Rechners) bekannt sein muß.

Weil die asymmetrischen Verfahren im allgemeinen langsam sind, können sie mit symmetrischen Verfahren [BlGo_85] kombiniert werden. Jedoch müssen hierbei gewisse Rahmenbedingungen (z.B. Speichergröße und daraus resultierende „Direktzugriffe“ für Paging etc.) beachtet werden.

Der SW-Schutz kann mit asymmetrischen oder symmetrischen Konzelationssystemen realisiert werden, weil hiermit der Programmcode sicher verborgen werden kann. Es sind aber zusätzliche HW-Maßnahmen erforderlich, um neben dem SW-Schutz auch noch gegebene Rahmenbedingungen zu erfüllen. Diese ergeben sich aus den Anforderungen des Benutzers. Zum Beispiel sollten gewisse Daten auch nach dem Verlust eines solchen Rechners (entspricht dem Verlust des Dechiffrierschlüssels) weiterbenutzt werden können. Auf diesen Aspekt wird im nächsten Unterkapitel eingegangen.

Die Konzelationssysteme wurden hierbei nur zum Verbergen des Programmcodes benutzt. Wichtige Information über ein Programm (z.B. Ablaufstruktur des Programms) kann aber auch durch die Beobachtung der Kommunikation mit seiner Umwelt

(Lokalität des Zugriffs) erhalten werden. Deshalb sollte auch dieser Aspekt in die SW-Schutz-Mechanismen eingehen.

Diese Probleme werden in [Gold_87] betrachtet. Dort wird auch ein Verfahren gezeigt, das auf „zufällig erscheinenden Zugriffen“ basiert. Mit diesem Verfahren kann die Lokalität des Zugriffs und somit die Ablaufstruktur des Programms verborgen werden.

Im nächsten Unterkapitel sollen die Anforderungen, die sich aus dem SW-Schutz ergeben, mit denen für Instanzen kombiniert werden. Dort wird auch die Initialisierung von solchen Komponenten betrachtet.

5.2 Signierrechner mit SW-Schutz kombiniert

Wie im vorigen Kapitel gezeigt wurde, kann SW-Schutz mit kryptographischen Methoden und unausforschbaren Komponenten erreicht werden. Jedoch ergeben sich bei der Übertragung der Instanzen aus herkömmlichen Systemen (ohne SW-Schutz) in Systeme mit SW-Schutz Probleme, weil das Programm vor der ratifizierenden Instanz verborgen werden soll.

Insbesondere entstehen Probleme bei der Verantwortungszuweisung und beim Ratifizieren von fremden (außerhalb des betrachteten System erstellten) Programmen. Diese Probleme sollen nach der im Anschluß vorgestellten Struktur eines Signierrechners, mit dem auch SW-Schutz möglich ist, betrachtet werden; dieser wird **SW-Schutz ermöglichender Signierrechner** genannt.

Ein solcher Rechner benötigt mindestens eine Chiffrier- und Dechiffrierkomponente (c/d-Komponente) und eine Signier- und Testkomponente (s/t-Komponente). Um die Struktur zu vereinfachen, wird das Zugriffsverfahren von [Gold_87] zur Dechiffrierkomponente gezählt.

Mit der c/d-Komponente und der s/t-Komponente sind zwei logische Realisierungen von SW-Schutz garantierenden Rechnern denkbar, die sich in der Reihenfolge dieser Komponenten unterscheiden; sie besitzen folgende Ablaufstruktur:

- die Ausgaben werden vor dem Chiffrieren signiert, oder
- die Ausgaben werden vor dem Signieren chiffriert.

Für die Eingaben gilt der entsprechend inverse Ablauf.

Weil aber die Instanzen für ihre Ausgabe-Dokumente gültige Eingabe-Dokumente benötigen, müssen die Dokumente zuerst signiert und anschließend chiffriert werden. Somit ist die Struktur dieser Komponenten festgelegt. Ein solcher SW-Schutz ermöglichender Signierrechner ist in Bild 11 dargestellt, wobei wegen der Übersichtlichkeit auf die Komponente zur Berechnung des Merkmals verzichtet wurde.

Bei dieser Struktur eines Signierrechners treten Probleme bei den Programm-Dokumenten auf, die anschließend betrachtet werden.

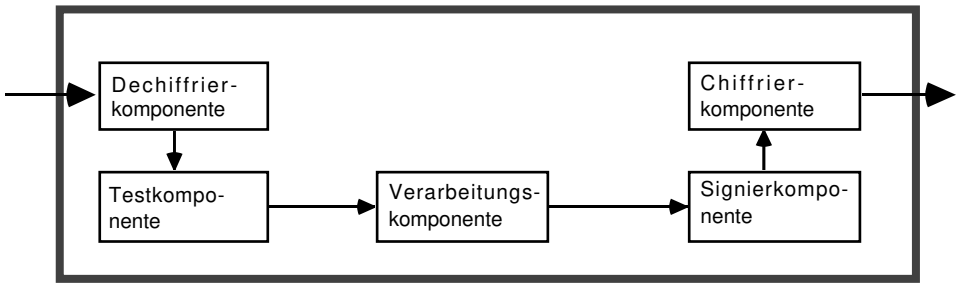

Bild 11: SW-Schutz ermöglichender Signierrechner

Der Signierrechner chiffriert alle Dokumente mit seiner öffentlich bekannten Chiffrierfunktion, außer wenn Ausgaben an Instanzen auf einem anderen solchen Rechner (z.B. Terminal) erfolgen sollen. Dann muß eine entsprechende Chiffrierung für diesen Rechner erfolgen.

Durch den geforderten SW-Schutz treten aber noch weitere Probleme auf, und zwar:

- Ratifizieren eines fremden Programms,

- Verantwortungszuweisung innerhalb einer Instanz,
- Verlust von Daten, wenn der Rechner ersetzt werden muß, und
- Initialisieren eines solchen Rechners,

auf die näher einzugehen ist.

5.2.1 Ratifizieren von Programmen

Das Ziel des SW-Schutzes ist es, alle Implementierungsdetails eines Programms zu verbergen. Deshalb sollten alle, außer dem Hersteller des Programms und dem ausführenden Rechner, nur die Spezifikation des Programms kennen. Da aber zur Realisierung einer Instanz das Programm und zur Signatur das Merkmal des Programms benötigt wird, mit dem auch die Verantwortungszuweisung erfolgt, muß die Korrektheit des Merkmals sichergestellt sein.

Da die ratifizierende Instanz fremde Programme für gültig erklärt, muß sie die Programme bzw. deren Merkmale verwalten, d.h. sie muß die Relation zwischen den Merkmalen und den Herstellern der Programme kennen, um die Verantwortung an die Hersteller zuweisen zu können. Durch HW-Maßnahmen muß erreicht werden, daß nur ratifizierte Programme, d.h. Programme, deren Merkmal der ratifizierenden Instanz bekannt sind, gestartet werden.

Durch eine Modifikation der Struktur der Programm-Dokumente und des Lader/Starters kann dies erreicht werden, nämlich durch:

1. Hinzufügen des Merkmals des Programms in das Programm-Dokument und
2. einem Vergleich, den der Lader/Starter ausführt. Es startet das Programm nur dann, wenn das vom Signierrechner berechnete Merkmal mit dem Merkmal im Dokument übereinstimmt.

Hiermit wird erreicht, daß nur Programme gestartet werden können, deren Merkmal im Dokument korrekt und der ratifizierenden Instanz bekannt ist. Dadurch kann die ratifizierende Instanz die Merkmale verwalten, ohne das Programm zur Berechnung des Merkmals zu benötigen.

5.2.2 Zuweisen von Verantwortung

Das größere Problem besteht in der Verantwortungszuweisung innerhalb einer Instanz, da jeder Rechner über einen eigenen individuellen Schlüssel verfügt. Damit ist die Verantwortungszuweisung innerhalb der Instanz schwieriger, denn das Ziel des SW-Schutzes ist es ja, daß nur der Programmhersteller den Programmcode als Klartext kennt. Deshalb soll im folgenden geklärt werden, wie eine Zuweisung der $(1-\varepsilon)$-Verantwortung (siehe Kapitel 3) erreicht werden kann, ohne daß das Programm dazu außerhalb der Rechner als Klartext vorliegen muß (richtige Verantwortung kann analog mit Referenzrechner erreicht werden).

Weil nur der Programmhersteller das Programm als Klartext kennt, kann nur dieser das Merkmal berechnen und für alle Ersatzrechner das Programm chiffrieren. Da der modifizierte Lader/Starter das Merkmal überprüft, kann der Hersteller nur dieses Programm chiffrieren, da ansonsten eine falsche Instanz (falsches Programmerkmal) entsteht.

Die Programm-Dokumente werden aber noch von einer ratifizierenden Instanz unterschrieben. Daher hat ein solches Programm-Dokument eine andere Struktur bezüglich der Reihenfolge der s/t- und c/d-Komponente und deshalb muß zwischen Programmen und Daten unterschieden werden. Beim Laden von Programmen muß zuerst die Signatur getestet werden und erst anschließend kann dechiffriert werden.

Wenn das Programm auf dem betrachteten SW-Schutz garantierenden Signierrechner hergestellt und durch eine echte Instanz (z.B. Linker) innerhalb des Systems ratifiziert wurde, also nicht von der ratifizierenden Instanz, kann die normale

Struktur der Dokumente beibehalten werden; ansonsten muß auch das obige Verfahren angewendet werden.

Somit kann die Verantwortung auch innerhalb SW-Schutz ermöglichender Systeme zugewiesen werden, ohne daß hierzu die Schutzmaßnahmen aufgegeben werden müssen.

5.2.3 Datensicherung gegen Verlust

Weil alle Ausgaben des SW-Schutz ermöglichenden Signierrechners verschlüsselt werden, muß ein Verlust der Daten durch Verlust des Dechiffrierschlüssels verhindert werden. Programme wurden bewußt für einen speziellen Rechner chiffriert, damit sie auf anderen Rechnern nicht verwendet werden können. Deshalb sollen nur die Daten vor Verlust geschützt werden.

Wegen dieser unterschiedlichen Anforderungen müssen Programme und Daten mit separaten Schlüsseln $c_{Rech_Programm}$ und c_{Rech_Daten} chiffriert werden. $c_{Rech_Programm}$ darf nicht rekonstruierbar sein, c_{Rech_Daten} muß es sein.

Für den rekonstruierbaren Schlüssel werden am besten Schwellwertschemata (siehe Kasten) benutzt. Damit kann ein Mißbrauch durch eine Minderheit ausgeschlossen werden.

Die Teile, die für die Rekonstruktion notwendig sind, sollten aber nur vertrauenswürdige Instanzen oder Personen erhalten. Diese vertrauenswürdigen Personen müssen dem Rechner bekannt gemacht werden. Dieses Problem wird bei der Initialisierung von solchen Rechnern im folgenden Kapitel betrachtet.

Schwellwertschema zur Zerlegung von Geheimnissen

Mittels eines Schwellwertschemas kann der Besitzer eines Geheimnisses G es so in n Teile zerlegen, daß die Kenntnis von mindestens k Teilen die effiziente Rekonstruktion des Geheimnisses erlaubt, k–1 Teile aber keine Information über das Geheimnis liefern.

Oftmals sollen Geheimnisse nicht nur vertraulich bleiben, sondern sie müssen auch für ihren Besitzer über lange Zeit verfügbar sein. Möchte sich der Besitzer eines Geheimnisses G vor seinem Verlust etwa durch höhere Gewalt schützen, andererseits es aber niemand anderem allein zur Aufbewahrung anvertrauen, so gibt der Besitzer des Geheimnisses n Instanzen jeweils ein unterschiedliches Teil. Verliert er sein Geheimnis, so bittet er k Instanzen, ihm Kopien ihrer Teile zu geben. Aus ihnen kann er dann G effizient berechnen. Arbeiten von den n Instanzen nur höchstens k–1 in böser Absicht zusammen, so können sie G nicht hinter dem Rücken seines Besitzers heimlich rekonstruieren, wie viel auch immer sie rechnen. Wählt der Besitzer k fast so groß wie n, kann sein Geheimnis durch Verlust von wenigen Teilen bei den gewählten Instanzen (sei es durch höhere Gewalt oder Bosheit) unrekonstruierbar werden, andererseits kann bei solcher Wahl von k in Bezug auf n sein Geheimnis fast nicht unbefugt rekonstruiert werden. Wählt der Besitzer k sehr viel kleiner als n, kann sein Geheimnis nur durch Verlust von vielen Teilen bei den gewählten Instanzen (sei es durch höhere Gewalt oder Bosheit) unrekonstruierbar werden, andererseits kann bei solcher Wahl von k in Bezug auf n sein Geheimnis wesentlich leichter unbefugt rekonstruiert werden. Durch geeignete Wahl von k und n kann der Besitzer eines Geheimnisses die Zerlegung seines Geheimnisses also an seine Bedürfnisse anpassen.

Eine Realisierung mit den zugehörigen Beweisen finden Sie in [Sham_79], Erweiterungen in [ToWo_88, BrSt_90].

5.2.4 Initialisierung

Wie schon früher angedeutet, ist die Initialisierung von Komponenten, die Geheimnisse auch vor ihren Herstellern verbergen können, ein Problem.

Beim SW-Schutz ermöglichenden Signierrechner treten dabei folgende Probleme auf:

- Initialisieren bzw. Erzeugen von s_{Rech} und t_{Rech}
- " $c_{Rech_Programm}$ und $d_{Rech_Programm}$
- " c_{Rech_Daten} und d_{Rech_Daten}
- Initialisierung des initialen Testprädikats $\mathbf{t_{init}}$ (der ratifizierenden, vertrauenswürdigen Instanz)

Die Initialisierung des Signiergeheimnisses für s_{Rech} (und somit t_{Rech}) muß nicht durch den Hersteller, sondern kann durch den Rechner selbst erfolgen, da dieses Geheimnis nur zum Signieren benötigt wird. Der Hersteller kann nach der Selbstinitialisierung des Rechners die Verantwortung für den Rechner übernehmen, weil der Rechner t_{Rech} ausgeben kann.

Eine solche Selbstinitialisierung kann ebenfalls für $d_{Rech_Programm}$ erfolgen, denn diese Funktion wird nur zum Entschlüsseln von Programmen für diesen speziellen Rechner benötigt. Diese Programme sind mit dem öffentlich bekannten Schlüssel $c_{Rech_Programm}$ verschlüsselt.

Die Selbstinitialisierung kann mit nicht beeinflußbaren, variierenden Umgebungsbedingungen des Rechners erfolgen, mit denen Indeterminismus (bezogen auf jeden realen Beobachter) im Rechner erzeugt werden kann. Ansonsten kann die Wahl der Schlüssel beeinflußt werden und die Sicherheit des Schlüssels ist gefährdet.

Die Selbstinitialisierung kann auch mit Schaltungen innerhalb des Rechners erfolgen. Dazu können die bei der Produktion von Bauteilen auftretenden Streuungen benutzt werden.

Da die Daten vor Verlust bei Rechnerverlust oder -defekt geschützt werden sollen, muß d_{Rech_Daten} rekonstruierbar sein. Hierzu werden, wie oben angedeutet, Schwellwertschemata benutzt.

Für die Realisierung von d_{Rech_Daten} gibt es folgende zwei Möglichkeiten:

- d_{Rech_Daten} wird durch Selbstinitialisierung gewählt und die entsprechenden Teile des Geheimnisses an vorbestimmte Benutzer gegeben, oder
- d_{Rech_Daten} wird durch die Eingabe der entsprechenden Teile vom Rechner berechnet.

Bei beiden Verfahren werden Instanzen benötigt, die die entsprechenden Schlüsselteile sicher aufbewahren, so daß keine ungewollte Mehrheit (im Sinne des Schwellwertes) den Schlüssel d_{Rech_Daten} rekonstruiert. Diese Instanzen bzw. ein gewisser Teil davon müssen vertrauenswürdig sein und müssen dem Rechner bekannt gemacht werden. Dies kann mit Hilfe des initialen Testprädikats erfolgen.

Das Hauptproblem besteht also bei der Initialisierung des initialen Testprädikats (t_{init}), weil der Rechner dem Besitzer dieses Prädikats vertraut und dadurch auch Mißbrauch möglich ist (weil dieser auch fremde Programme ratifiziert).

Im Regelfall entspricht der Besitzer von s_{init} dem Besitzer des Rechners, da dieser gekaufte Programme auch für lauffähig erklären möchte. Durch den Besitzer von s_{init} können weitere Signierfunktionen bzw. deren Besitzer für vertrauenswürdig erklärt werden und somit kann eine Hierarchie von vertrauenswürdigen Instanzen mit verschiedenen Kompetenzen realisiert werden.

Zur Initialisierung kann ein gewisser Speicherbereich (z.B. PROM) benutzt werden, in den der Besitzer des Rechners sein Testprädikat t_{init} speichern kann. Es muß jedoch ausgeschlossen werden, daß der Hersteller selbst ein zusätzliches „initiales“

Testprädikat definiert hat, denn sonst kann der Hersteller entsprechenden Mißbrauch betreiben, falls er den Rechner benutzen kann.

Wenn das initiale Testprädikat t_{init} in t_{neu} geändert werden soll (z.B. bei Verkauf des Rechners), können folgende Fälle unterschieden werden:

1) Eine gültige Änderungsanweisung liegt vor, d.h. sie ist mit s_{init} unterschrieben.
   - Wenn keine Daten und keine Programme weiterverwendet werden sollen, wird der Rechner neu initialisiert und t_{init} entsprechend geändert.
   - Wenn bestimmte Daten oder Programme weiterverwendet werden sollen, müssen diese in der Änderungsanweisung spezifiziert sein. Danach müssen die spezifizierten, mit s_{init} unterschriebenen Daten oder Programme innerhalb eines zuvor definierten Zeitraums mit s_{neu} unterschrieben werden. In diesem Zeitraum können keine weiteren Dokumente, die mit s_{init} unterschrieben sind, eingebracht werden. Nach diesen Zeitraum wird das Testprädikat t_{init} in t_{neu} geändert und alle Daten und Programme, die mit s_{init} unterschrieben sind, werden gelöscht.
2) Keine gültige Änderungsanweisung liegt vor; deshalb wir t_{init} nicht geändert.

6 Redundante Signierrechner

Die Verantwortungszuweisung innerhalb von Instanzen ist bei allen Realisierungen von Signierrechnern (ohne oder mit SW-Schutz) möglich. Um die Leistungsfähigkeit des Systems zu erhöhen oder fehlerhafte Signierrechner in einem Mehrrechnersystem zu tolerieren, sind redundante Signierrechner möglich. Man kann auch redundante Instanzen mit redundanter SW verwenden; dies wird hier nicht betrachtet.

Bei Signierrechnern treten einige Probleme auf, die bei herkömmlichen Rechnern nicht existieren. Eine Unterscheidung in Signierrechner mit bzw. ohne SW-Schutz ist möglich, und deshalb werden diese Fälle kurz betrachtet.

6.1 Redundante Signierrechner ohne SW-Schutz

Der Voter berechnet in einem solchen redundanten wie in herkömmlichen redundanten Systemen ebenfalls die Entscheidungsfunktion. Der Voter kann in HW oder SW realisiert sein und stellt selbst eine Instanz dar, wenn die Redundanzmaßnahmen nicht von einem Hersteller durchgeführt wurden. Aus diesem Grund muß der Voter die Testprädikate der redundanten Rechner kennen.

Ansonsten werden dieselben Funktionen wie in herkömmlichen Systemen ausgeführt, außer daß Dokumente für die Ausgabe benutzt werden.

6.2 Redundante Signierrechner mit SW-Schutz

Bei redundanten Signierrechnern, die SW-Schutz ermöglichen, treten wegen den Verschlüsselungen der Dokumente Probleme auf. Folgende zwei unterschiedliche Realisierungen mit redundanten Signierrechnern sind möglich:

- Die Redundanz wird vom Hersteller des Rechners realisiert. Deshalb kann ein solches System nach außen als ein einfacher Signierrechner erscheinen, denn der Voter kann die Eingaben aus den redundanten Subrechnern (z.B. Prozessor) als Klartext verarbeiten und die Entscheidungsfunktion berechnen. Die Ausgaben des Voters werden von ihm signiert und anschließend chiffriert. Der Voter sollte hierbei empfängerseitig (Verteiler) und senderseitig (eigentlicher Voter) realisiert sein, damit nach außen die interne Struktur verborgen ist. In diesem Fall ist der Voter keine Instanz, da die Verantwortung vom Hersteller des Rechners übernommen wird. Diese Realisierung ist für den Benutzer ein nicht-redundanter Signierrechner, der SW-Schutz ermöglicht, da er die Verwendung der internen Redundanz nicht zu kennen braucht.

- Mit Signierrechnern, die SW-Schutz ermöglichen, kann man ein redundantes System realisieren. Der Voter berechnet wiederum die Entscheidungsfunktion und deshalb muß er die Ausgaben der Signierrechner „verstehen“; sie müssen deshalb für den Voter chiffriert (mit c_{Voter}) werden. Der Voter ist in diesem Fall eine Instanz, er muß die Ausgaben signieren und Dokumente erzeugen.
  Die Realisierung des Voters kann hierbei sender- oder empfängerseitig erfolgen. Der Nachteil der empfängerseitigen Realisierung ist die mögliche Existenz von falschen Dokumenten, die bei der senderseitigen Realisierung durch die Entscheidungsfunktion eliminiert werden und nur noch im Präfix auftauchen.
  Bei der empfängerseitigen Realisierung wird eine Komponente benötigt, die die Eingaben an alle Signierrechner verteilt. Wenn alle Signierrechner denselben Schlüssel zur Chiffrierung der Daten benutzen, kann eine solche Komponente durch Vervielfachung der Leitungen realisiert werden. Falls sie verschiedene Schlüssel benutzen, muß diese Komponente die Daten für jeden Signierrechner separat verschlüsseln. In diesem Fall ist diese Komponente ebenfalls eine Instanz.

Ein solches Mehrrechner-Signiersystem ist in Bild 12 dargestellt, wobei **SR** die Abkürzung für Signierrechner ist.

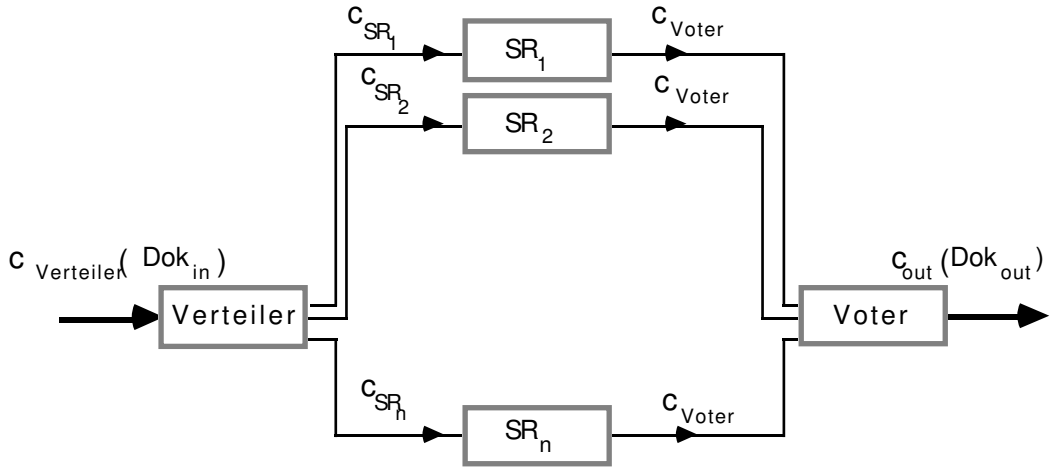

Bild 12: Mehrrechner-Signiersystem.

Das obige Rechnersystem kann im weitesten Sinne als offen betrachtet werden, da alle Dokumente verschlüsselt sind. Diese Konfiguration ist also auch in „unsicheren“ Umgebungen möglich.

Damit soll die Betrachtung der Instanzen und der Signierrechner abgeschlossen sein. Im folgenden soll eine Anwendung von solchen Systemen vorgestellt werden, bei denen die Zugriffskontrolle durch die ratifizierende Instanz und den Lader/Starter erfolgt.

7 Beispiel: Computerviren

In der letzten Zeit machten Computerviren Furore. Die Virenproblematik wurde zuerst von Cohen untersucht [Cohe_87]. Im deutschsprachigen Raum haben Dierstein [Dier_86], Hoffmann [Hoff_87] und Brunnstein [Bru1_87] ebenfalls Computerviren untersucht und mögliche Konsequenzen dargestellt. In Computerviren wird eine enorme Gefahr gesehen, da sie in der Lage sind, gesamte Computersysteme lahmzulegen, ohne daß der Verantwortliche, also der Virenprogrammierer, ermittelt werden kann.

Im folgenden werden Computerviren und ihre Eigenschaften betrachtet. Diese werden dann auf ein System aus Signierrechnern übertragen und untersucht, um zu zeigen, daß ihre Urheber in einem solchen System zur Verantwortung gezogen werden können. Wenn die Zugriffsmaßnahmen durch den Lader/Starter realisiert und bei der Speicherung ebenfalls die Unterschriften von Instanzen benutzt werden, können sie unwirksam gemacht werden.

7.1 Definition der Computerviren und deren Eigenschaften

Die Definition von Computerviren und deren Eigenschaften werden weitgehend aus [Cohe_87] und [Bru1_87] entnommen. Die folgende Definition eines Computervirus stammt aus [Bru1_87].

Definition Computervirus

Ein **Computervirus** ist ein *nicht eigenständig existierendes Programmstück*, welches, nachdem es in (mindestens) ein selbständig existierendes „*Wirtsprogramm*“ eingepflanzt wurde, die Fähigkeit besitzt, eine Kopie seiner selbst in (mindestens) ein weiteres, zuvor nicht infiziertes Programm einzubringen, und welches vor der Ausführung seines Wirtsprogramms eigene Anweisungen ausführen lassen kann.

Ein **Computervirus** enthält:

(a) ein **Infektions-Verfahren**, welches eine Kopie des Virus in andere Wirtsprogramme einbringen kann,

(b) ein **Wirkungs-Verfahren**, mit dem abhängig von vorgebbaren Bedingungen vor Ausführung des Wirtsprogramms eine gewünschte Wirkung erzielt werden kann.

Um das Virus effizienter zu machen, kann es noch weitere Komponenten besitzen, zum Beispiel:

(c) ein **Erkennungs-Verfahren**, mit dem festgestellt werden kann, ob ein betrachtetes Programm bereits das Virus enthält.

(d) Tarn-Techniken („Stealth-Viren“), die eine Analyse des Computervirus erschweren,

(e) eine Auslösebedingung zur Aktivierung des Infektions-Verfahrens (um bis dahin möglichst unerkannt zu bleiben),

u. a. m.

Die Ausbreitung des Virus erfolgt also durch Hinzukopieren seiner selbst oder einer funktional äquivalenten Version an sogenannte Wirtsprogramme. Aus dieser Eigenschaft geht hervor, daß sich Viren nur dann ausbreiten können, wenn die Programme als Daten interpretierbar sind. Eine erste Lösung der Virenproblematik ergäbe sich im Verbot, Programme als Daten zu interpretieren; hiermit wäre aber die Funktionalität des Systems zu eingeschränkt.

Ein Virus kann nur Programme infizieren, zu denen das Wirtsprogramm Schreibzugriff hat. Es kann also, da alle vom Virus-Wirtsprogramm schreib-zugreifbaren und damit infizierbaren Programme wiederum alle von ihnen schreib-zugreifbaren Programme infizieren können, usw., die gesamte transitive Hülle der schreibzugreifbaren Programme infizieren. Wenn Mehrfachbenutzung von Programmen mit verschiedenen Aufrufumgebungen erlaubt ist und die Zugriffsrechte vererbt werden, kann die transitive Hülle sehr groß werden, insbesondere wenn das Virus ein Betriebssystemprogramm infiziert hat. Solche Programme werden im allgemeinen selten

geändert und sollten deshalb auf Nur-Lese-Speichern gespeichert werden, um eine Modifikation zu verhindern.

Durch Isolation der einzelnen Benutzer kann die Ausbreitung der Viren begrenzt werden. Diese Lösung ist im allgemeinen nicht akzeptabel, da eine Kommunikation zwischen Benutzern für eine effiziente Bearbeitung ihrer Aufgaben (z.B. Programmentwicklung in Teamarbeit) notwendig ist.

Eine Lösung des Problems der Computerviren stellen Zugriffskontrollen nach dem *Prinzip der geringstmöglichen Privilegierung* dar, bei denen jedes Programm nur die zur Erledigung seiner Aufgabe benötigten Rechte erhält. In herkömmlichen Rechensystemen mit Zugriffsschutzmechanismen wird dieses Prinzip im allgemeinen nicht benutzt – üblicherweise werden alle Zugriffsrechte der Aufrufumgebung vererbt; deshalb sind auch solche Systeme in relativ kurzer Zeit verseucht [Cohe_87].

Es gelten folgende Sachverhalte [Cohe_87] über Computerviren, die eine Bekämpfung schwierig machen:

(i) Es ist **unentscheidbar**, ob ein gegebenes Programm ein Virus ist bzw. ob ein Programm ein anderes infiziert.

(ii) Es gibt sogenannte „*evolutionäre*“ Viren, die ihr Aussehen ändern, z.B. indem wirkungslose Anweisungen zwischen echte Anweisungen eingefügt werden.

(iii) Die Äquivalenz von zwei verschiedenen Programmen ist ebenfalls **unentscheidbar**. Deshalb können auch evolutionäre Viren nicht entdeckt werden.

Neben diesen frustrierenden theoretischen Resultaten gibt es einen weiteren negativen Fakt.

(iv) Es ist sehr schwer, wenn nicht gar unmöglich, ein Virus aus einem komplexen System zu entfernen, es sei denn durch Löschung aller Programme und der anschließenden Rekonstruktion des Programmbestandes aus (hoffentlich!) unverseuchten Sicherheitskopien.

Trotzdem gibt es Hoffnung, nämlich:

(v) Zu jedem nicht evolutionären Virus gibt es einen speziellen Detektionsmechanismus.

Eine weitere Infektion durch ein entdecktes nicht evolutionäres Virus kann verhindert werden, indem ein infiziertes Programm nicht mehr ausgeführt wird. Wenn der Hersteller des Virus diesen Detektionsmechanismus aber kennt, kann er das Virus so ändern, daß dieses nicht mehr erkannt wird. Somit muß für dieses neue Virus wiederum ein Detektionsmechanismus gefunden werden, um es unwirksam zu machen.

Eine weitere Ausbreitungsmöglichkeit für ein Virus besteht darin, daß das Virus nicht den Code eines Programms manipuliert, sondern dessen Zwischenstufen (z.B. den Quellcode). Dadurch wird das Virus erst durch die Entwicklung zu einem lauffähigen Programm aktiv und ist so auf „quasi legale“ Weise entstanden. Die Entdeckungsgefahr für das Virus ist aber größer, da ein solches Virus in den Zwischenstufen (z.B. im Quellcode) durch den Hersteller des Programms entdeckt werden kann.

Als Resultat der Analyse von Computerviren ergeben sich folgende Schutzmaßnahmen:

- Mit Zugriffskontrollen, die eine feine Granularität der Zugriffsrechte benutzen und nur die unbedingt notwendigen Schreibzugriffsrechte für das Wirtsprogramm vergeben, kann die Ausbreitung von Computerviren unterbunden werden.

- Perfekter Schutz ist leicht durch Isolation zu erreichen; diese Isolation ist aber für allgemeine Systeme, in denen gemeinsam benutzte Programme vorhanden sind, nicht akzeptabel.
- Die Vermeidung von Infektion kann durch stark eingeschränkte Dienste erreicht werden.
- Statistische Methoden, die auf Größe und Zeit basieren, können benutzt werden, um eine Ausbreitung zu vermeiden; dazu werden Verhaltensprofile erzeugt und diese mit Soll-Verhaltensprofilen korreliert [Denn_86].
- Während des Programmablaufs werden von einer Überwachungshardware Kontrollwerte berechnet, die vom genauen Programmablauf abhängen. An den Verzweigungspunkten des Programms werden sie von der Überwachungshardware mit den vom Übersetzer und Binder (Linker) berechneten verglichen, so daß Änderungen durch Fehler oder Computerviren erkannt werden [JoAv_88].

In einem physisch perfekt geschützten, zentralisierten System würden feine Zugriffskontrollen genausoviel erreichen wie das beschriebene und im folgenden bzgl. seiner Auswirkungen auf Computerviren untersuchte Unterschriftensystem.

7.2 Computerviren in einem Instanzensystem

In diesem Kapitel werden die Computerviren auf Instanzen übertragen und ihre Wirkungsweise wird näher betrachtet.

Ein Computervirus sei in einer Instanz (Wirtsprogramm) enthalten, die ihre Ausgaben mit s_{CV} signiere. Im folgenden wird das Wirtsprogramm und das Virus als eine Einheit betrachtet, die mit **CV** bezeichnet wird. Ein solches CV wurde von einer ratifizierenden Instanz für lauffähig erklärt und somit existiert ein Verantwortlicher für CV (die ratifizierende Instanz kann den Hersteller von CV zur Verantwortung ziehen).

Das CV möge im folgenden sein Infektions-Verfahren durchführen. Es sind zwei Fälle zu unterscheiden:

a) CV ist keine ratifizierende Instanz.
b) CV ist ratifizierende Instanz.

Das Computervirus möge beliebige Programme modifizieren und somit infizieren. Da aber alle Ausgaben von den Instanzen unterschrieben werden, muß das modifizierte Programm-Dokument eine gültige Unterschrift enthalten.

a) Wenn das CV keine ratifizierende Instanz ist, können diese modifizierten Programme nicht gestartet werden, weil hierzu eine gültige Unterschrift einer ratifizierenden Instanz fehlt.

b) Falls es sich um ein ratifizierendes Virus handelt, das die Eigenschaft, Programme ratifizieren zu können, vererben kann, breitet sich das Virus wie in herkömmlichen Systemen aus. Deshalb werden im folgenden zwei Klassen von ratifizierenden Instanzen untersucht und es werden Maßnahmen dargestellt, mit denen eine Ausbreitung verhindert werden kann.

1) CV ist ratifizierende Instanz, die die Eigenschaft, Programme ratifizieren zu können, nicht vererben kann, und
2) CV ist ratifizierende Instanzen, die obige Eigenschaft vererben kann.

1) Wenn das CV eine ratifizierende Instanz ist, sind die infizierten Programme lauffähig und stellen CV vom Typ a) dar.

2) Die ratifizierenden Instanzen mit der Eigenschaft der Vererbung müssen während ihrer gesamten Existenz frei von Viren sein; deshalb sollten Techniken zur Vermeidung „allmächtiger“ Einzelner, beispielsweise in der Rolle als Systemadministrator, [Cles_88 Kapitel 2.2] angewendet werden, um die Entstehung von Computerviren zu verhindern. Die ratifizierenden Instanzen bzw. deren Signierfunktionen müssen geschützt werden, damit kein Virus eingebracht werden kann.

Somit stellt ein Computervirus in einem System aus Instanzen, in dem eine *ausreichende* Zugriffskontrolle durch ratifizierende Instanzen und Lader/Starter verwirklicht ist, keine wesentliche Gefahr dar, da es keine anderen Programme wirksam infizieren kann, wenn die Mächtigkeit der ratifizierenden Instanz entsprechend eingeschränkt ist bzw. nur Computerviren-freie Instanzen ratifizieren. (Signierrechner leisten also bzgl. der Verhinderung der Ausbreitung von Computerviren genausoviel, wie das in [Denn_82 Seite 317f, PoGr_87] vorgeschlagene digitale Unterschreiben von Programmen. Bezüglich der Bestimmung von Verantwortung leistet das hier vorgeschlagene Signieren aller Ausgaben jedoch mehr.)

Die Gefahr durch Trojanische Pferde, insbesondere transitive [Pfit_89], kann nicht eliminiert werden. Auch deren Hersteller können jedoch zur Verantwortung gezogen werden, wenn ein abnormales Verhalten entdeckt wurde.

8 Resümee und Ausblick

1) Verantwortung in IT-Systemen *kann* hergestellt werden.
2) Digitale Signaturen erlauben *Beweissicherung* statt *Anscheinsspeicherung*.
3) Durch den technischen Fortschritt wird das Beschriebene *realisierbar*.

Natürlich gibt dieser Artikel weder Antworten auf alle sich aufdrängenden Fragen, noch löst das vorgeschlagene Rechnerkonzept alle Sicherheitsprobleme:

Für verschiedene Anwendungen von IT-Systemen wäre zu untersuchen, wie umfangreich Schnittstellenprotokollierung sein muß (unter anderem inwieweit innere Zustände von Signierrechnern mit protokolliert werden müssen), was die Protokollierung an Rechenzeit, Kommunikation und Speicher kostet, wie genau die Verfahren zur

Fehlerdiagnose und Verantwortungszuweisung aussehen, was ihre Zeitkomplexität ist, wie Protokollierung und Verifikation zusammenspielen sollten. Kurzum: Wir wissen heute nicht, für welche Anwendungen die skizzierten Verfahren ab wann anwendbar sein werden. Andererseits sind grundsätzliche Alternativen zum Skizzierten schwer vorstellbar. Dieser Artikel ist also nur ein Anfang. Wir würden uns über Mit- und Weiterarbeit sehr freuen!

Noch so schön signierte Protokolle nützen nichts, wenn sie im Fehler- oder Angriffsfall verloren gehen. Hiergegen sind physische Maßnahmen zu ergreifen, die nie perfekt sein können. Außerdem hilft das vorgeschlagene Rechnerkonzept prinzipiell nur bei erkannten „Fehlern“ und befriedigend auch nur bei solchen, deren Auswirkungen sich rückgängig machen lassen, vgl. [Pfit5_91 Kap. 5]. Das vorgeschlagene Rechnerkonzept muß also mit weiteren Schutzmaßnahmen kombiniert werden, was problemlos möglich ist.

Inzwischen wurde eine realisierungsnähere Arbeit publiziert, die insbesondere einige Aspekte von Kapitel 4 dieser Arbeit vertieft behandelt [Groß_91].

Ein Dankeschön

Für Kritik an und Anregungen zu diesem Text danken wir Manfred Böttger, Birgit Pfitzmann und Michael Waidner herzlich.

Literatur

BlGo_85 Manuel Blum, Shafi Goldwasser: An Efficient Probabilistic Public-Key Encryption Scheme Which Hides All Partial Information; Crypto '84, LNCS 196, Springer-Verlag, Berlin 1985, 289-299.

BlPW_91 Gerrit Bleumer, Birgit Pfitzmann, Michael Waidner: A remark on a signature scheme where forgery can be proved; Eurocrypt '90, LNCS 473, Springer-Verlag, Berlin 1991, 441-445.

Bras_88 Gilles Brassard: Modern Cryptology - A Tutorial; LNCS 325, Springer-Verlag, Berlin 1988.

BrSt_90 Ernest F. Brickell, D. R. Stinson: The Detection of Cheaters in Threshold Schemes; Crypto '88, LNCS 403, Springer-Verlag, Berlin 1990, 564-577.

Bru1_87 Klaus Brunnstein: Über Viren, Würmer und anderes seltsames Getier in Computer-Systemen; ein kleines "Informatik-Bestiarium"; Angewandte Informatik 29/10 (1987) 397-402.

Chau_84 David Chaum: Design Concepts for Tamper Responding Systems; Crypto '83, Plenum Press, New York 1984, 387-392.

Cles_88 Wolfgang Clesle: Schutz auch vor Herstellern und Betreibern von Informationssystemen; Diplomarbeit am Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, Juni 1988.

Cohe_87 Fred Cohen: Computer Viruses – Theory and Experiments; Computers & Security 6/1 (1987) 22-35.

Damg_88 Ivan Bjerre Damgård: Collision free hash functions and public key signature schemes; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 203-216.

DaPr_89 D. W. Davies, W. L. Price: Security for Computer Networks, An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer; (2nd ed.) John Wiley & Sons, New York 1989.

Denn_82 Dorothy E. Denning: Cryptography and Data Security; Addison-Wesley, Reading 1982; Reprinted with corrections, January 1983.

Denn_85 Dorothy E. Denning: Commutative Filters for Reducing Inference Threats in Multilevel Database Systems; Proc. 1985 Symposium on Security and Privacy, April 22-24, 1985, Oakland, California, IEEE Computer Society, 134-146.

Denn_86 Dorothy E. Denning: An Intrusion-Detection Model; Proc. 1986 IEEE Symp. on Security and Privacy, April 7-9, 1986, Oakland, California, 118-131.

Dier_86 Rüdiger Dierstein: Computerviren; Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt e. V. (DFVLR), Zentrale Datenverarbeitung, D-8031 Oberpfaffenhofen; Institutsbericht IB 562/6, Juli 1986.

DiHe_76 Whitfield Diffie, Martin E. Hellman: New Directions in Cryptography; IEEE Transactions on Information Theory 22/6 (1976) 644-654.

FoPf_91 Dirk Fox, Birgit Pfitzmann: Effiziente Software-Implementierung des GMR-Signatursystems; Proc. Verläßliche Informationssysteme (VIS'91), März 1991, Darmstadt, Informatik-Fachberichte 271, Springer-Verlag, Heidelberg 1991, 329-345.

Gold_87 Oded Goldreich: Towards a Theory of Software Protection and Simulation by Oblivious RAMs (Extended Abstract); 19th STOC 1987, 182-194.

GoMi_84 Shafi Goldwasser, Silvio Micali: Probabilistic Encryption; Journal of Computer and System Sciences 28 (1984) 270-299.

GoMR_88 Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM J. Comput. 17/2 (1988) 281-308.

Groß_91 Michael Groß: Vertrauenswürdiges Booten als Grundlage authentischer Basissysteme; Proc. Verläßliche Informationssysteme (VIS'91), März 1991, Darmstadt, Informatik-Fachberichte 271, Springer-Verlag, Heidelberg 1991, 190-207.

Hoff_87 Frank Hoffmeister: Ein Ansatz zur Abwehr von Computerviren; Organisation und Betrieb der verteilten Datenverarbeitung, 7. GI-Fachgespräch über Rechenzentren, IFB 134, Springer-Verlag, Berlin 1987, 101-110.

Home_?? Homer: Ilias; Diogenes Taschenbuch detebe 20779, übersetzt aus dem Altgriechischen von Heinrich Voß, herausgegeben von Peter Von der Mühll; Diogenes Verlag, Zürich 1980.

Horg_85 John Horgan: Thwarting the information thieves; IEEE spectrum 22/7 (1985) 30-41.

Hors_85 Patrick Horster: Kryptologie; Reihe Informatik/47, Herausgegeben von K. H. Böhling, U. Kulisch, H. Maurer, Bibliographisches Institut, Mannheim, 1985.

JoAv_88 Mark K. Joseph, Algirdas Avizienis: A fault tolerant approach to computer viruses; Proc. 1988 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Washington 1988, 52-58.

Pfit_89 Andreas Pfitzmann: Diensteintegrierende Kommunikationsnetze mit teilnehmerüberprüfbarem Datenschutz; Universität Karlsruhe, Fakultät für Informatik, Dissertation, IFB 234, Springer-Verlag, Heidelberg 1990.

Pfit_90 Andreas Pfitzmann: Entwicklungslinien der Informationstechnik und Informatik und ihre Auswirkungen auf rechtliche Beherrschung; Datenschutz und Datensicherung DuD 14/12 (1990) 620-627.

Pfit5_91 Andreas Pfitzmann: Zur Verantwortung der Informatiker; Datenschutz-Berater 15/8-9 (1991).

PfWa_91 Birgit Pfitzmann, Michael Waidner: Fail-stop-Signaturen und ihre Anwendung; Proc. Verläßliche Informationssysteme (VIS'91), März 1991, Darmstadt, Informatik-Fachberichte 271, Springer-Verlag, Heidelberg 1991, 289-301.

PoGr_87 Maria M. Pozzo, Terence E. Gray: An Approach to Containing Computer Viruses; Computers & Security 6/4 (1987) 321-331.

PWP_90 Birgit Pfitzmann, Michael Waidner, Andreas Pfitzmann: Rechtssicherheit trotz Anonymität in offenen digitalen Systemen; Datenschutz und Datensicherung DuD 14/5-6 (1990) 243-253, 305-315.

RSA_78 R. L. Rivest, A. Shamir, L. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems; Communications of the ACM 21/2 (1978) 120-126, reprinted: 26/1 (1983) 96-99.

Sham_79 Adi Shamir: How to Share a Secret; Communications of the ACM 22/11 (1979) 612-613.

ToWo_88 Martin Tompa, Heather Woll: How to Share a Secret with Cheaters; Journal of Cryptology 1/2 (1988) 133-138.

Wein_87 Steve H. Weingart: Physical Security for the microABYSS System; Proc. 1987 IEEE Symp. on Security and Privacy, April 27-29, 1987, Oakland, California, 52-58.

WhCo_87 Steve R. White, Liam Comerford: ABYSS: A Trusted Architecture for Software Protection; Proc. 1987 IEEE Symp. on Security and Privacy, April 27-29, 1987, Oakland, California, 38-51.